www.mientayvn.com

Khi đọc qua tài liệu này, nếu phát hiện sai sót hoặc nội dung kém chất lượng xin hãy thông báo để chúng tôi sửa chữa hoặc thay thế bằng một tài liệu cùng chủ đề của tác giả khác. Tài liệu này bao gồm nhiều tài liệu nhỏ có cùng chủ đề bên trong nó. Phần nội dung bạn cần có thể nằm ở giữa hoặc ở cuối tài liệu này, hãy sử dụng chức năng Search để tìm chúng.

Bạn có thể tham khảo nguồn tài liệu được dịch từ tiếng Anh tại đây:

http://mientayvn.com/Tai_lieu_da_dich.html

Thông tin liên hệ:

Yahoo mail: thanhlam1910_2006@yahoo.com

Gmail: frbwrthes@gmail.com

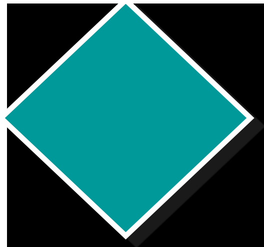
Theo yêu cầu của khách hàng, trong một năm qua, chúng tôi đã dịch qua 16 môn học, 34 cuốn sách, 43 bài báo, 5 sổ tay (chưa tính các tài liệu từ năm 2010 trở về trước) Xem ở đây

DỊCH VỤ DỊCH TIẾNG ANH CHUYÊN NGÀNH NHANH NHẤT VÀ CHÍNH XÁC NHẤT

Chỉ sau một lần liên lạc, việc dịch được tiến hành

Giá cả: có thể giảm đến 10 nghìn/1 trang

Chất lượng: Tạo dựng niềm tin cho khách hàng bằng công nghệ 1.Bạn thấy được toàn bộ bản dịch; 2.Bạn đánh giá chất lượng. 3.Bạn quyết định thanh toán.



CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT 1

Số tiết lý thuyết: **45**

Số tiết thực hành: **30**

Tài Liệu Tham Khảo

- Trần Hạnh Nhi, Dương Anh Đức. ***Giáo trình Cấu Trúc Dữ Liệu 1***, ĐHQG Tp. HCM, 2000.
- Robert Sedgewick. ***Cẩm nang thuật toán*** (bản dịch của nhóm tác giả ĐH KHTN), NXB Khoa học kỹ thuật, 1994.
- P. S. Deshpande, O. G. Kakde. ***C & Data Structures,*** 2004.
- Dr. Dobb's. ***Algorithms and Data Structures***, 1999
- A.V. Aho, J.E Hopcroft, J.D Ullman. ***Data structures and Algorithms,*** Addison Wesley, 1983.

Nội Dung Chương Trình

- *Buổi 1*: Giới thiệu về CTDL & Giải Thuật.

  Các thuật toán tìm kiếm.
- *Buổi 2*: Interchange Sort, Selection Sort, Bubble Sort, Insertion Sort.
- *Buổi 3*: Shaker Sort, Shell Sort, Heap Sort.
- *Buổi 4*: Quick Sort, MergeSort, Radix Sort.
- *Buổi 5*: Cấu trúc động, Danh sách liên kết đơn.

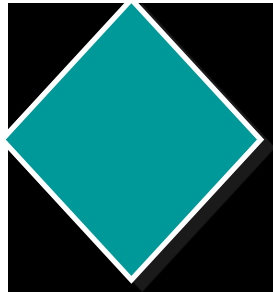
Nội Dung Chương Trình

- *Buổi 6*: Stack, Queue.
- *Buổi 7*: Danh sách liên kết kép.
- *Buổi 8*: Cây, Cây nhị phân, cây nhị phân tìm kiếm.
- *Buổi 9*: Cây cân bằng (AVL).
- *Buổi 10*: Các CTDL mở rộng.
- *Buổi 11*: Ôn tập.

Hình Thức Thi

- Giữa kỳ: **2 điểm (giấy)**
- Cuối kỳ: **8 điểm**
  - Lý thuyết: **Thi trên giấy (5 điểm)**
  - Thực hành: **Viết CT (3 điểm)**
- Bài cộng thêm điểm:

  Seminar, vấn đáp. Tối đa 2 điểm.
- Tổng điểm: **10 điểm.**



TỔNG QUAN VỀ CTDL VÀ THUẬT TOÁN

Nội Dung

- Tổng quan về CTDL và thuật toán
- Các tiêu chuẩn của CTDL
- Vai trò của CTDL
- Độ phức tạp của thuật toán
- Thực hiện và hiệu chỉnh chương trình
- Tiêu chuẩn của chương trình

Khái Niệm Về CTDL Và Thuật Toán

- Niklaus Wirth:

  CTDL + Thuật toán = Chương trình

- Cần nghiên cứu về thuật toán và CTDL!

Sự Cần Thiết Của Thuật Toán

- Tại sao sử dụng máy tính để xử lý dữ liệu?
  - Nhanh hơn.
  - Nhiều hơn.
  - Giải quyết những bài toán mà con người không thể hoàn thành được.
- Làm sao đạt được những mục tiêu đó?
  - Nhờ vào sự tiến bộ của kỹ thuật: tăng cấu hình máy ⇨ chi phí cao ☹
  - Nhờ vào các thuật toán hiệu quả: thông minh và chi phí thấp ☺

"Một máy tính siêu hạng vẫn không thể cứu vãn một thuật toán tồi!"

Thuật Toán

- **Thuật toán**: Một dãy hữu hạn các chỉ thị có thể thi hành để đạt mục tiêu đề ra nào đó.
- **Ví dụ**: Thuật toán tính tổng tất cả các số nguyên dương nhỏ hơn n gồm các bước sau:

Bước 1: *S=0, i=1;*

Bước 2: *nếu i<n thì s=s+i;*

Ngược lại: qua bước 4;

Bước 3:

i=i+1;

Quay lại bước 2;

Bước 4: *Tổng cần tìm là S.*

Các Tiêu Chuẩn Của Thuật Toán

- Xác định
- Hữu hạn
- Đúng
- Tính hiệu quả
- Tính tổng quát

Biễu Diễn Thuật Toán

- Dạng ngôn ngữ tự nhiên
- Dạng lưu đồ (sơ đồ khối)
- Dạng mã giả
- Ngôn ngữ lập trình

Biểu Diễn Bằng Ngôn Ngữ Tự Nhiên

- NN tự nhiên thông qua các bước được tuần tự liệt kê để biễu diễn thuật toán.
- Ưu điểm:
  - Đơn giản, không cần kiến thức về về cách biểu diễn (mã giả, lưu đồ,...)
- Nhược điểm:
  - Dài dòng, không cấu trúc.
  - Đôi lúc khó hiểu, không diễn đạt được thuật toán.

Lưu Đồ

- Là hệ thống các nút, cung hình dạng khác nhau thể hiện các chức năng khác nhau.

A

Thực hiện A

A

Gọi hàm A

Vào / Ra dữ liệu

B

Đúng

Sai

Điều kiện rẻ nhánh B

Begin

End

Nút giới hạn bắt đầu / kết thúc chương trình

Biểu Diễn Bằng Lưu Đồ

Bắt đầu

$a_{max} = a_0$

$i = 1$

$i<n$

S

a_{max} là lớn nhất

Kết thúc

Đ

$a_{max} < a_i$

Đ

$a_{max} = a_i$

S

$i = i+1$

Tìm phần tử mang giá trị lớn nhất trong mảng

Biểu Diễn Bằng Mã Giả

- Ngôn ngữ tựa ngôn ngữ lập trình:
  - Dùng cấu trúc chuẩn hóa, chẳng hạn tựa Pascal, C.
  - Dùng các ký hiệu toán học, biến, hàm.
- Ưu điểm:
  - Đỡ cồng kềnh hơn lưu đồ khối.
- Nhược điểm:
  - Không trực quan bằng lưu đồ khối.

Biểu Diễn Bằng Mã Giả

- **Một số quy ước**
  1. Các biểu thức toán học
  2. Lệnh gán: "=" (A←B)
  3. So sánh: "==", "!="
  4. Khai báo hàm (thuật toán)

     Thuật toán *<tên TT> (<tham số>)*

     Input*: <dữ liệu vào>*

     Output*: <dữ liệu ra>*

     <Các câu lệnh>

     End

Biểu Diễn Bằng Mã Giả

5. Các cấu trúc:

Cấu trúc chọn:

if … **then** … [**else** …] **fi**

Vòng lặp:

while … **do**

do … **while** (…)

for … **do** … **od**

6. Một số câu lệnh khác:

Trả giá trị về: **return** [giá trị]

Lời gọi hàm: <Tên>(tham số)

Biểu Diễn Bằng Mã Giả

❖ **Ví dụ**: Tìm phần tử lớn nhất trong mảng một chiều.

$a_{max}=a_0$;

i=1;

while (i<n)

  if ($a_{max}<a_i$) $a_{max} = a_i$;

   i++;

end while;

Biểu Diễn Bằng Ngôn Ngữ Lập Trình

- Dùng ngôn ngữ máy tính (C, Pascal,...) để diễn tả thuật toán, CTDL thành câu lệnh.
- Kỹ năng lập trình đòi hỏi cần học tập và thực hành (nhiều).
- Dùng phương pháp tinh chế từng bước để chuyển hoá bài toán sang mã chương trình cụ thể.

Độ Phức Tạp Của Thuật Toán

- Một thuật toán hiệu quả:
  - Chi phí cần sử dụng tài nguyên thấp: Bộ nhớ, thời gian sử dụng CPU, …
- Phân tích độ phức tạp thuật toán:
  - **N** là khối lượng dữ liệu cần xử lý.
  - Mô tả độ phức tạp thuật toán qua một hàm **f(N)**.
  - Hai phương pháp đánh giá độ phức tạp của thuật toán:
    - Phương pháp thực nghiệm.
    - Phương pháp xấp xỉ toán học.

Phương Pháp Thực Nghiệm

- Cài thuật toán rồi chọn các bộ dữ liệu thử nghiệm.
- Thống kê các thông số nhận được khi chạy các bộ dữ liệu đó.
- *Ưu điểm*: Dễ thực hiện.
- *Nhược điểm*:
  - Chịu sự hạn chế của ngôn ngữ lập trình.
  - Ảnh hưởng bởi trình độ của người lập trình.
  - Chọn được các bộ dữ liệu thử đặc trưng cho tất cả tập các dữ liệu vào của thuật toán: khó khăn và tốn nhiều chi phí.
  - Phụ thuộc vào phần cứng.

Phương Pháp Xấp Xỉ

- Đánh giá giá thuật toán theo hướng tiệm xấp xỉ tiệm cận qua các khái niệm O().
- *<u>Ưu điểm</u>*: Ít phụ thuộc môi trường cũng như phần cứng hơn.
- *<u>Nhược điểm</u>*: Phức tạp.
- Các trường hợp độ phức tạp quan tâm:
  - Trường hợp tốt nhất (phân tích chính xác)
  - Trường hợp xấu nhất (phân tích chính xác)
  - Trường hợp trung bình (mang tích dự đoán)

Sự Phân Lớp Theo Độ Phức Tạp Của Thuật Toán

- **Sử dụng ký hiệu BigO**
  - Hằng số : O(c)
  - logN : O(logN)
  - N : O(N)
  - NlogN : O(NlogN)
  - N^2 : $O(N^2)$
  - N^3 : $O(N^3)$
  - 2^N : $O(2^N)$
  - N! : O(N!)

Độ phức tạp tăng dần

Dữ Liệu

- Theo *từ điển Tiếng Việt*: số liệu, tư liệu đã có, được dựa vào để giải quyết vấn đề

- *Tin học:* Biểu diễn các thông tin cần thiết cho bài toán.

Cấu Trúc Dữ Liệu

- Cách tổ chức lưu trữ dữ liệu.
- Các tiêu chuẩn của CTDL:
  - Phải biểu diễn đầy đủ thông tin.
  - Phải phù hợp với các thao tác trên đó.
  - Phù hợp với điều kiện cho phép của NNLT.
  - Tiết kiệm tài nguyên hệ thống.

Vai Trò Của Cấu Trúc Dữ Liệu

- Cấu trúc dữ liệu đóng vai trò quan trọng trong việc kết hợp và đưa ra cách giải quyết bài toán.
- CTDL hỗ trợ cho các thuật toán thao tác trên đối tượng được hiệu quả hơn

Thực Hiện Và Hiệu Chỉnh Chương Trình

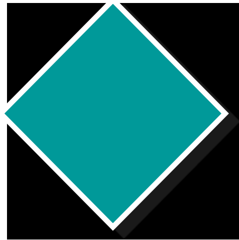
- Chạy thử.
- Lỗi và cách sửa:
  - Lỗi thuật toán.
  - Lỗi trình tự.
  - Lỗi cú pháp.
- Xây dựng bộ test.
- Cập nhật, thay đổi chương trình theo yêu cầu (mới).

Tiêu Chuẩn Của Một Chương Trình

- Tính tin cậy
  - Giải thuật + Kiểm tra cài đặt
- Tính uyển chuyển
  - Đáp ứng quy trình làm phần mềm.
- Tính trong sáng
  - Dễ hiểu và dễ chỉnh sửa
- Tính hữu hiệu.
  - Tài nguyên + giải thuật

Quy Trình Làm Phần Mềm

- *Bước 0:* Ý tưởng (concept).
- *Bước 1:* Xác định yêu cầu (Requirements Specification).
- *Bước 2:* Phân tích (Analysis).
- *Bước 3:* Thiết kế (Design).
- *Bước 4:* Cài đặt (Implementation).
- *Bước 5:* Thử nghiệm (Testing).
- *Bước 6:* Vận hành, theo dõi và bảo dưỡng (Operation, follow-up and Maintenance).



TÌM KIẾM VÀ SẮP XẾP NỘI

Nội Dung

- Các giải thuật tìm kiếm nội
  1. Tìm kiếm tuyến tính
  2. Tìm kiếm nhị phân
- Các giải thuật sắp xếp nội
  1. Đổi chỗ trực tiếp – Interchange Sort
  2. Chọn trực tiếp – Selection Sort
  3. Nổi bọt – Bubble Sort

Nội Dung (Tt)

4. Chèn trực tiếp – Insertion Sort
5. Chèn nhị phân – Binary Insertion Sort
6. Shaker Sort
7. Shell Sort
8. Heap Sort
9. Quick Sort
10. Merge Sort
11. Radix Sort

Bài Toán Tìm Kiếm

- Cho danh sách có n phần tử a_0, a_1, a_2..., a_{n-1}.
- Để đơn giản trong việc trình bày giải thuật ta dùng mảng 1 chiều a để lưu danh sách các phần tử nói trên trong bộ nhớ chính.
- Tìm phần tử có khoá bằng X trong mảng
  - Giải thuật tìm kiếm tuyến tính (tìm tuần tự)
  - Giải thuật tìm kiếm nhị phân

❖ ***Lưu ý***: *Trong quá trình trình bày thuật giải ta dùng ngôn ngữ lập trình C.*

Tìm Kiếm Tuyến Tính

- **Ý tưởng** : So sánh X lần lượt với phần tử thứ 1, thứ 2,…của mảng a cho đến khi gặp được khóa cần tìm, hoặc tìm hết mảng mà không thấy.
- **Các bước tiến hành**
  - Bước 1: Khởi gán i=0;
  - Bước 2: So sánh a[i] với giá trị x cần tìm, có 2 khả năng

    + a[i] == x tìm thấy x. Dừng;

    + a[i] != x sang bước 3;
  - Bước 3: i=i+1 // Xét tiếp phần tử kế tiếp trong mảng

    Nếu i==N: Hết mảng. Dừng;

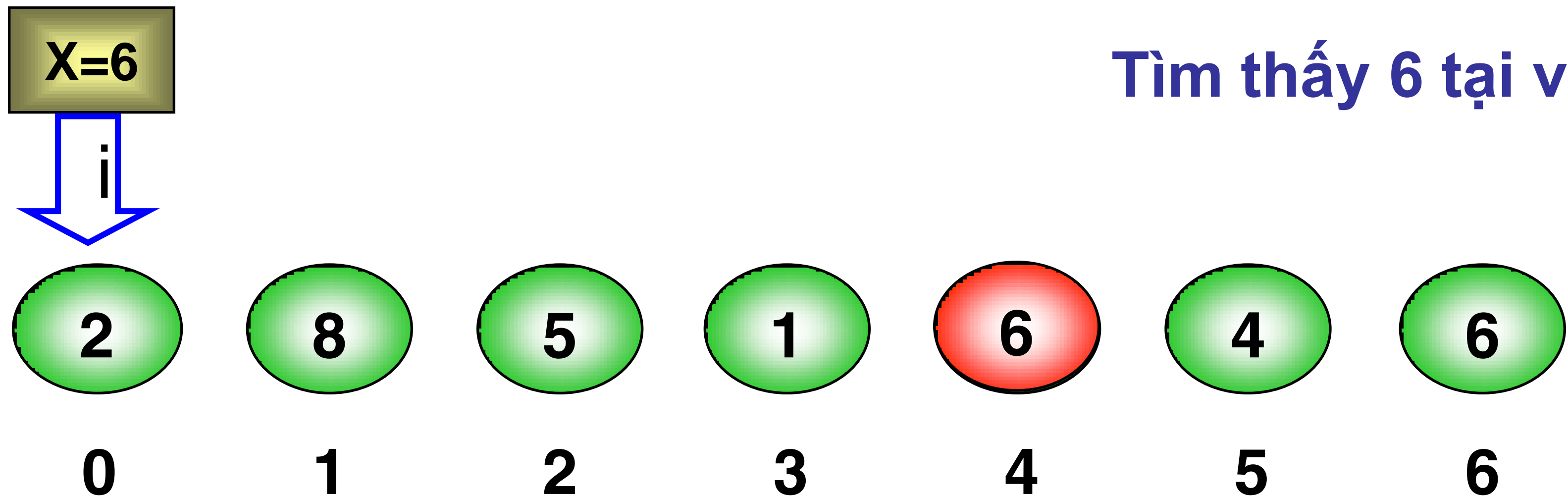
    Ngược lại: Lặp lại bước 2;

Thuật Toán Tìm Kiếm Tuyến Tính

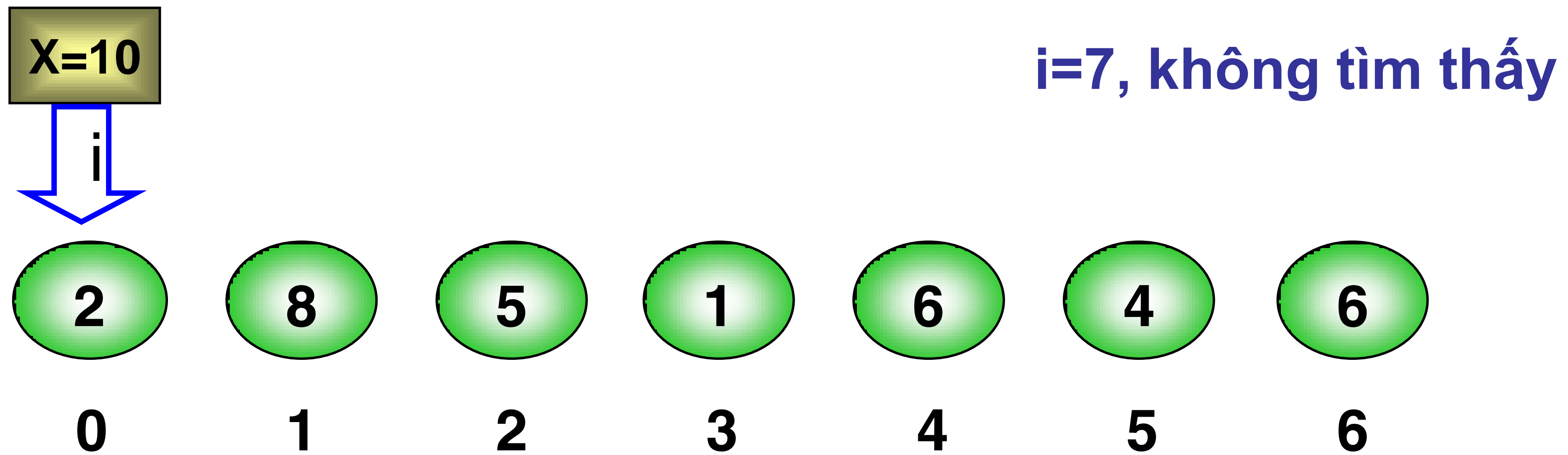
- Hàm trả về 1 nếu tìm thấy, ngược lại trả về 0:

```
int LinearSearch(int a[],int n, int x)
{
        int i=0;
        while((i<n)&&(a[i]!=x))
                i++;
        if(i==n)
                return 0; //Tìm không thấy x
        else
                return 1; //Tìm thấy
}
```

Minh Họa Thuật Toán Tìm Kiếm Tuyến Tính



Minh Họa Thuật Toán Tìm Kiếm Tuyến Tính (tt)



Đánh Giá Thuật Toán Tìm Tuyến Tính

Trường hợp	Css
Tốt nhất	1
Xấu nhất	N
Trung bình	(N+1) / 2

➢ Độ phức tạp O(N)

Cải Tiến Thuật Toán Tìm Tuyến Tính

- Nhận xét: Số phép so sánh của thuật toán trong trường hợp xấu nhất là 2*n.
- Để giảm thiểu số phép so sánh trong vòng lặp cho thuật toán, ta thêm phần tử "lính canh" vào cuối dãy.

```
int LinearSearch(int a[],int n, int x)
{           int i=0; a[n]=x;   // a[n] là phần tử "lính canh"
            while(a[i]!=x)
                  i++;
            if(i==n)
                  return 0; // Tìm không thấy x
            else
                  return 1; // Tìm thấy
}
```

Thuật Toán Tìm Kiếm Nhị Phân

- Được áp dụng trên mảng đã có thứ tự.
- **Ý tưởng**: .
  - Giả xử ta xét mảng có thứ tự tăng, khi ấy ta có $a_{i-1}<a_i<a_{i+1}$
  - Nếu $X>a_i$ thì X chỉ có thể xuất hiện trong đoạn $[a_{i+1}, a_{n-1}]$
  - Nếu $X<a_i$ thì X chỉ có thể xuất hiện trong đoạn $[a_0, a_{i-1}]$
  - Ý tưởng của giải thuật là tại mỗi bước ta so sánh X với phần tử đứng giữa trong dãy tìm kiếm hiện hành, dựa vào kết quả so sánh này mà ta quyết định giới hạn dãy tìm kiếm ở nữa dưới hay nữa trên của dãy tìm kiếm hiện hành.

Các Bước Thuật Toán Tìm Kiếm Nhị Phân

- Giả sử dãy tìm kiếm hiện hành bao gồm các phần tử nằm trong a_{left}, a_{right}, các bước của giải thuật như sau:
- Bước 1: left=0; right=N-1;
- Bước 2:
  - mid=(left+right)/2; *//chỉ số phần tử giữa dãy hiện hành*
  - So sánh a[mid] với x. Có 3 khả năng
    - a[mid]= x: tìm thấy. Dừng
    - a[mid]>x : Right= mid-1;
    - a[mid]<x : Left= mid+1;
- Bước 3: Nếu Left <=Right ; *// còn phần tử trong dãy hiện hành*

  + Lặp lại bước 2

  Ngược lại : Dừng

Cài Đặt Thuật Toán Tìm Nhị Phân

- Hàm trả về giá trị 1 nếu tìm thấy, ngược lại hàm trả về giá trị 0

```
int BinarySearch(int a[],int n,int x)
{    int left, right, mid; left=0; right=n-1;
     do{
               mid=(left+right)/2;
               if(a[mid]==x) return 1;
               else    if(a[mid]<x)  left=mid+1;
                       else  right=mid-1;
     }while(left<=right);
     return 0;
}
```

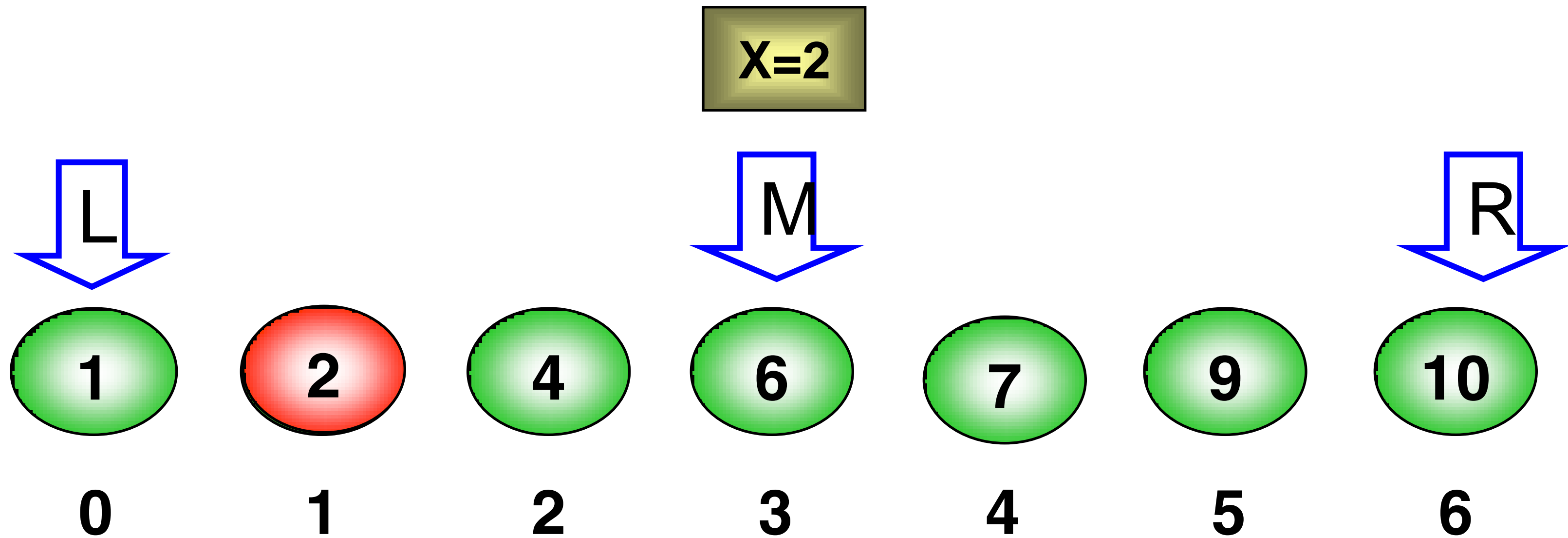
Đánh Giá Thuật Toán Tìm Tuyến Tính

Trường hợp	Css
Tốt nhất	1
Xấu nhất	$\log_2 N$
Trung bình	$\log_2 N / 2$

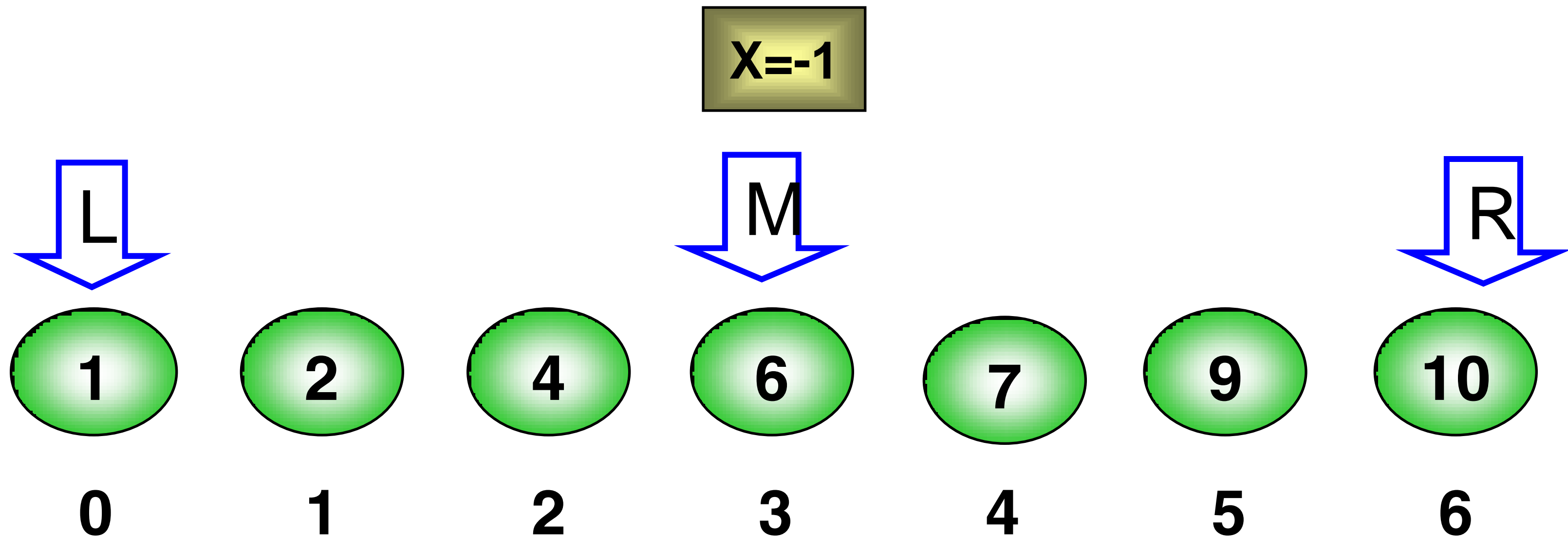
➢ Độ phức tạp $O(\log_2 N)$

Minh Họa Thuật Toán Tìm Nhị Phân

Tìm thấy 2 tại vị trí 1



Minh Họa Thuật Toán Tìm Nhị Phân (tt)



L=0

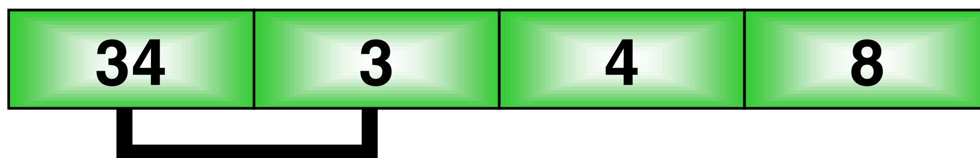
R=-1 => không tìm thấy X=-1

Bài Toán Sắp Xếp

- Cho danh sách có n phần tử a_0, a_1, a_2…, a_{n-1}.
- Sắp xếp là quá trình xử lý các phần tử trong danh sách để đặt chúng theo một thứ tự thỏa mãn một số tiêu chuẩn nào đó dựa trên thông tin lưu tại mỗi phần tử, như:
  - Sắp xếp danh sách lớp học tăng theo điểm trung bình.
  - Sắp xếp danh sách sinh viên tăng theo tên.
  - …
- Để đơn giản trong việc trình bày giải thuật ta dùng mảng 1 chiều **a** để lưu danh sách trên trong bộ nhớ chính.

Bài Toán Sắp Xếp (tt)

- a: là dãy các phần tử dữ liệu
- Để sắp xếp dãy a theo thứ tự (giả sử theo thứ tự tăng), ta tiến hành triệt tiêu tất cả các nghịch thế trong a.
  - Nghịch thế:
    - Cho dãy có n phần tử a_0, a_1,…,a_{n-1}
    - Nếu i<j và $a_i > a_j$



a[0], a[1] là cặp nghịch thế

- Đánh giá độ phức tạp của giải thuật, ta tính

  C_{ss}: Số lượng phép so sánh cần thực hiện

  C_{HV}: Số lượng phép hoán vị cần thực hiện

Các Thuật Toán Sắp Xếp

1. Đổi chỗ trực tiếp – Interchange Sort
2. Chọn trực tiếp – Selection Sort
3. Nổi bọt – Bubble Sort
4. Shaker Sort
5. Chèn trực tiếp – Insertion Sort
6. Chèn nhị phân – Binary Insertion Sort
7. Shell Sort
8. Heap Sort
9. Quick Sort
10. Merge Sort
11. Radix Sort

Các Thuật Toán Sắp Xếp

1. **Đổi chỗ trực tiếp – Interchange Sort**
2. Chọn trực tiếp – Selection Sort
3. Nổi bọt – Bubble Sort
4. Shaker Sort
5. Chèn trực tiếp – Insertion Sort
6. Chèn nhị phân – Binary Insertion Sort
7. Shell Sort
8. Heap Sort
9. Quick Sort
10. Merge Sort
11. Radix Sort

Đổi Chỗ Trực Tiếp – Interchange Sort

- **Ý tưởng**: Xuất phát từ đầu dãy, tìm tất các các nghịch thế chứa phần tử này, triệt tiêu chúng bằng cách đổi chỗ 2 phần tử trong cặp nghịch thế. Lặp lại xử lý trên với phần tử kế trong dãy.

Các Bước Tiến Hành

- Bước 1: i = 0; *// bắt đầu từ đầu dãy*
- Bước 2: j = i+1; *//tìm các nghịch thế với a[i]*
- Bước 3:

  Trong khi j < N thực hiện

  Nếu a[j]<a[i] *//xét cặp a[i], a[j]*

  Swap(a[i],a[j]);

  j = j+1;
- Bước 4: i = i+1;

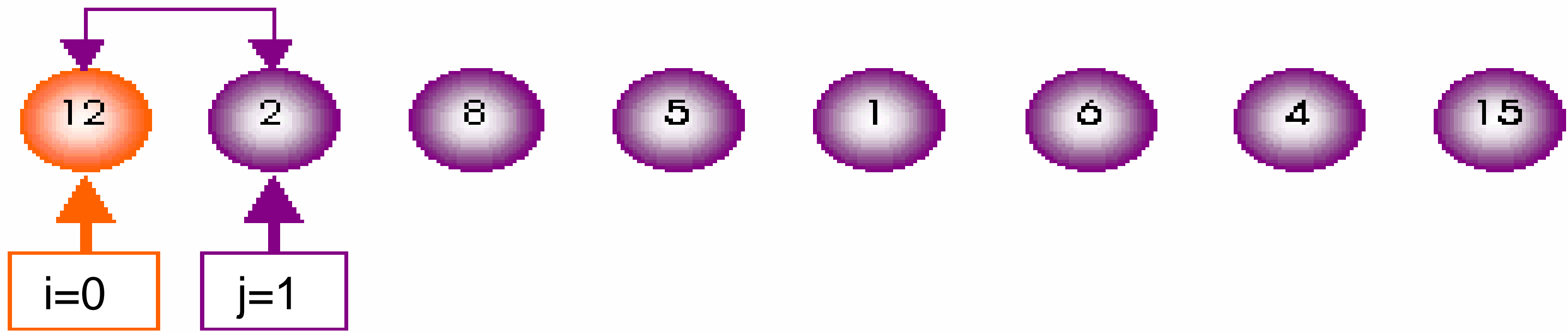
  Nếu i < N-1: Lặp lại Bước 2.

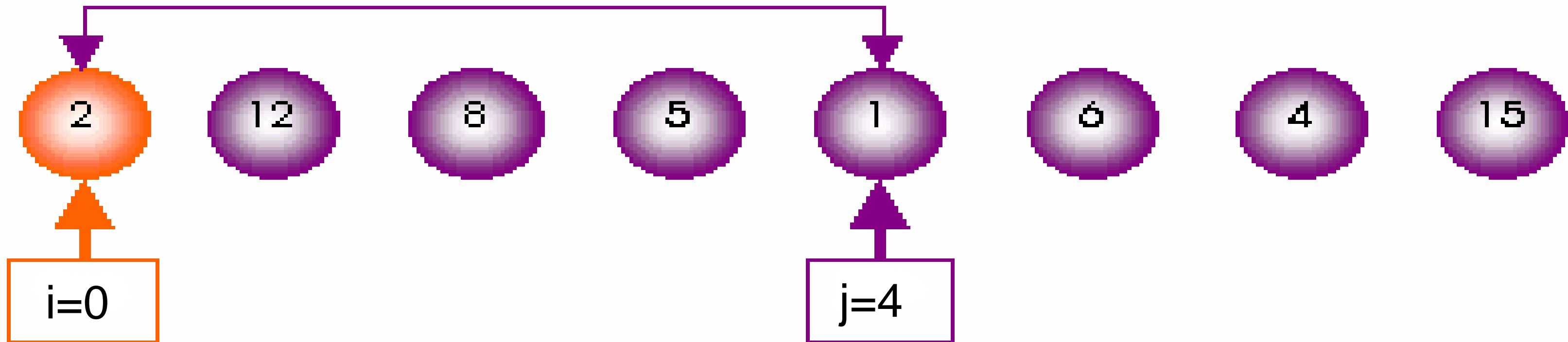
  Ngược lại: Dừng.

Đổi Chỗ Trực Tiếp – Interchange Sort

- Cho dãy số a:

12 2 8 5 1 6 4 15





Đổi Chỗ Trực Tiếp – Interchange Sort

1 12 8 5 2 6 4 15

i=1 j=2

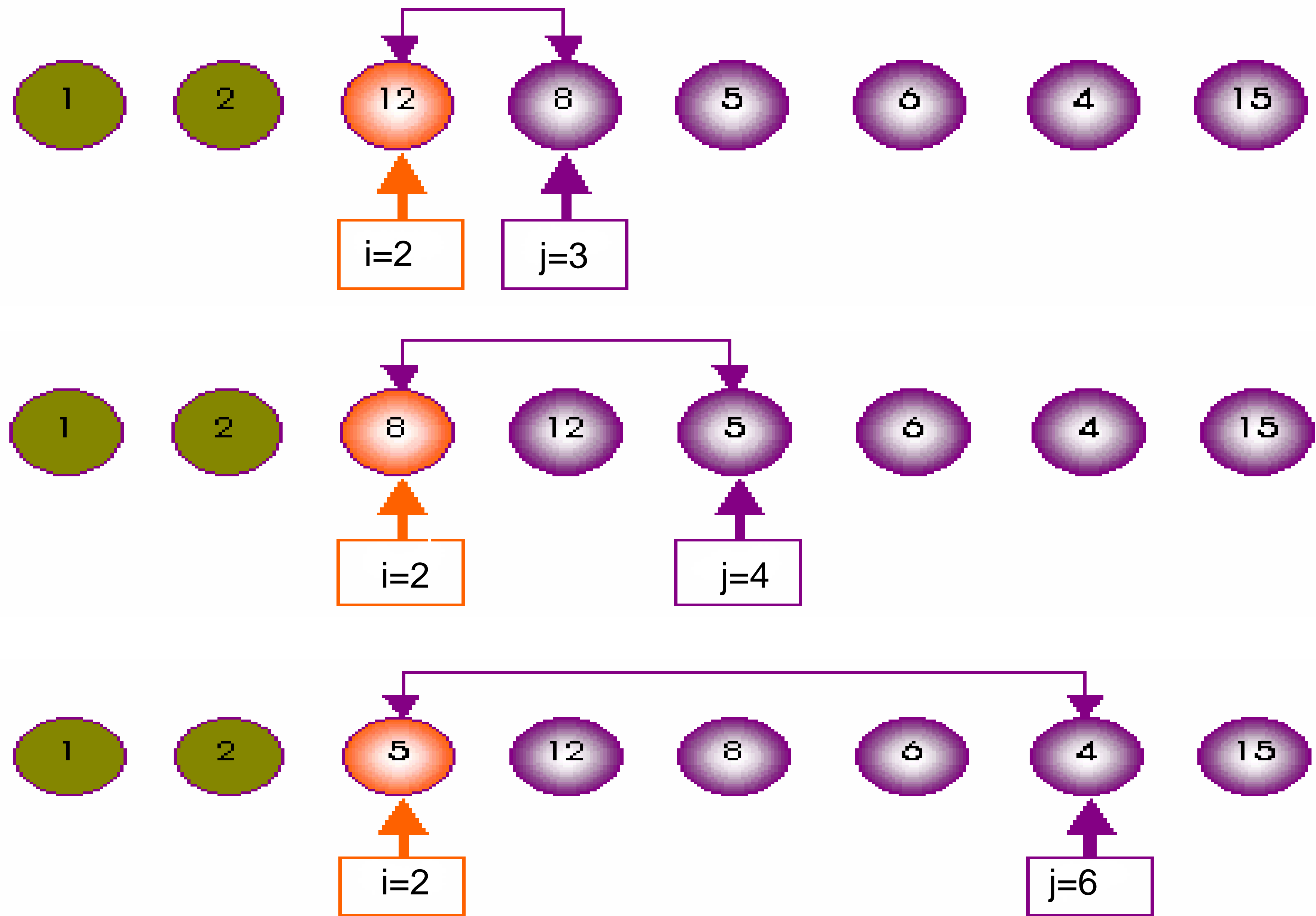
1 8 12 5 2 6 4 15

i=1 j=3

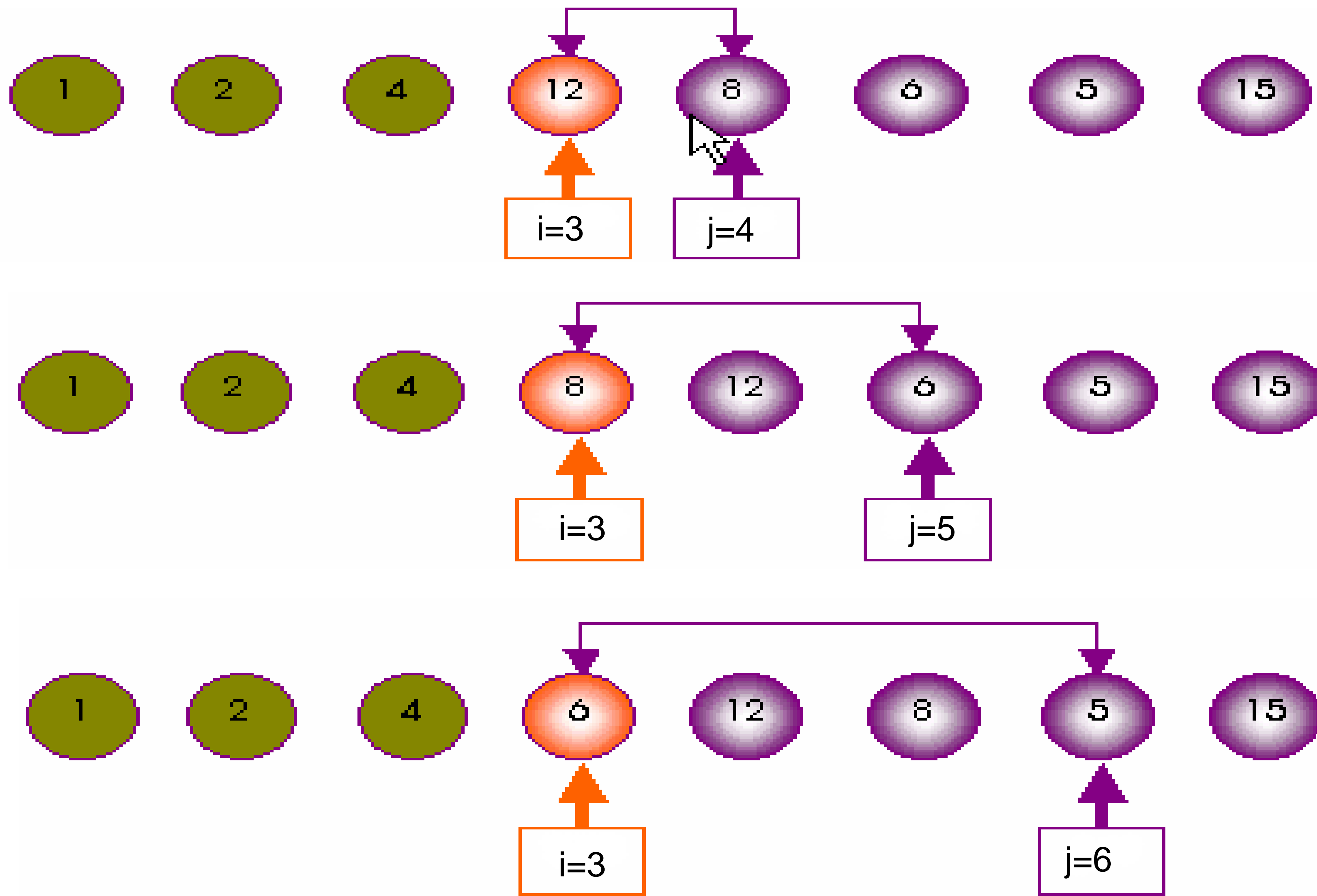
1 5 12 8 2 6 4 15

i=1 j=4

Đổi Chỗ Trực Tiếp – Interchange Sort



Đổi Chỗ Trực Tiếp – Interchange Sort



Đổi Chỗ Trực Tiếp – Interchange Sort

1 2 4 5 12 8 6 15

i=4 j=5

1 2 4 5 8 12 6 15

i=4 j=6

1 2 4 5 6 12 8 15

i=5 j=6

Đổi Chỗ Trực Tiếp – Interchange Sort

1 2 4 5 6 8 12 15

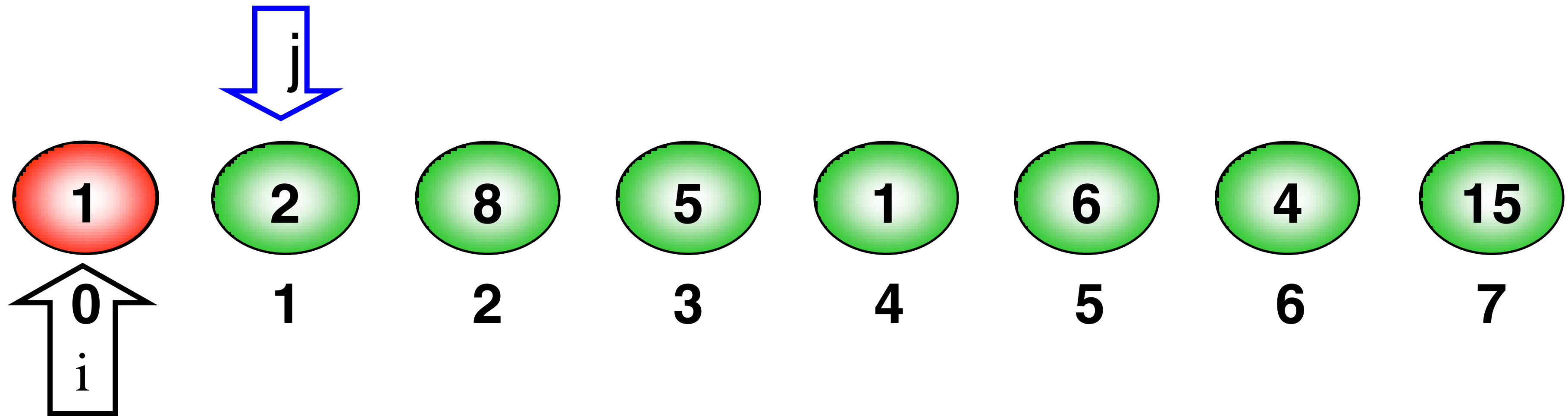
i=6 j=7

1 2 4 5 6 8 12 15

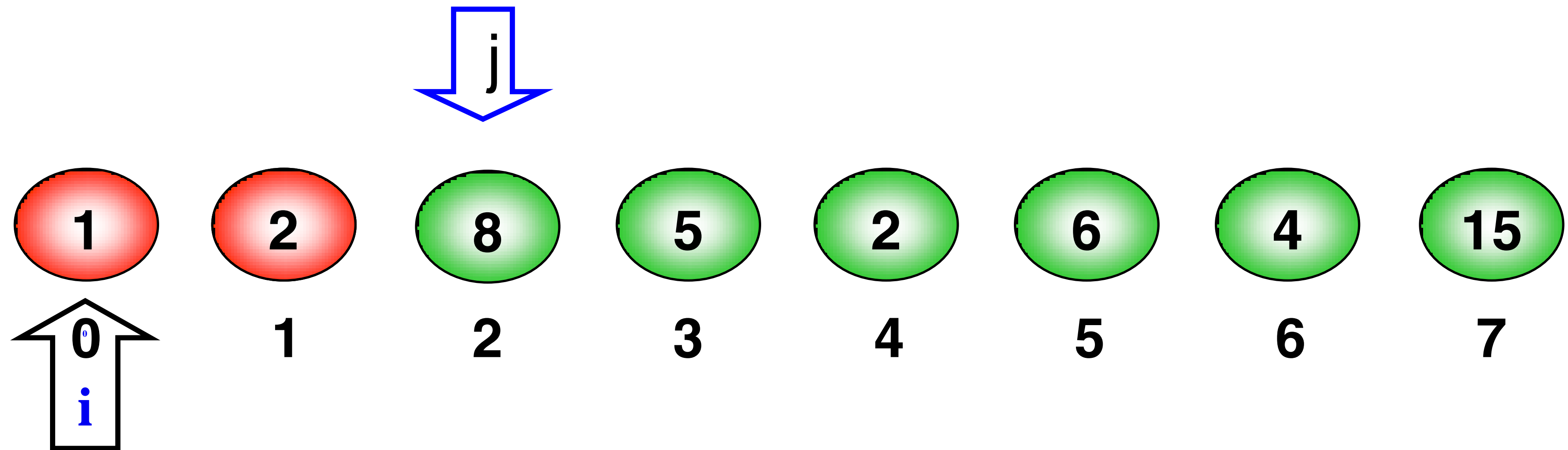
Cài Đặt Đổi Chỗ Trực Tiếp

```
void InterchangeSort(int a[], int N )
{
    int     i, j;
    for (i = 0 ; i<N-1 ; i++)
        for (j =i+1; j < N ; j++)
            if(a[j ]< a[i])  // Thỏa 1 cặp nghịch thế
                Swap(a[i], a[j]);
}
```

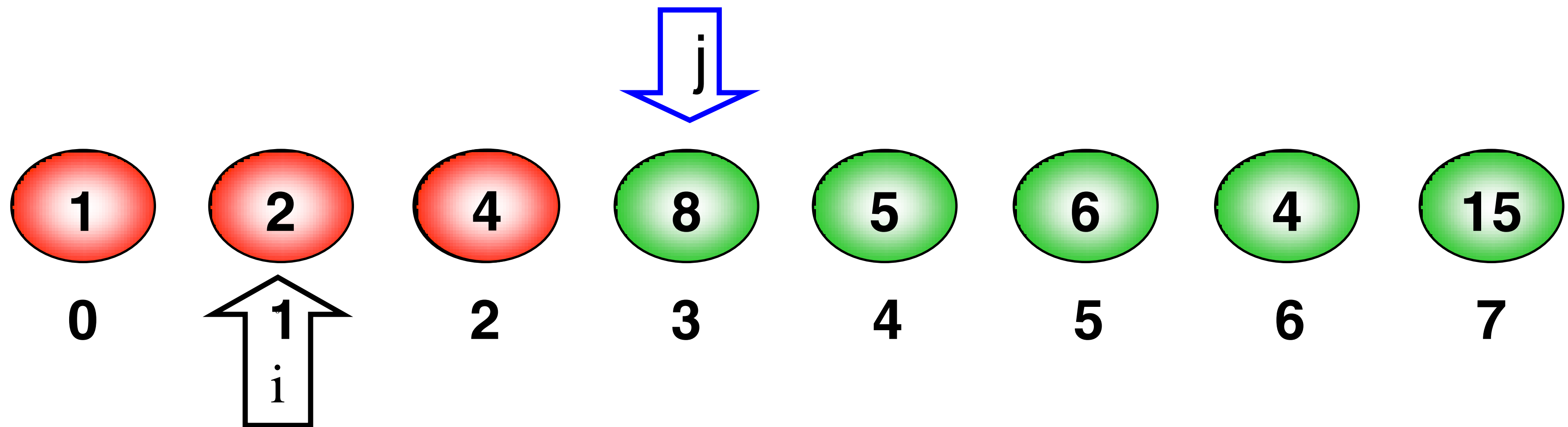
Minh Họa Thuật Toán



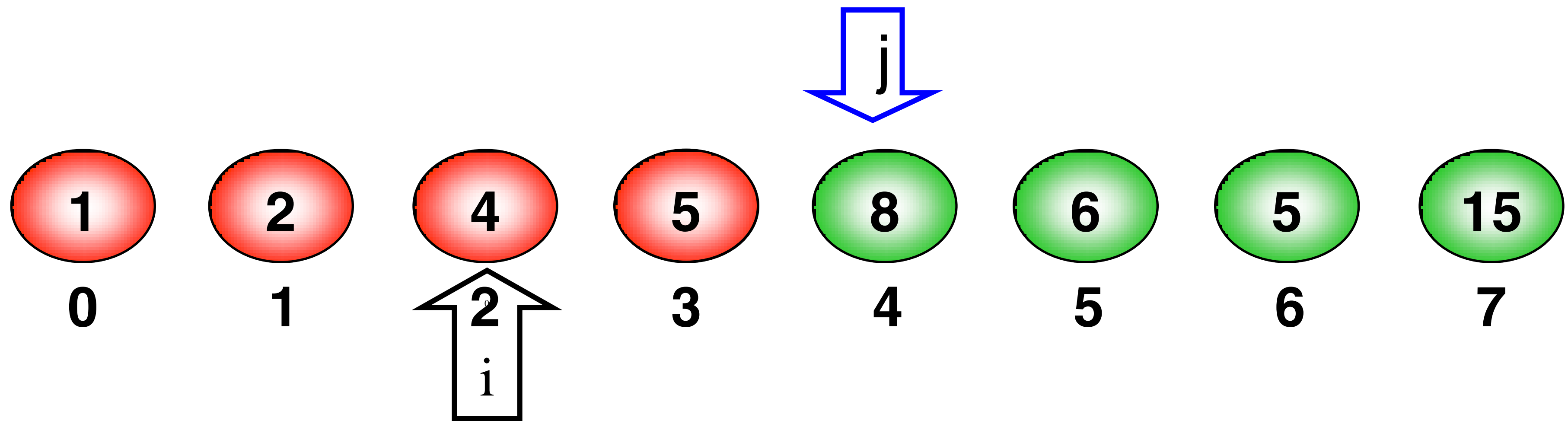
Minh Họa Thuật Toán



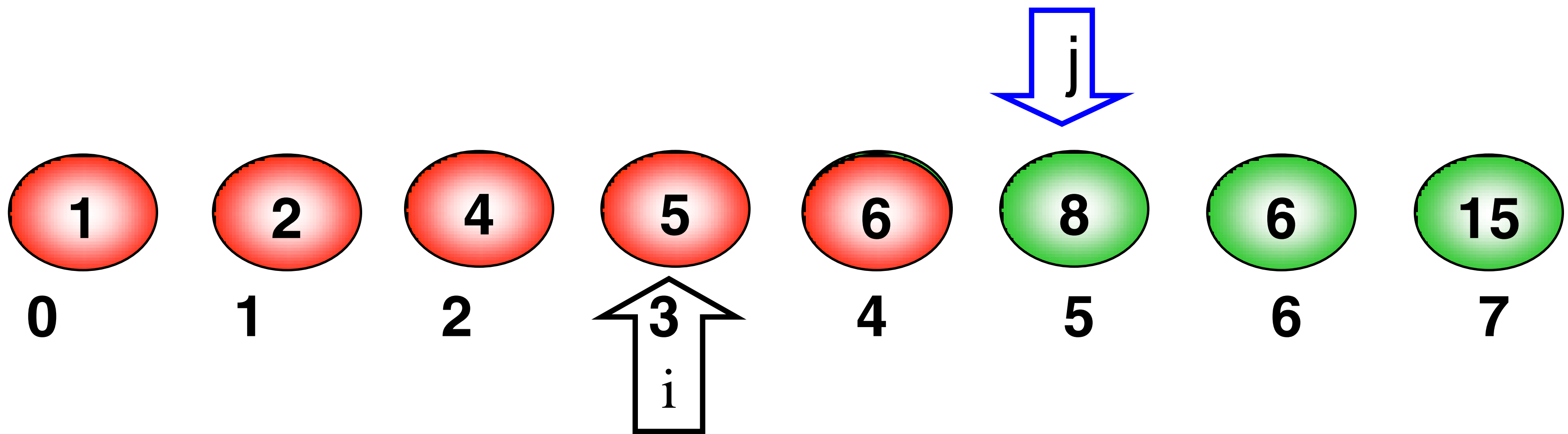
Minh Họa Thuật Toán



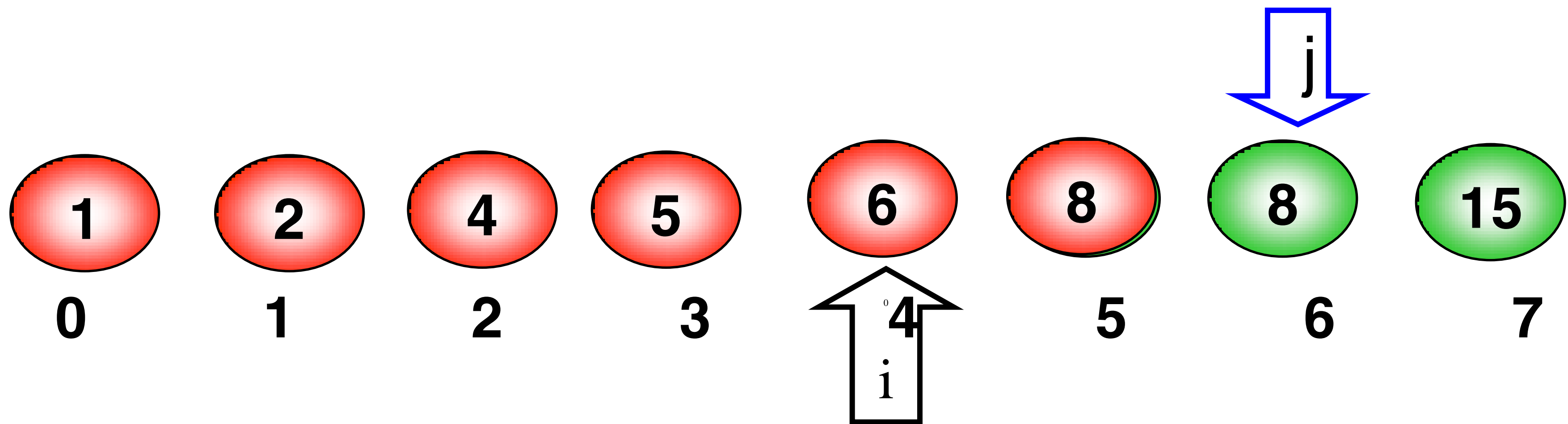
Minh Họa Thuật Toán



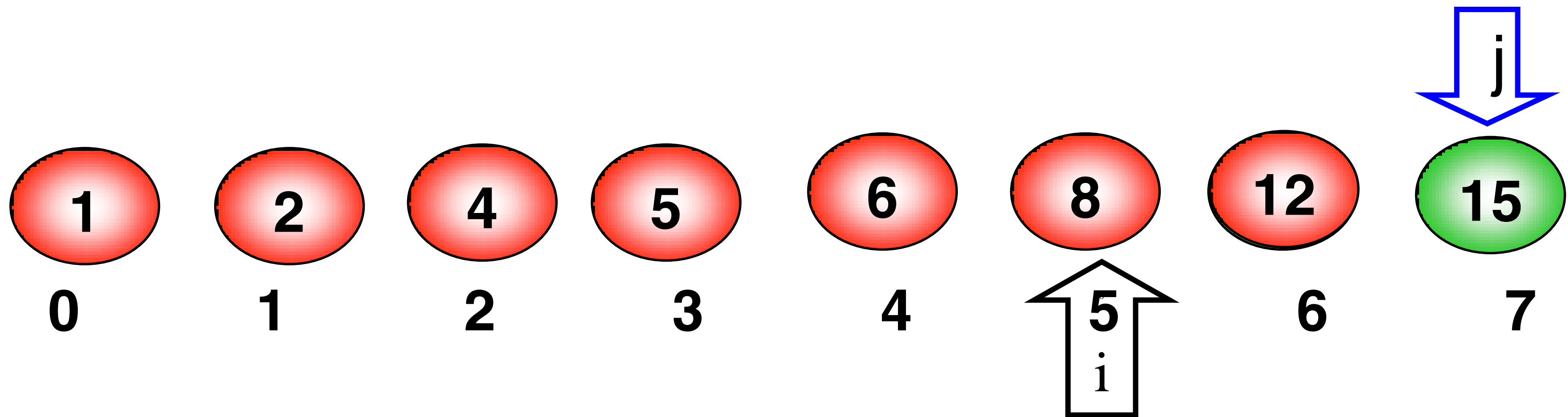
Minh Họa Thuật Toán



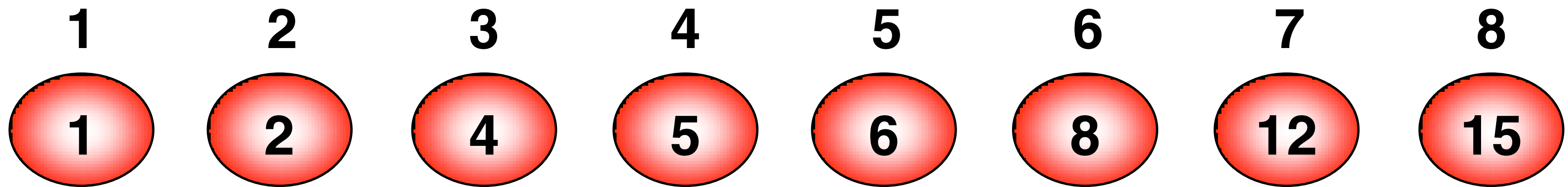
Minh Họa Thuật Toán



Minh Họa Thuật Toán



Minh Họa Thuật Toán



Độ Phức Tạp Của Thuật Toán

Trường hợp	Số lần so sánh	Số lần hoán vị
Tốt nhất	$\sum_{i=1}^{n-1}(n-i+1)=\frac{n(n-1)}{2}$	0
Xấu nhất	$\frac{n(n-1)}{2}$	$\sum_{i=1}^{n-1}(n-i+1)=\frac{n(n-1)}{2}$

Các Thuật Toán Sắp Xếp

1. Đổi chỗ trực tiếp – Interchange Sort
2. **Chọn trực tiếp – Selection Sort**
3. Nổi bọt – Bubble Sort
4. Shaker Sort
5. Chèn trực tiếp – Insertion Sort
6. Chèn nhị phân – Binary Insertion Sort
7. Shell Sort
8. Heap Sort
9. Quick Sort
10. Merge Sort
11. Radix Sort

Chọn Trực Tiếp – Selection Sort

- **Ý tưởng**:
  - Chọn phần tử nhỏ nhất trong N phần tử trong dãy hiện hành ban đầu.
  - Đưa phần tử này về vị trí đầu dãy hiện hành
  - Xem dãy hiện hành chỉ còn N-1 phần tử của dãy hiện hành ban đầu
    - Bắt đầu từ vị trí thứ 2;
    - Lặp lại quá trình trên cho dãy hiện hành... đến khi dãy hiện hành chỉ còn 1 phần tử

Các Bước Của Thuật Toán Chọn Trực Tiếp

- Bước 1: i = 0;
- Bước 2: Tìm phần tử **a[min]** nhỏ nhất trong dãy hiện hành từ a[i] đến a[N]
- Bước 3 : Đổi chỗ a[min] và a[i]
- Bước 4 : Nếu i < N-1 thì

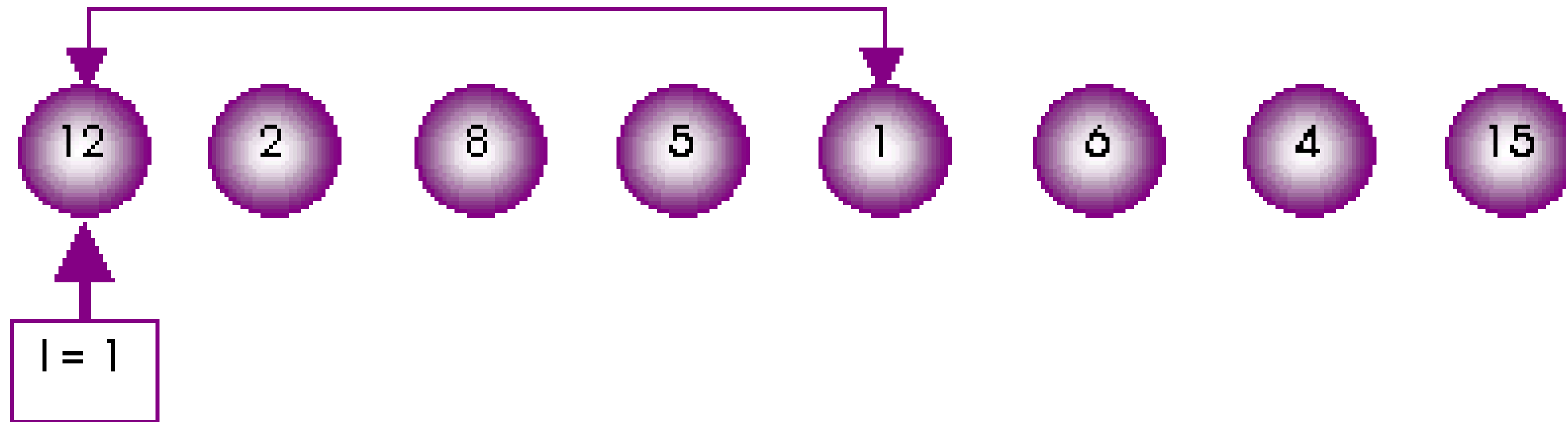
  i = i+1; Lặp lại Bước 2;

  Ngược lại: Dừng.

Chọn Trực Tiếp – Selection Sort

- Cho dãy số a:

12 2 8 5 1 6 4 15



Chọn Trực Tiếp – Selection Sort

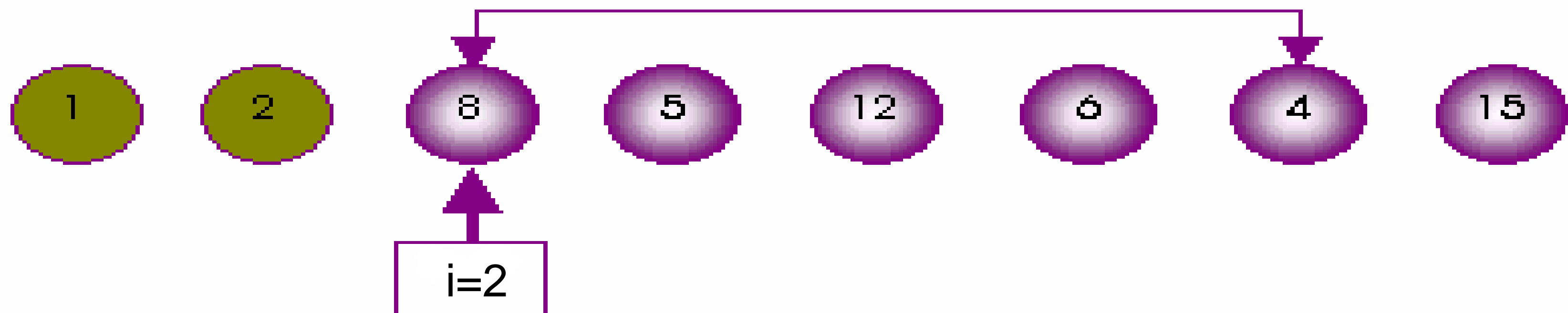
12 2 8 5 1 6 4 15

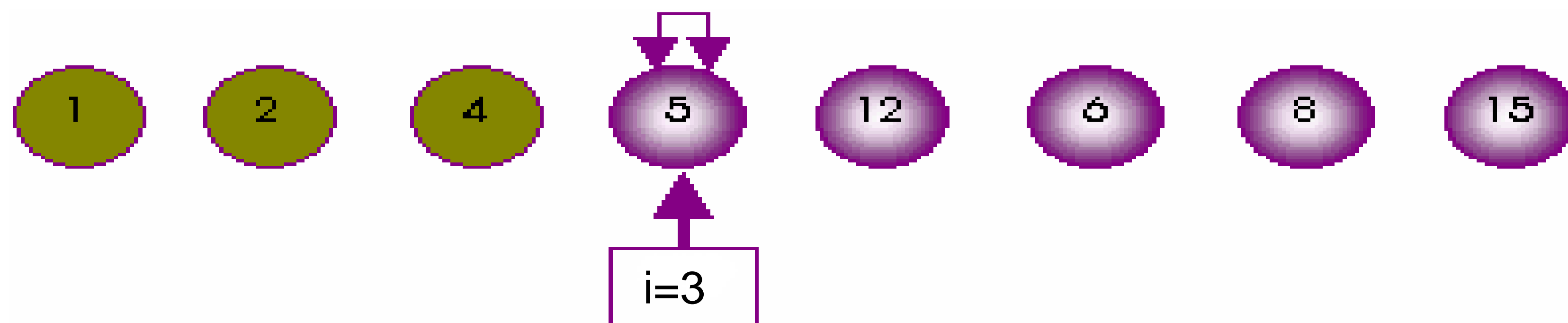
i=0

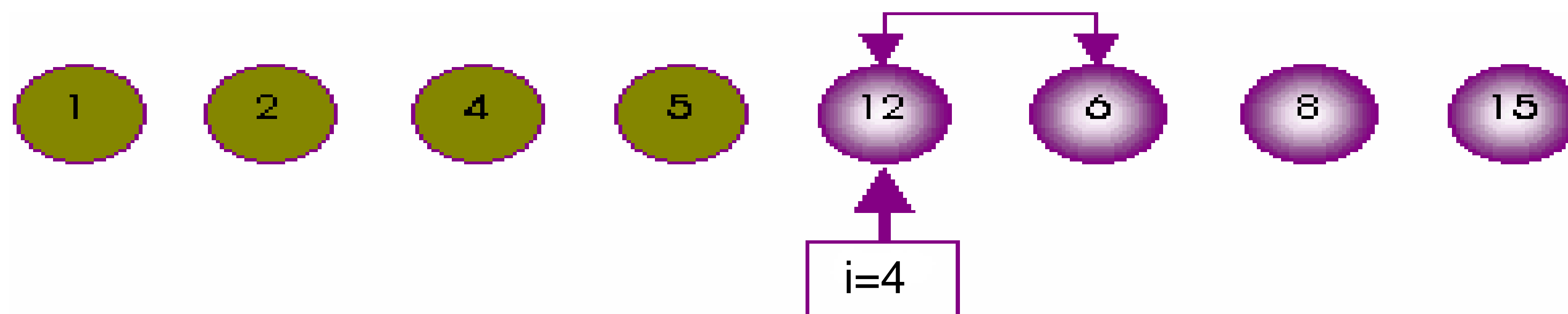
1 2 8 5 12 6 4 15

i=1

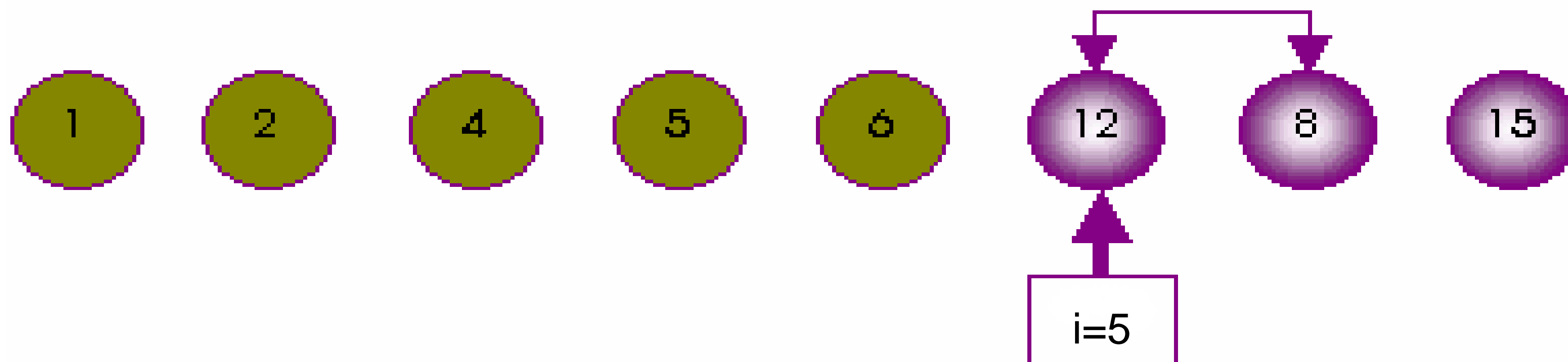
Chọn Trực Tiếp – Selection Sort

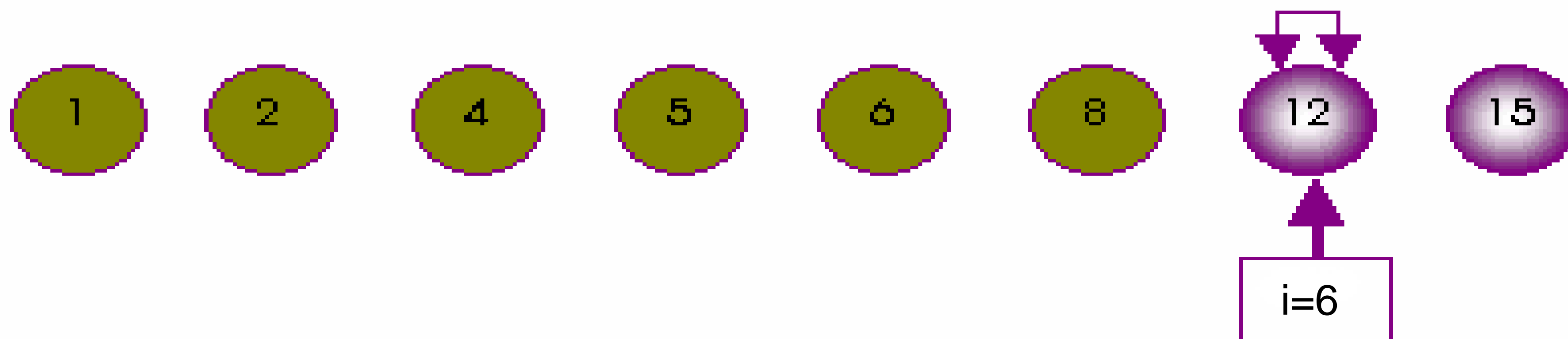






Chọn Trực Tiếp – Selection Sort

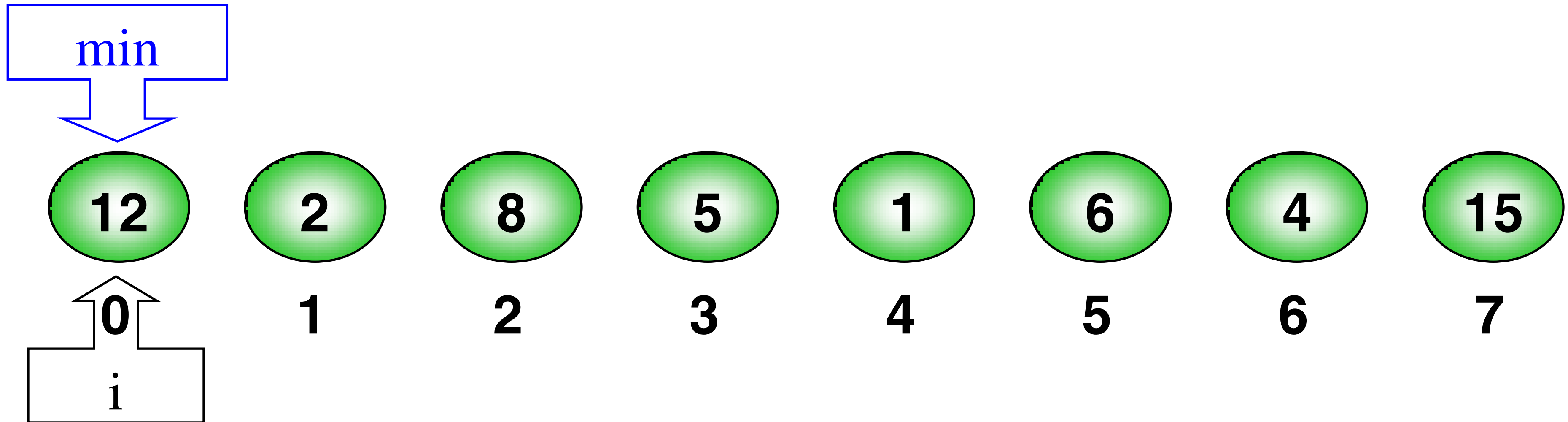




Cài Đặt Thuật Toán Chọn Trực Tiếp

```
void SelectionSort(int a[],int n )
{
        int     min,i,j; // chỉ số phần tử nhỏ nhất trong dãy hiện hành
        for (i=0; i<n-1 ; i++) //chỉ số đầu tiên của dãy hiện  hành
        {
                min = i;
                for(j = i+1; j <N ; j++)
                        if (a[j ] < a[min])
                                min = j; // lưu vtrí phần tử hiện nhỏ nhất
                        Swap(a[min],a[i]);
        }
    }
```

Minh Họa Thuật Toán Chọn Trực Tiếp



Minh Họa Thuật Toán Chọn Trực Tiếp

Vị trí nhỏ nhất(1,7)

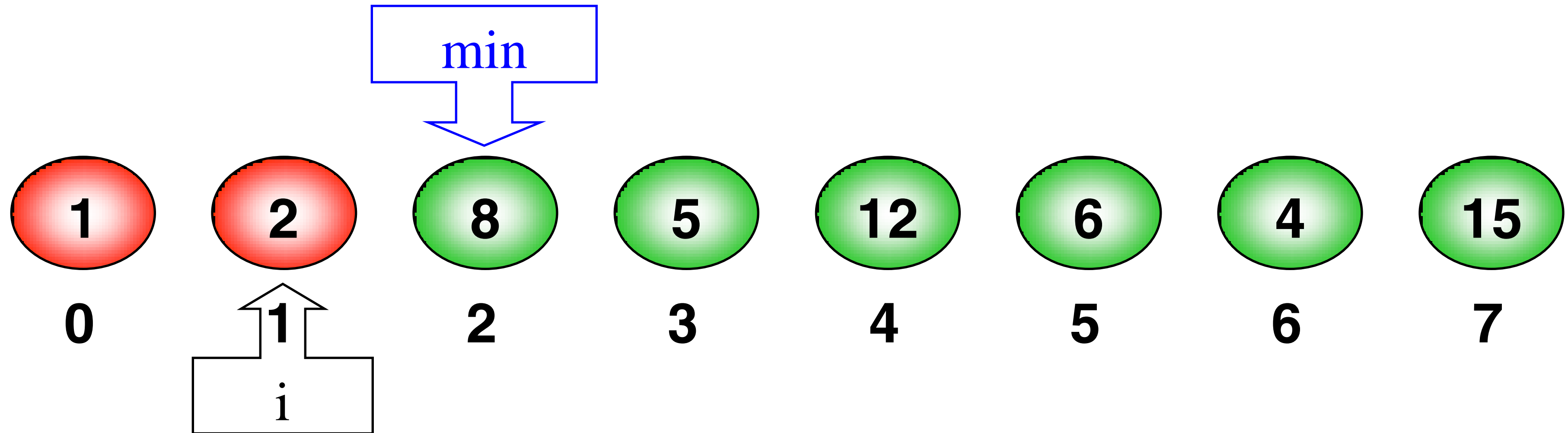
Swap(a[1], a[1])

min

1	2	8	5	12	6	4	15
0	1	2	3	4	5	6	7

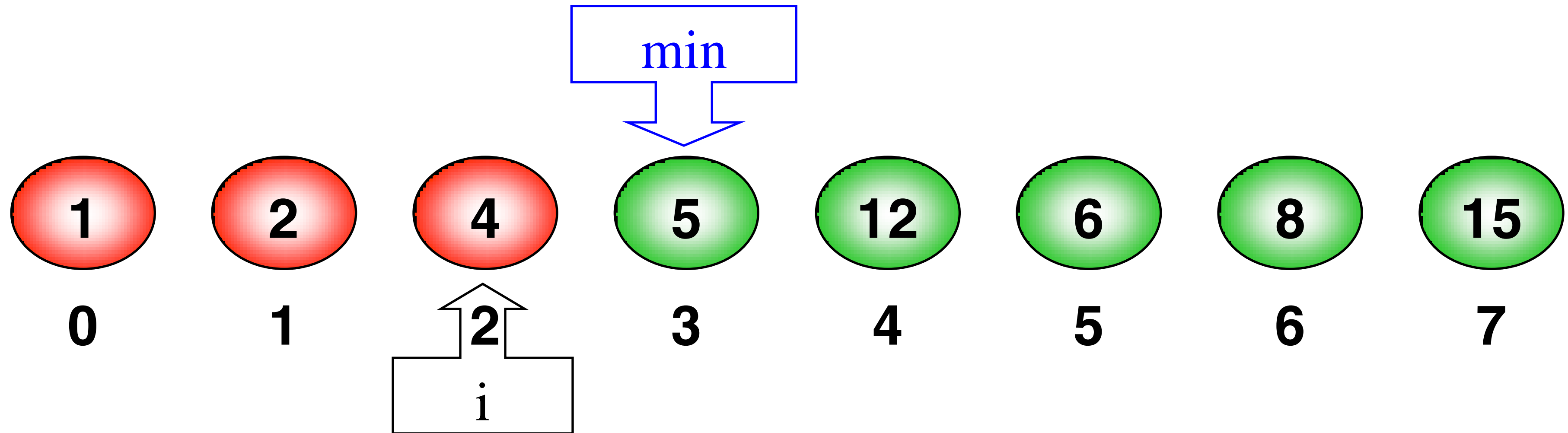
i

Minh Họa Thuật Toán Chọn Trực Tiếp

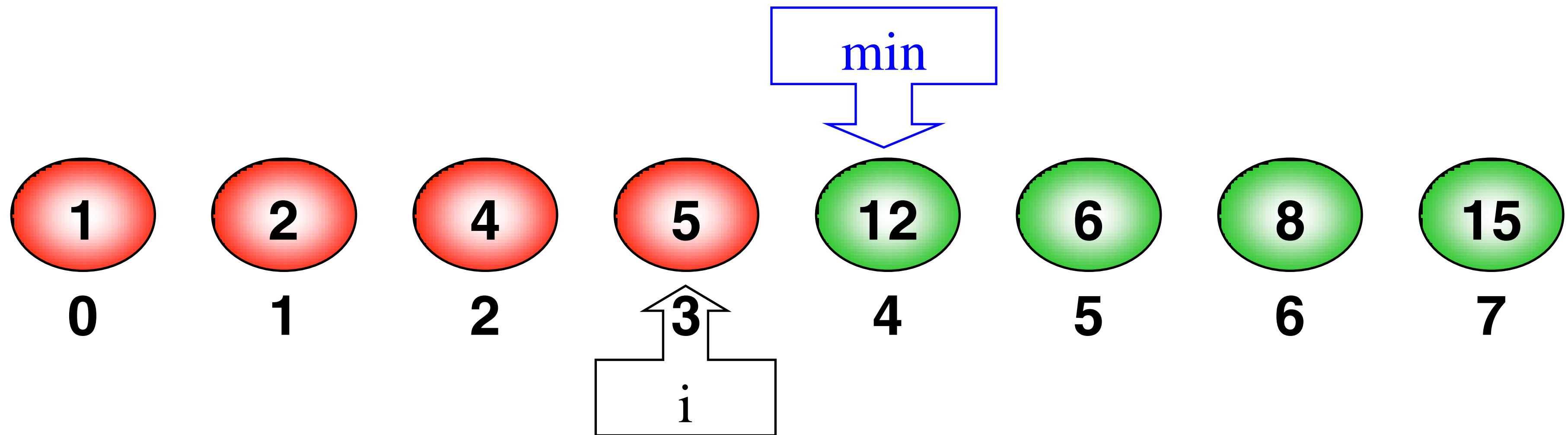


Minh Họa Thuật Toán Chọn Trực Tiếp

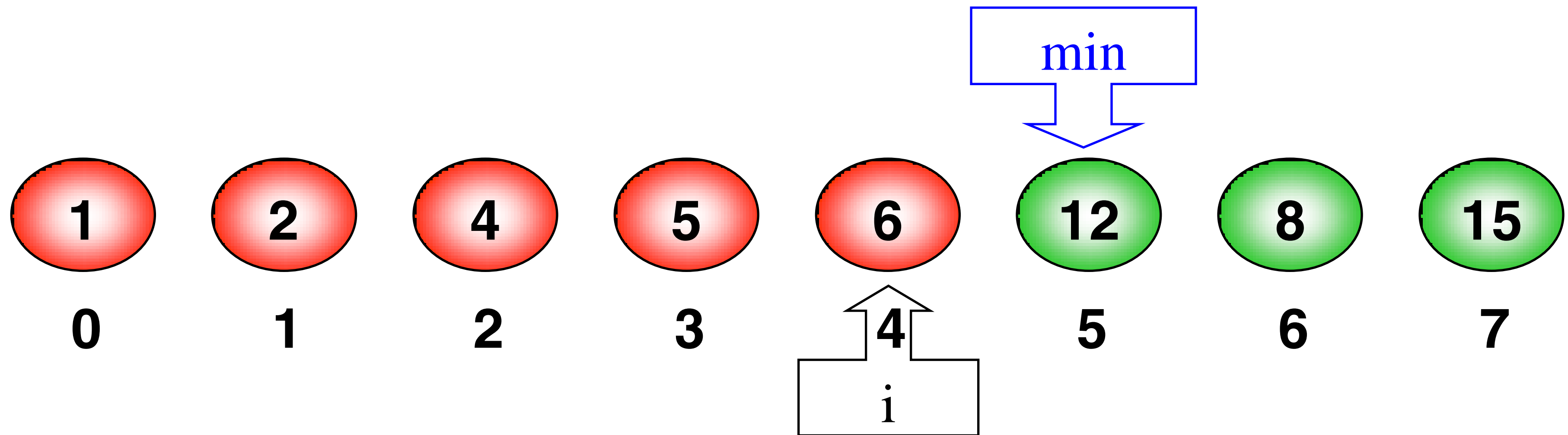
Vị trí nhỏ nhất(3, 7) **Swap(a[3], a[3])**



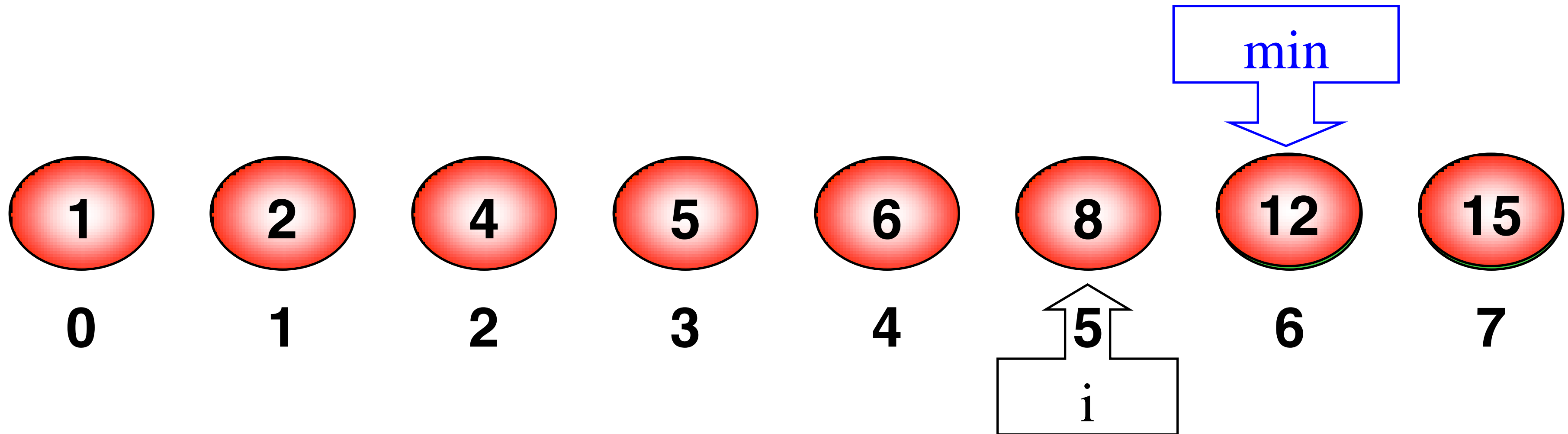
Minh Họa Thuật Toán Chọn Trực Tiếp



Minh Họa Thuật Toán Chọn Trực Tiếp



Minh Họa Thuật Toán Chọn Trực Tiếp



- **Đánh giá giải thuật**

số lần so sánh

Trường hợp	Số lần so sánh	Số phép gán
Tốt nhất	n(n-1)/2	0
Xấu nhất	n(n-1)/2	3n(n-1)/2

Các Thuật Toán Sắp Xếp

1. Đổi chỗ trực tiếp – Interchange Sort
2. Chọn trực tiếp – Selection Sort
3. **Nổi bọt – Bubble Sort**
4. Shaker Sort
5. Chèn trực tiếp – Insertion Sort
6. Chèn nhị phân – Binary Insertion Sort
7. Shell Sort
8. Heap Sort
9. Quick Sort
10. Merge Sort
11. Radix Sort

Nổi Bọt – Bubble Sort

➢ **Ý tưởng**:

- Xuất phát từ cuối dãy, đổi chỗ các cặp phần tử kế cận để đưa phần tử nhỏ hơn trong cặp phần tử đó về vị trí đúng đầu dãy hiện hành, sau đó sẽ không xét đến nó ở bước tiếp theo, do vậy ở lần xử lý thứ i sẽ có vị trí đầu dãy là i.
- Lặp lại xử lý trên cho đến khi không còn cặp phần tử nào để xét.

Nổi Bọt – Bubble Sort

- Bước 1 : i = 0; // lần xử lý đầu tiên
- Bước 2 : j = N-1;//Duyệt từ cuối dãy ngược về vị trí i

  Trong khi (j > i) thực hiện:

  Nếu a[j]<a[j-1]

  Doicho(a[j],a[j-1]);

  j = j-1;
- Bước 3 : i = i+1; // lần xử lý kế tiếp

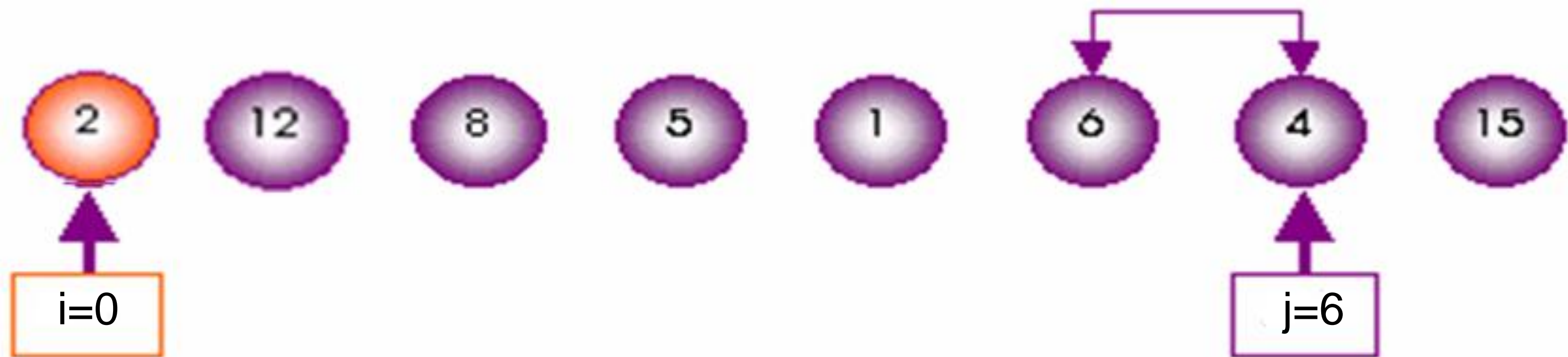
  Nếu i =N: Hết dãy. Dừng

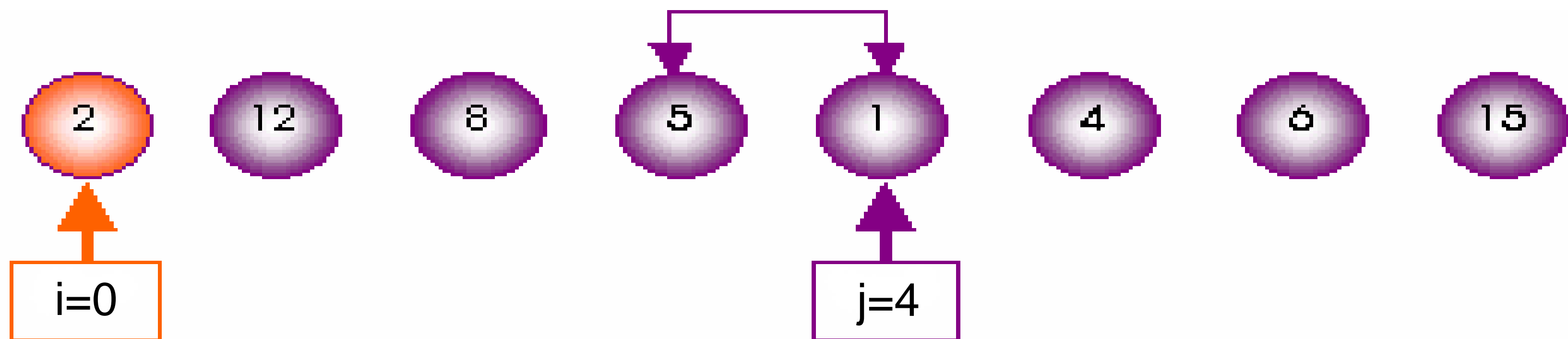
  Ngược lại : Lặp lại Bước 2.

Nổi Bọt – Bubble Sort

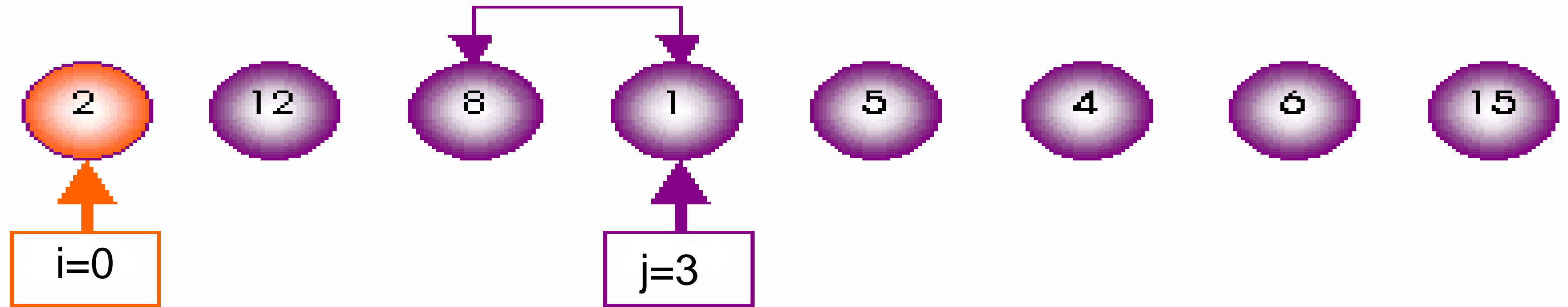
- Cho dãy số a:

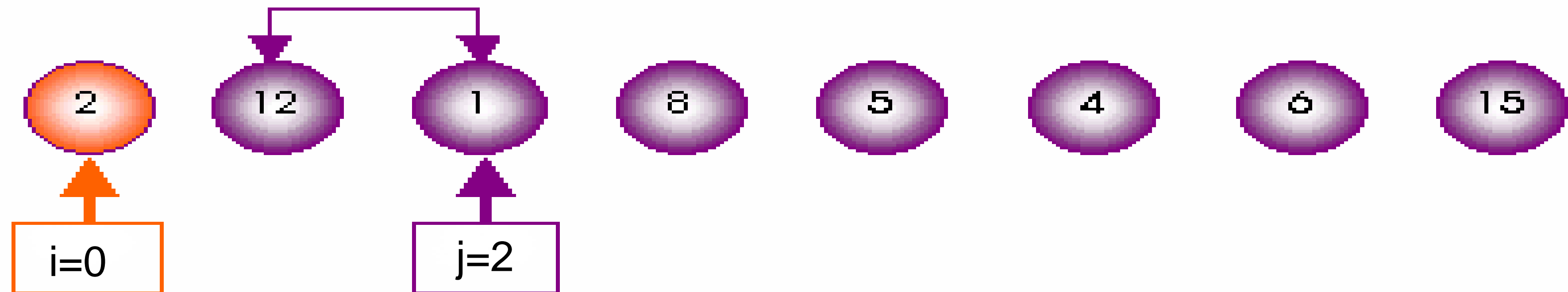
2 12 8 5 1 6 4 15

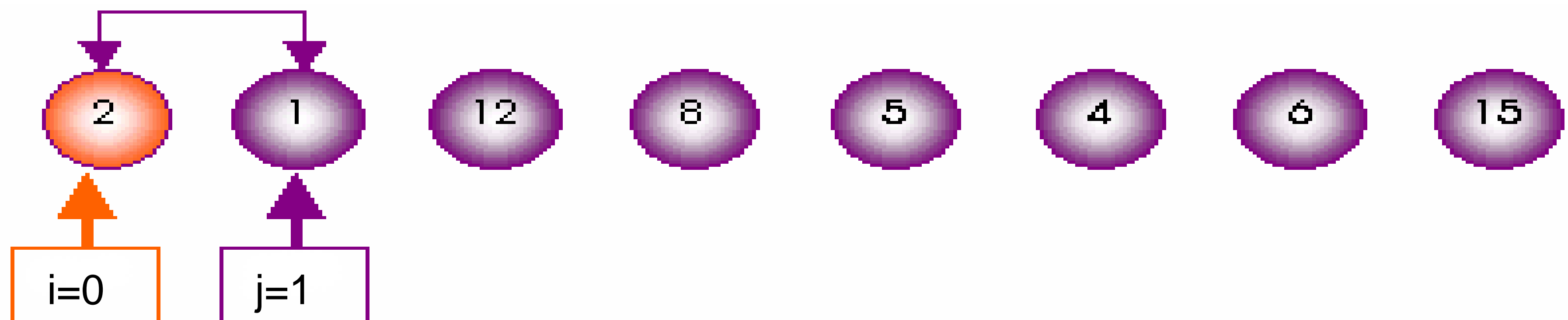




Nổi Bọt – Bubble Sort







Nổi Bọt – Bubble Sort

1 2 12 8 5 4 6 15

i=1 j=5

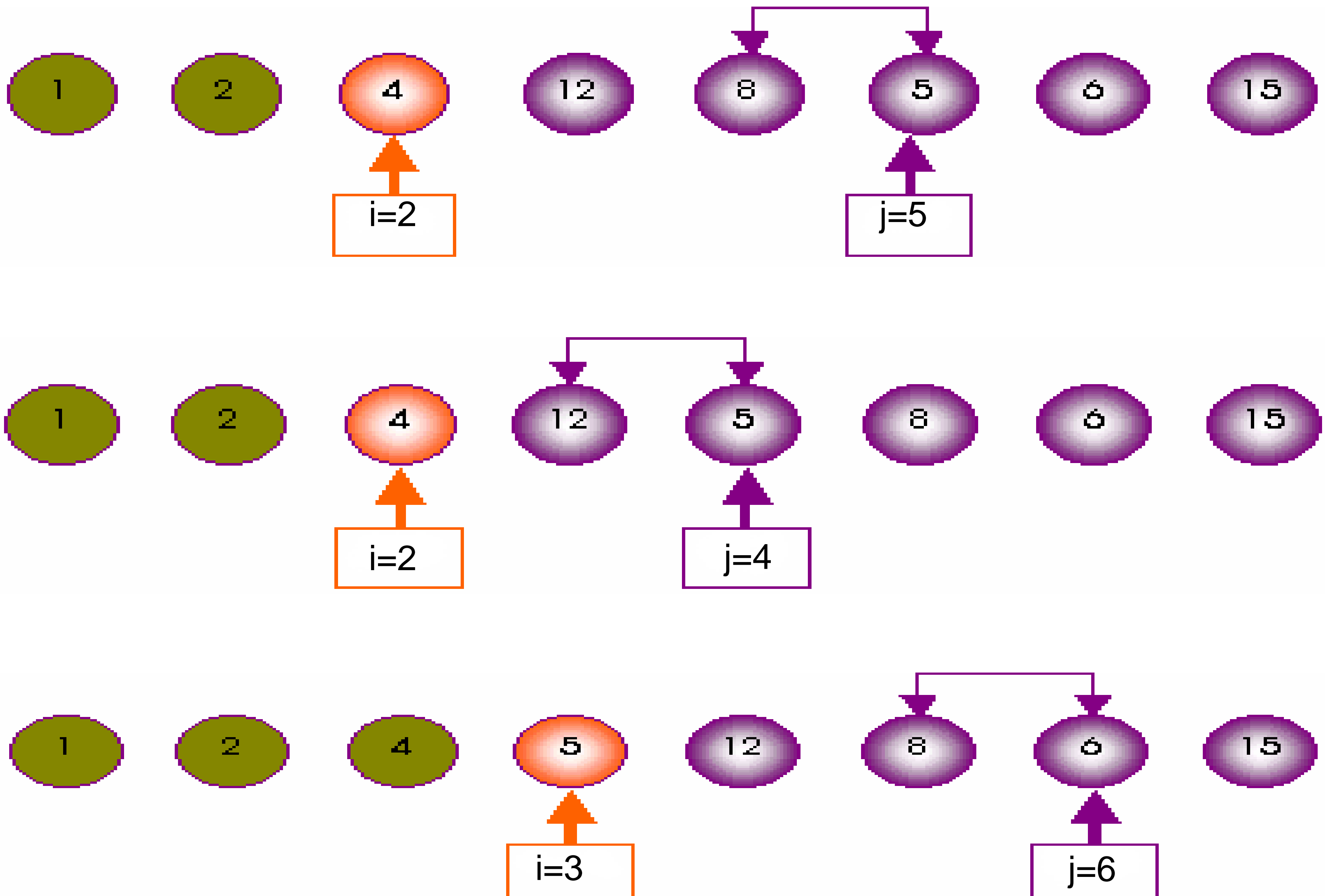
1 2 12 8 4 5 6 15

i=1 j=4

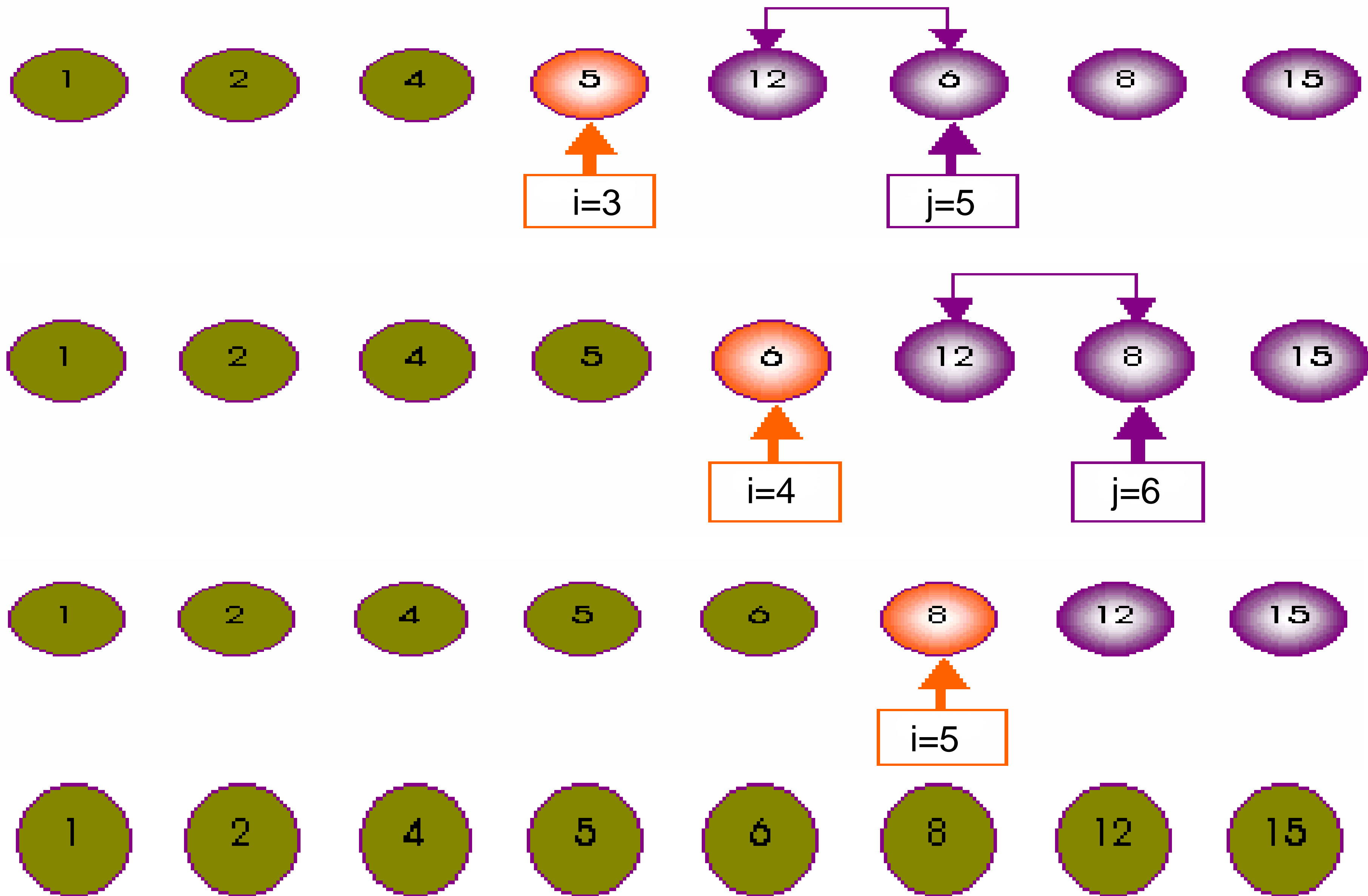
1 2 12 4 8 5 6 15

i=1 j=3

Nổi Bọt – Bubble Sort



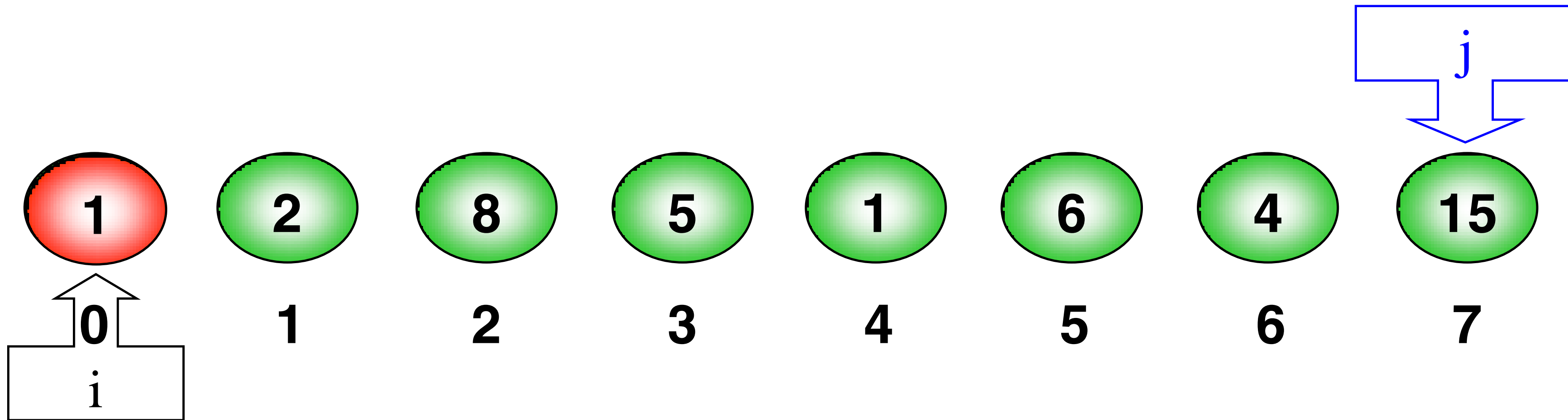
Nổi Bọt – Bubble Sort



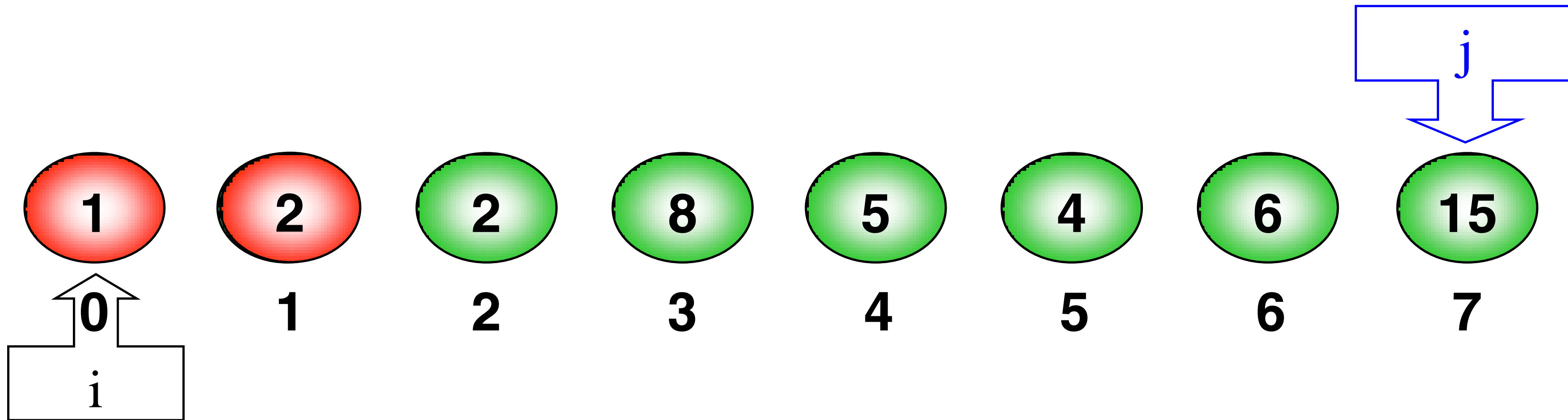
Cài Đặt Thuật Toán Nổi Bọt

```
void BubbleSort(int a[],int n)
{
    int     i, j;
    for (i = 0 ; i<n-1 ; i++)
        for (j =n-1; j >i ; j --)
            if(a[j]< a[j-1])// nếu sai vị trí thì đổi chỗ
                Swap(a[j], a[j-1]);
}
```

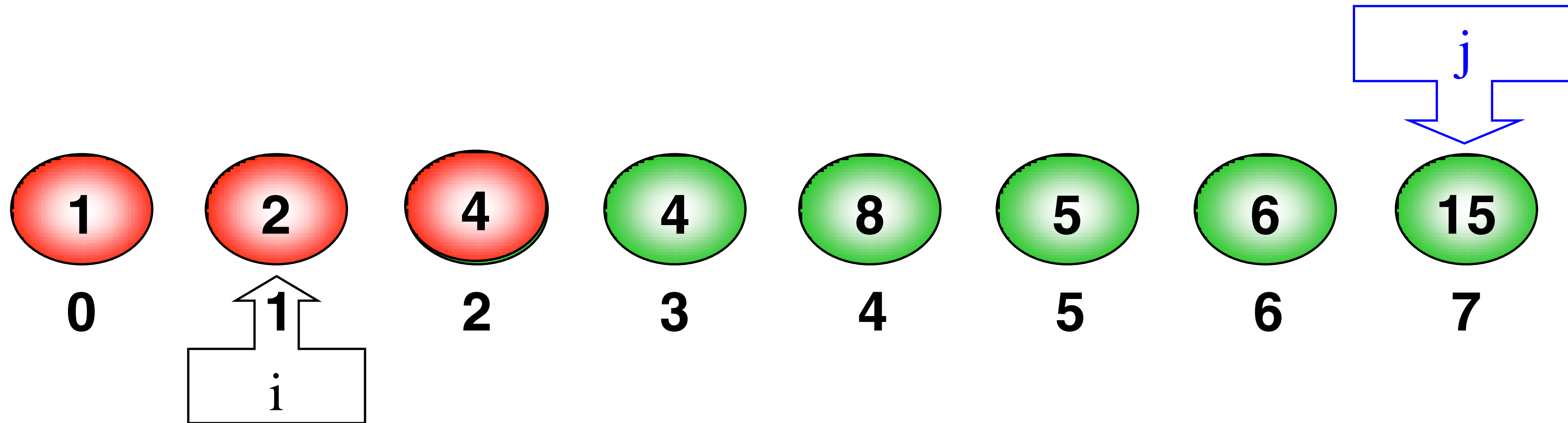
Minh Họa Thuật Toán



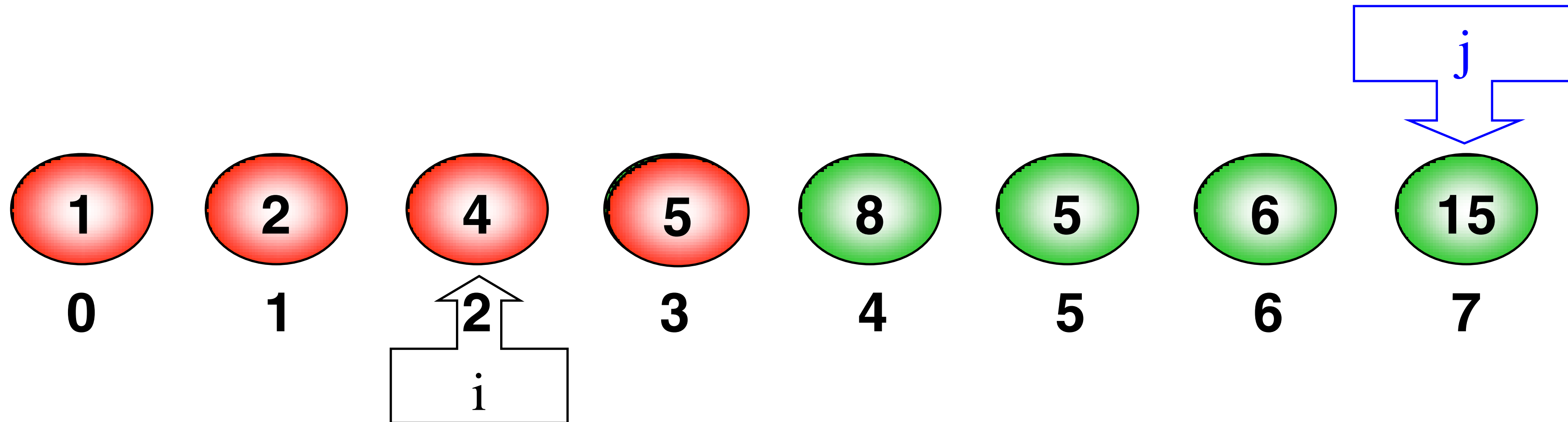
Minh Họa Thuật Toán



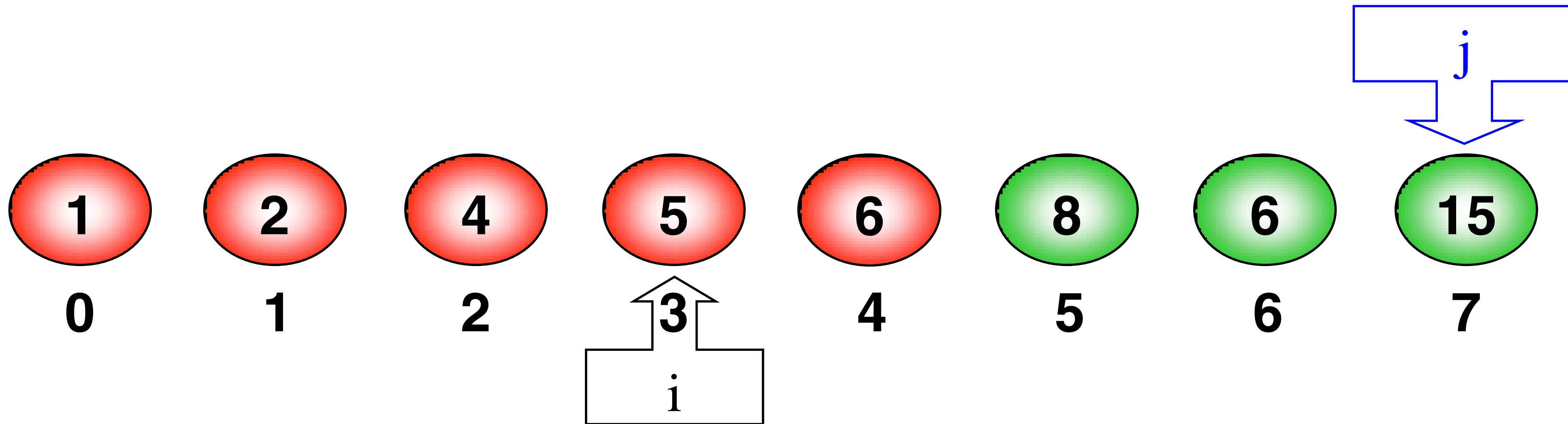
Minh Họa Thuật Toán



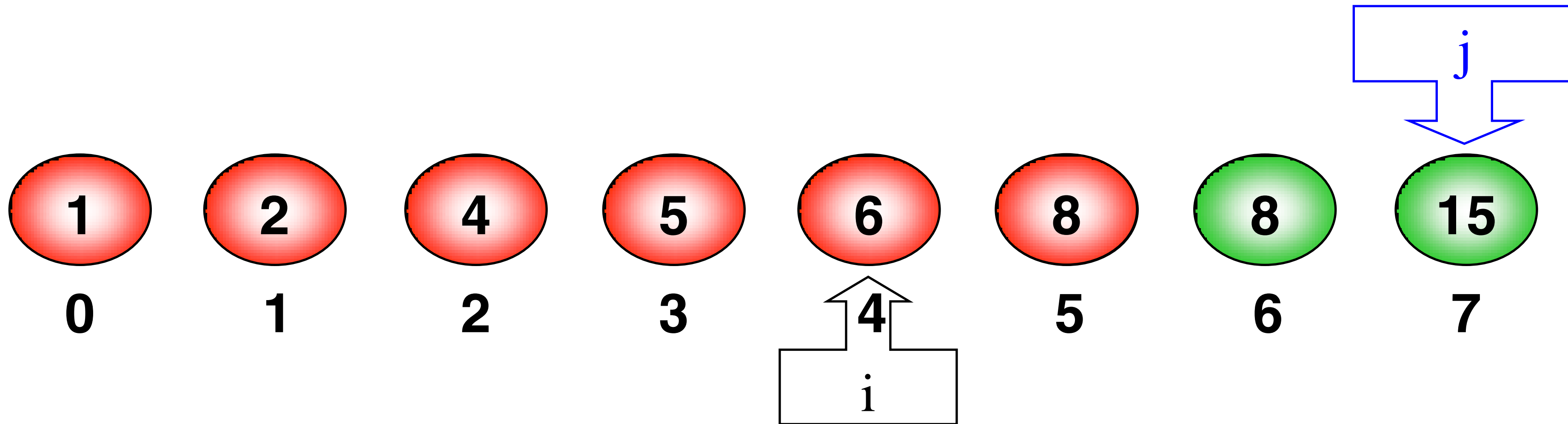
Minh Họa Thuật Toán



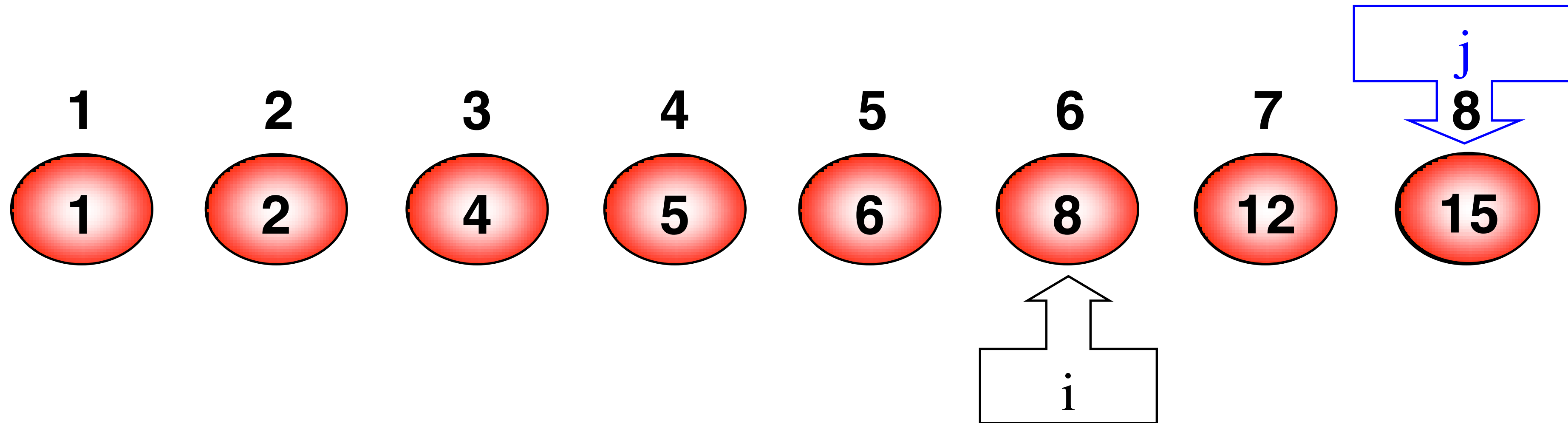
Minh Họa Thuật Toán



Minh Họa Thuật Toán



Minh Họa Thuật Toán



Độ Phức Tạp Của Thuật Toán Nổi Bọt

Trường hợp	Số lần so sánh	Số lần hoán vị
Tốt nhất	$\sum_{i=1}^{n-1}(n-i+1)=\frac{n(n-1)}{2}$	0
Xấu nhất	$\frac{n(n-1)}{2}$	$\sum_{i=1}^{n-1}(n-i+1)=\frac{n(n-1)}{2}$

Các Thuật Toán Sắp Xếp

1. Đổi chỗ trực tiếp – Interchange Sort
2. Chọn trực tiếp – Selection Sort
3. Nổi bọt – Bubble Sort
4. **Shaker Sort**
5. Chèn trực tiếp – Insertion Sort
6. Chèn nhị phân – Binary Insertion Sort
7. Shell Sort
8. Heap Sort
9. Quick Sort
10. Merge Sort
11. Radix Sort

Shaker Sort

- Trong mỗi lần sắp xếp, duyệt mảng theo 2 lượt từ 2 phía khác nhau:
  - Lượt đi: đẩy phần tử nhỏ về đầu mảng.
  - Lượt về: đẩy phần tử lớn về cuối mảng.
- Ghi nhận lại những đoạn đã sắp xếp nhằm tiết kiệm các phép so sánh thừa.

Các Bước Của Thuật Toán

- Bước 1: l=0; r=n-1; //Đoạn l->r là đoạn cần được sắp xếp
  k=n; //ghi nhận vị trí k xảy ra hoán vị sau cùng
  // để làm cơ sơ thu hẹp đoạn l->r
- Bước 2:
  Bước 2a:
  j=r; //đẩy phần tử nhỏ về đầu mảng
  Trong khi j>l
  nếu a[j]<a[j-1] thì {Doicho(a[j],a[j-1]): k=j;}
  j--;
  l=k; //loại phần tử đã có thứ tự ở đầu dãy
  Bước 2b: j=l
  Trong khi j<r
  nếu a[j]>a[j+1] thì {Doicho(a[j],a[j+1]); k=j;}
  j++;
  r=k; //loại phần tử đã có thứ tự ở cuối dãy
- Bước 3: Nếu l<r lặp lại bước 2
  Ngược lại: dừng

Cài Đặt Thuật Toán Shaker Sort

```
void ShakeSort(int a[],int n)
{
    int     i, j;
    int      left, right, k;
    left = 0; right = n-1; k = n-1;
    while (left < right)
    {
            for (j = right; j > left; j --)
                    if (a[j]< a[j-1])
                            {Swap(a[j], a[j-1]);k =j;}
            left = k;
            for (j = left; j < right; j ++)
                    if (a[j]> a[j+1])
                            {Swap(a[j], a[j-1]);k = j; }
            right = k;
    }
}
```

Các Thuật Toán Sắp Xếp

1. Đổi chỗ trực tiếp – Interchange Sort
2. Chọn trực tiếp – Selection Sort
3. Nổi bọt – Bubble Sort
4. Shaker Sort
5. **Chèn trực tiếp – Insertion Sort**
6. Chèn nhị phân – Binary Insertion Sort
7. Shell Sort
8. Heap Sort
9. Quick Sort
10. Merge Sort
11. Radix Sort

Chèn Trực Tiếp – Insertion Sort

- Giả sử có một dãy $\mathbf{a_0}$ **,** $\mathbf{a_1}$ **,... ,**$\mathbf{a_{n-1}}$ trong đó i phần tử đầu tiên $\mathbf{a_0}$ **,** $\mathbf{a_1}$ **,... ,**$\mathbf{a_{i-1}}$ đã có thứ tự.
- Tìm cách chèn phần tử $\mathbf{a_i}$ vào vị trí thích hợp của đoạn đã được sắp để có dãy mới $\mathbf{a_0}$ **,** $\mathbf{a_1}$**,... ,**$\mathbf{a_i}$ trở nên có thứ tự. Vị trí này chính là vị trí giữa hai phần tử $\mathbf{a_{k-1}}$ và $\mathbf{a_k}$ thỏa $\mathbf{a_{k-1} < a_i < a_k}$ $(1 \leq k \leq i)$.

Chèn Trực Tiếp – Insertion Sort

- Bước 1: i = 1; //giả sử có đoạn a[1] đã được sắp
- Bước 2: x = a[i]; Tìm vị trí pos thích hợp trong đoạn a[1] đến a[i-1] để chèn a[i] vào
- Bước 3: Dời chỗ các phần tử từ a[pos] đến a[i-1] sang phải 1 vị trí để dành chỗ cho a[i]
- Bước 4: a[pos] = x; //có đoạn a[1]..a[i] đã được sắp
- Bước 5: i = i+1;

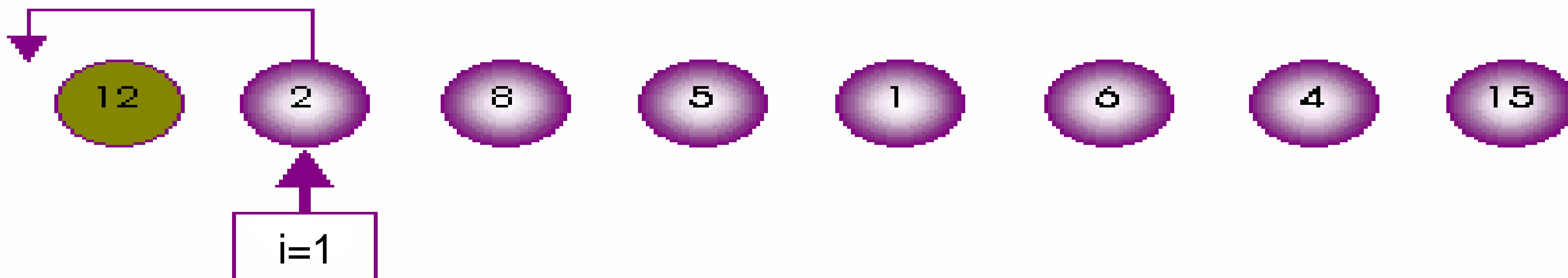
  Nếu i < n : Lặp lại Bước 2

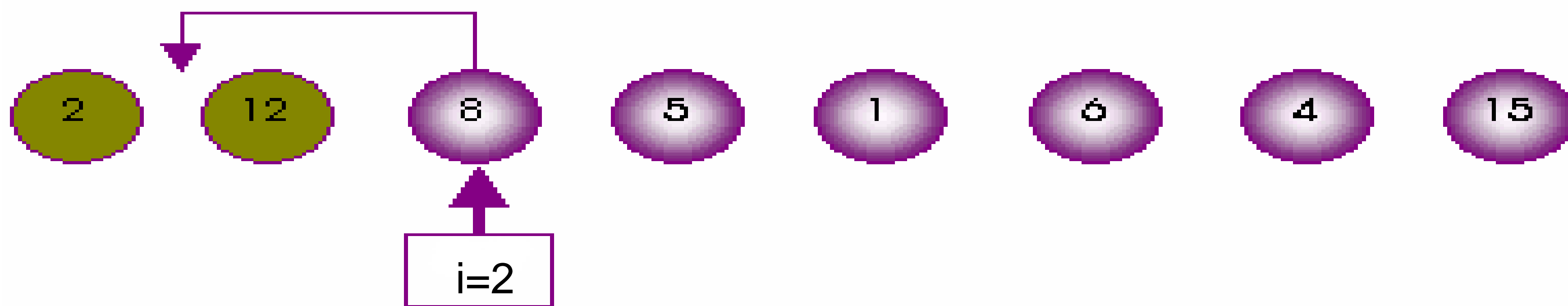
  Ngược lại : Dừng

Chèn Trực Tiếp – Insertion Sort

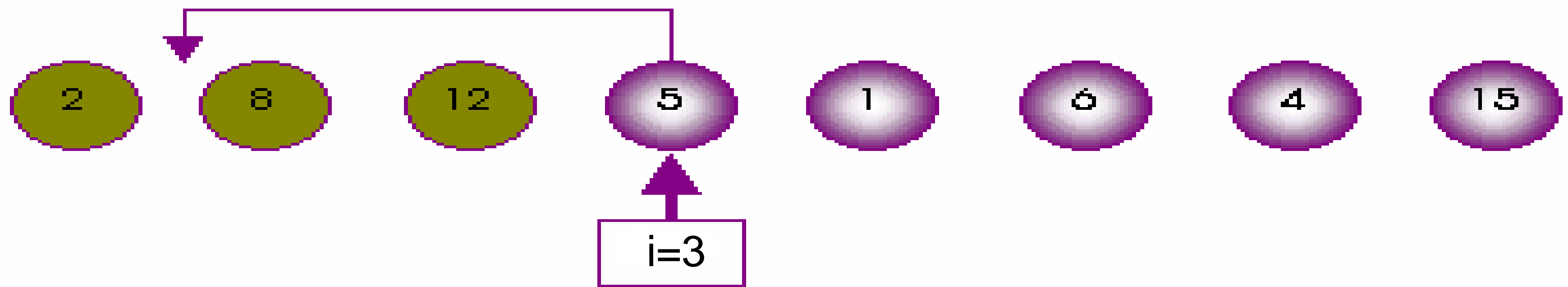
- Cho dãy số :

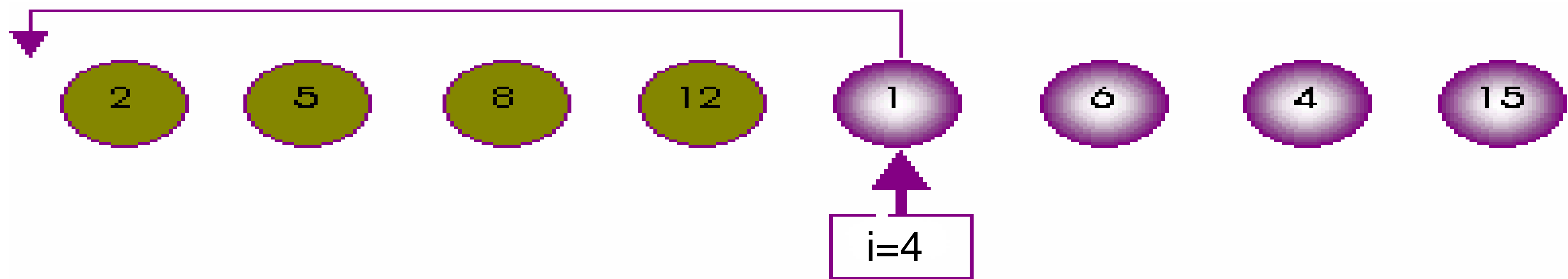
12 2 8 5 1 6 4 15

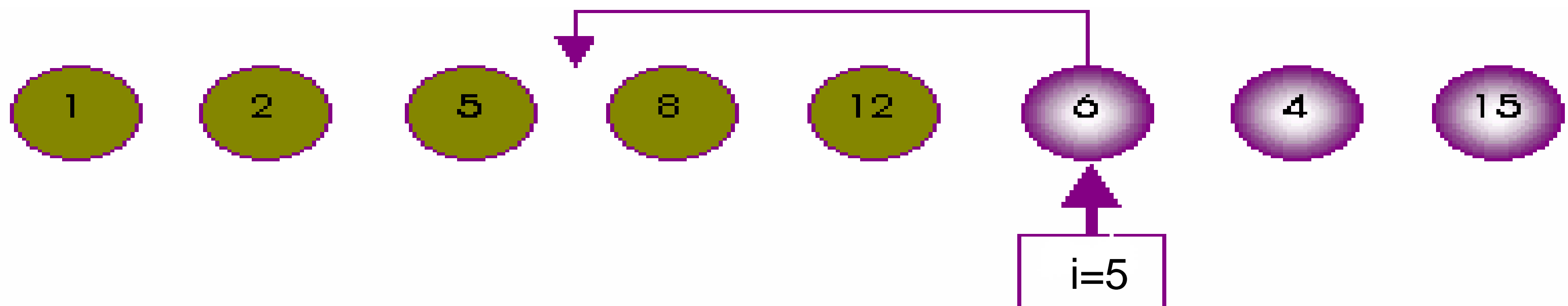




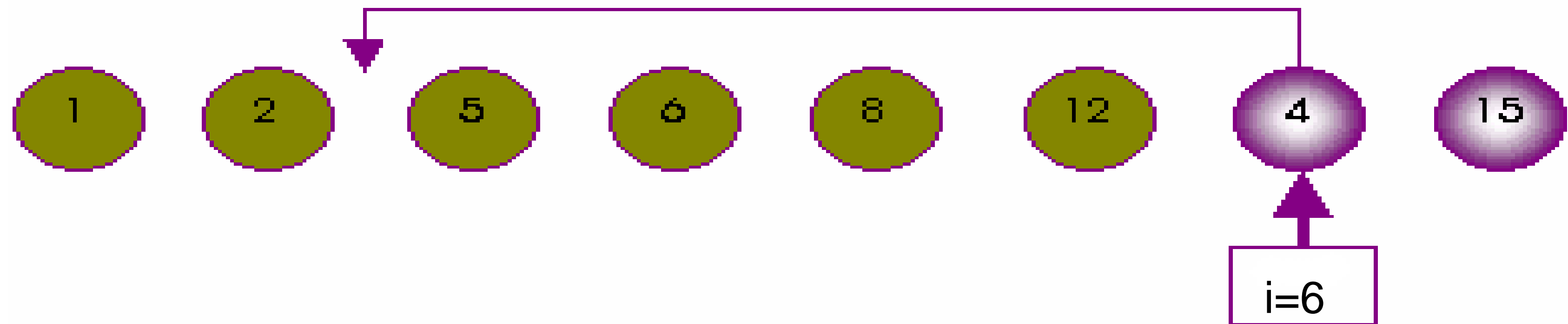
Chèn Trực Tiếp – Insertion Sort

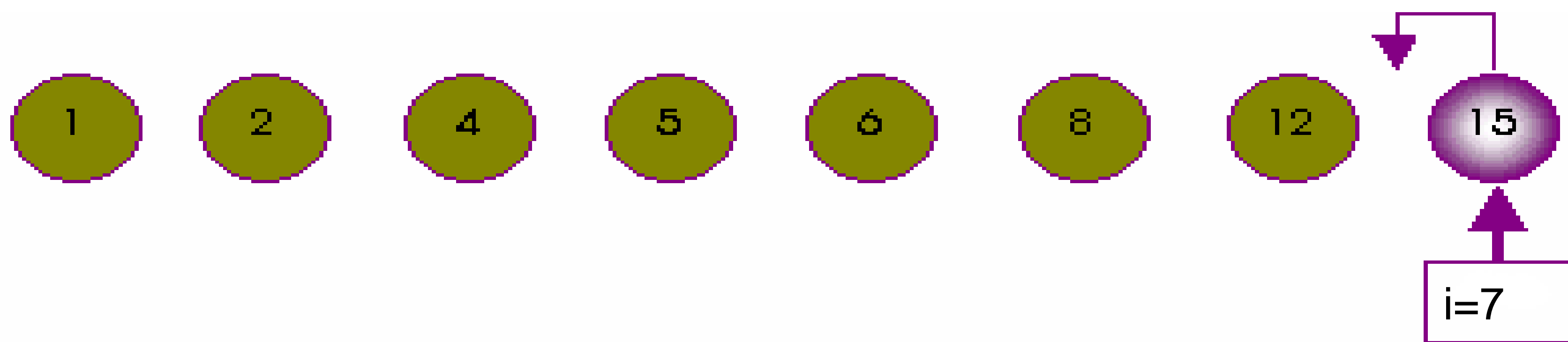






Chèn Trực Tiếp – Insertion Sort

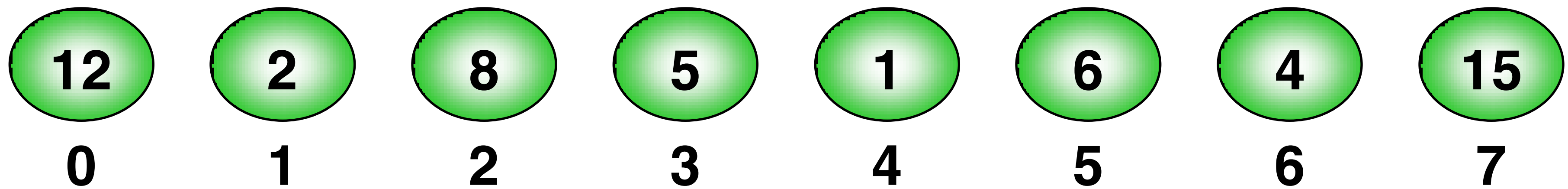




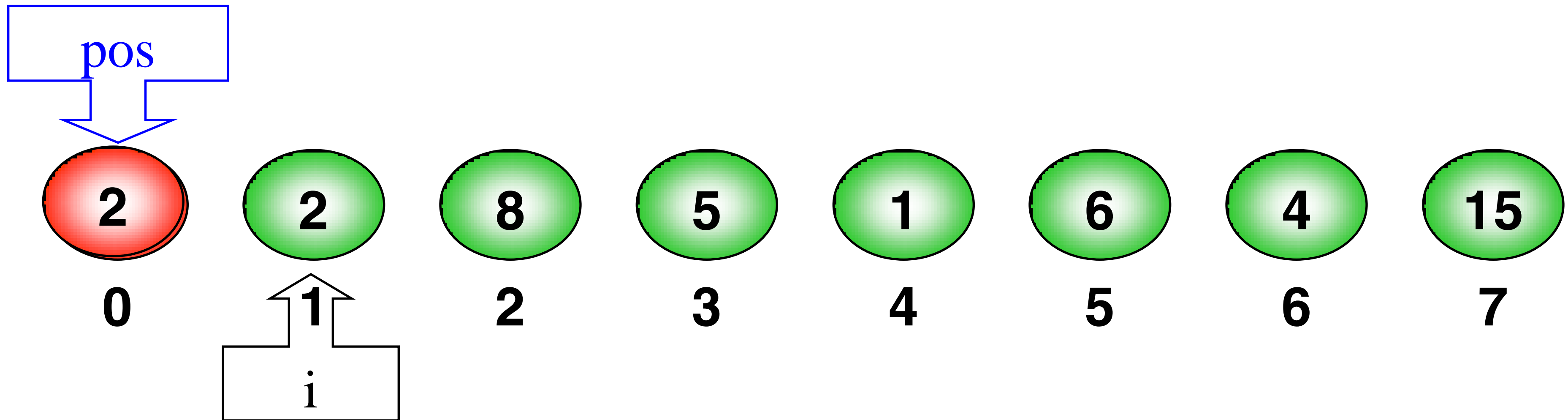
Cài Đặt Thuật Toán Chèn Trực Tiếp

```
void InsertionSort(int d, int n )
    {     int pos, i;
          int x;//lưu giá trị a[i] tránh bị ghi đè khi dời chỗ các phần tử.
          for(i=1 ; i<n ; i++) //đoạn a[0] đã sắp
          {
                    x = a[i]; pos = i-1;
                    // tìm vị trí chèn x
                    while((pos >= 0)&&(a[pos] > x))
                    {//kết hợp dời chỗ các phần tử sẽ đứng sau x trong dãy
    mới
                              a[pos+1] = a[pos];
                              pos--;
                    }
                    a[pos+1] = x; // chèn x vào dãy
          }
    }
```

Minh Họa Thuật Toán Insertion Sort

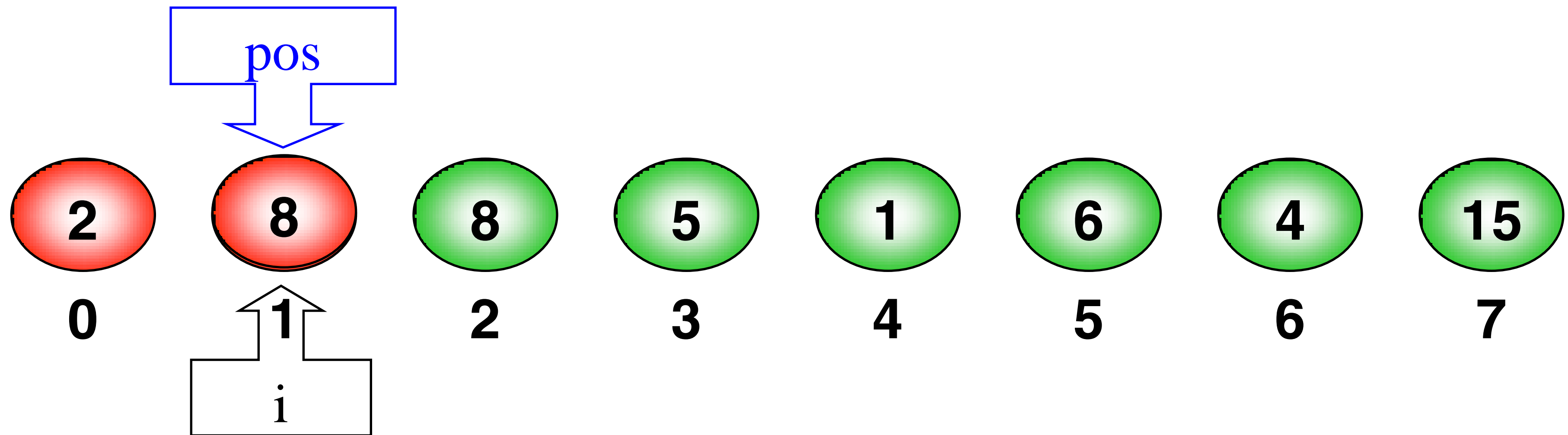


Minh Họa Thuật Toán Insertion Sort



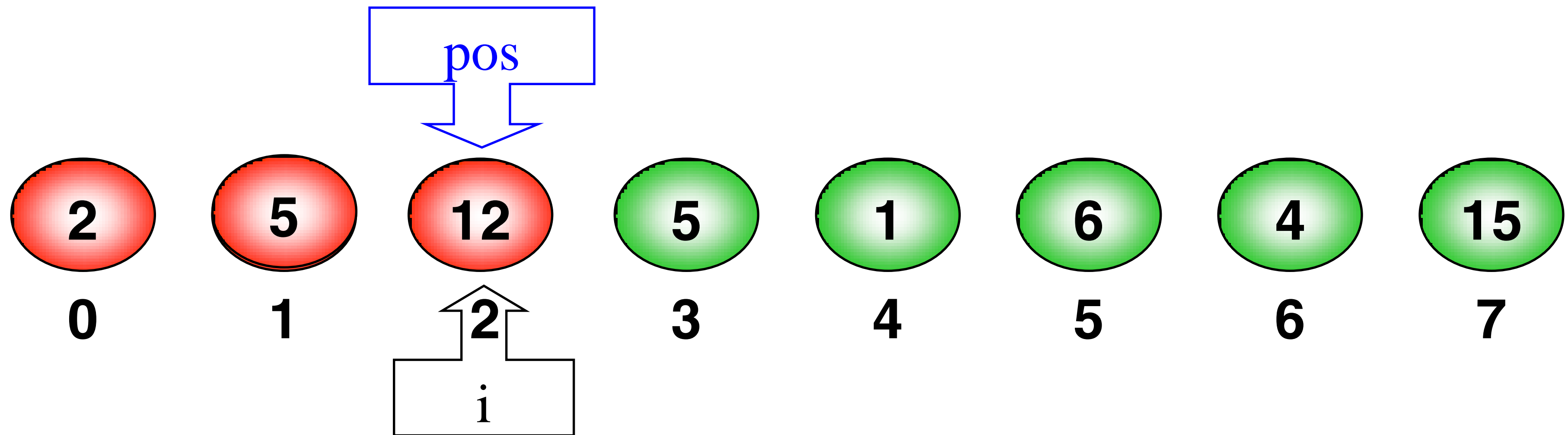
X

Minh Họa Thuật Toán Insertion Sort



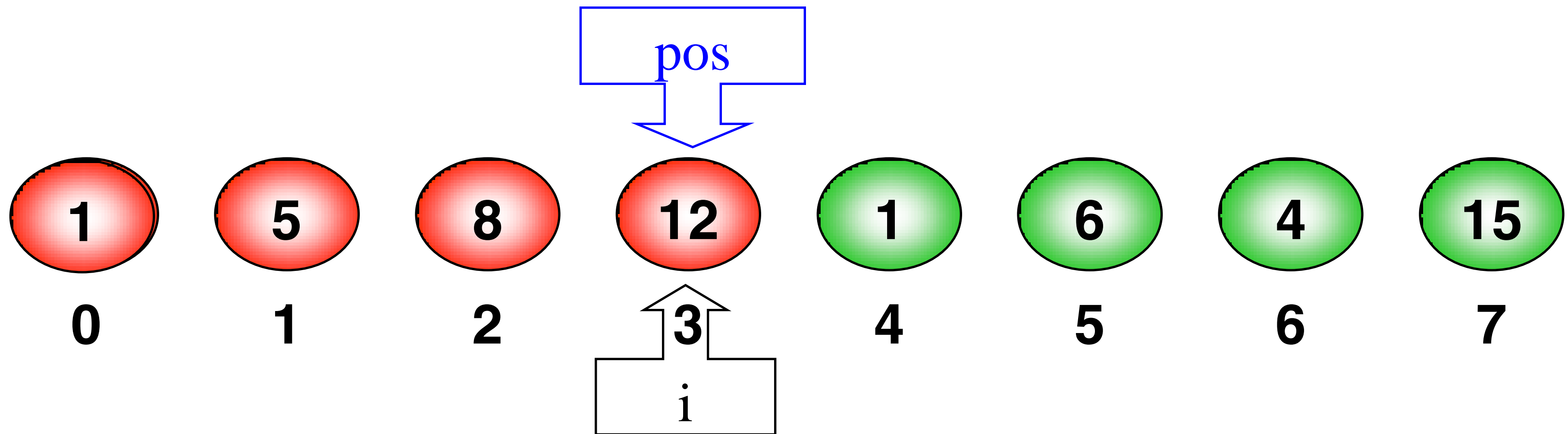
X

Minh Họa Thuật Toán Insertion Sort

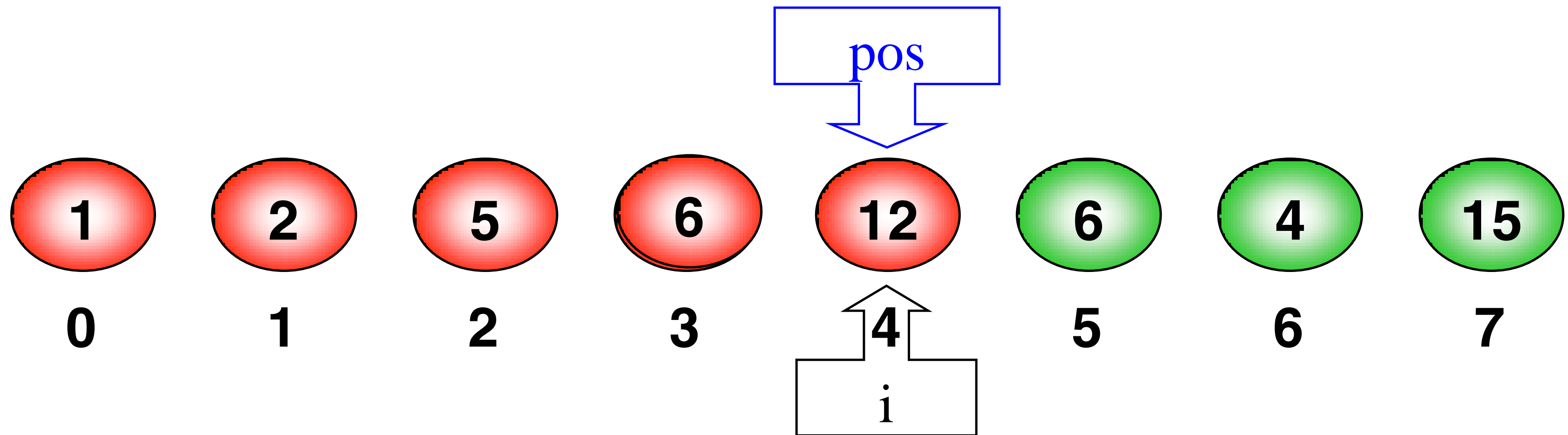


X

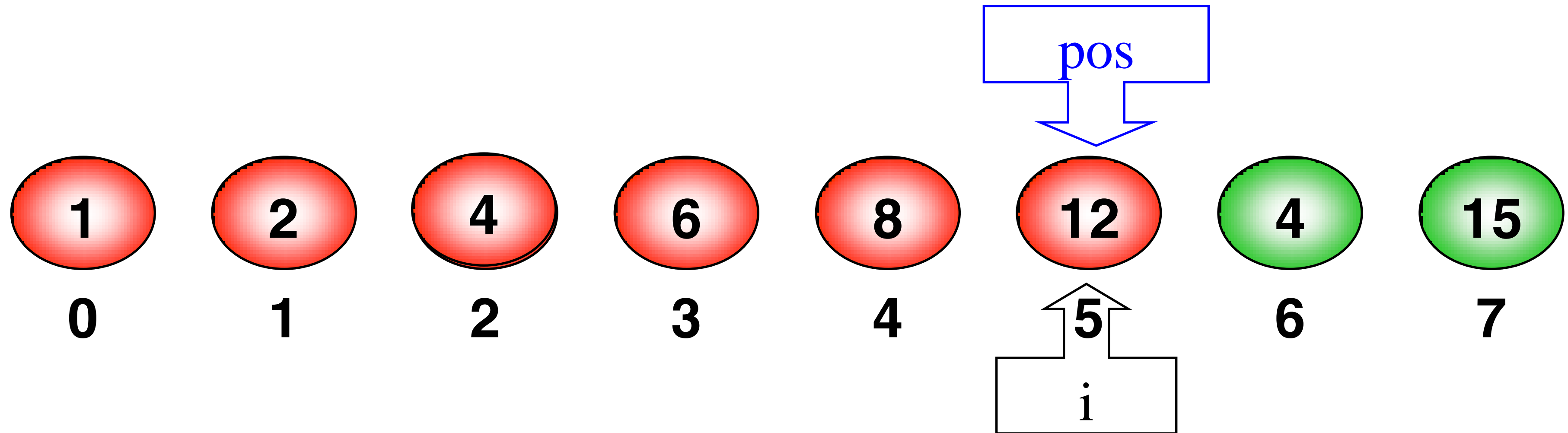
Minh Họa Thuật Toán Insertion Sort



Minh Họa Thuật Toán Insertion Sort

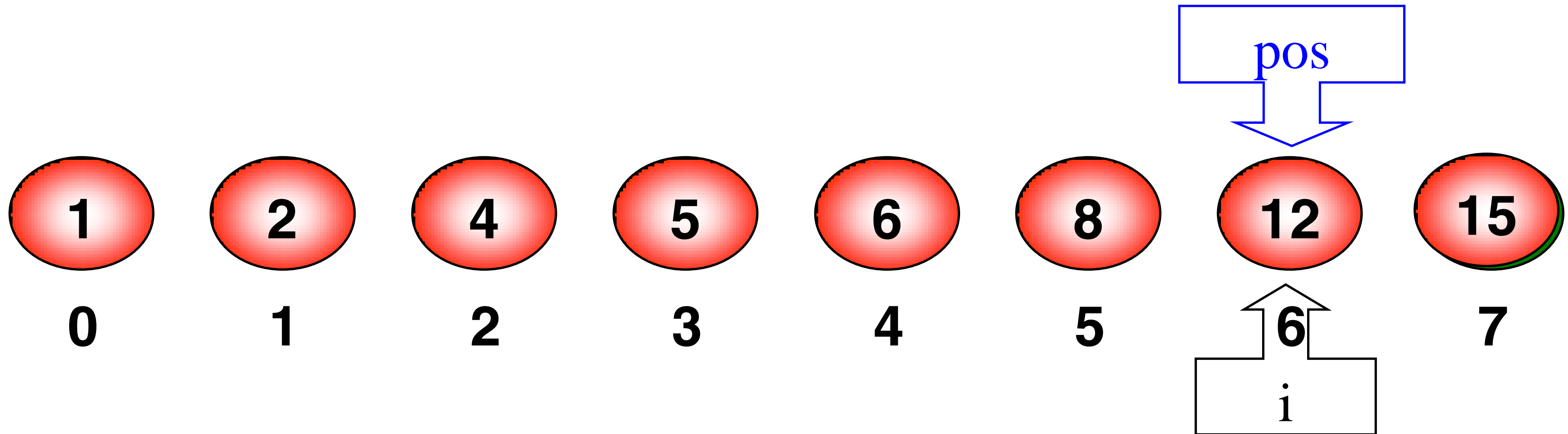


Minh Họa Thuật Toán Insertion Sort



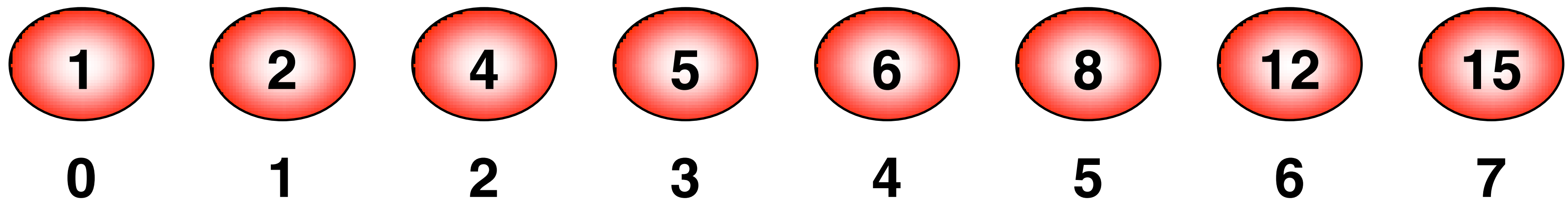
Minh Họa Thuật Toán Insertion Sort

Insert a[8] into (0, 6)



X

Minh Họa Thuật Toán Insertion Sort



Độ Phức Tạp Của Insertion Sort

Trường hợp	Số phép so sánh	Số phép gán
Tốt nhất	$\sum_{i=1}^{n-1} 1 = n-1$	$\sum_{i=1}^{n-1} 2 = 2(n-1)$
Xấu nhất	$\sum_{i=1}^{n-1} (i-1) = \frac{n(n-1)}{2}$	$\sum_{i=1}^{n-1} (i+1) = \frac{n(n+1)}{2} - 1$

Các Thuật Toán Sắp Xếp

1. Đổi chỗ trực tiếp – Interchange Sort
2. Chọn trực tiếp – Selection Sort
3. Nổi bọt – Bubble Sort
4. Shaker Sort
5. Chèn trực tiếp – Insertion Sort
6. **Chèn nhị phân – Binary Insertion Sort**
7. Shell Sort
8. Heap Sort
9. Quick Sort
10. Merge Sort
11. Radix Sort

Chèn Nhị Phân – Binary Insertion Sort

```
void            BInsertionSort(int a[],int n )
{
    int l,r,m,i;
    int x;//lưu giá trị a[i] tránh bị ghi đè khi dời chỗ các phần tử.
    for(int i=1 ; i<n ; i++)
    {       x = a[i]; l = 0; r = i-1;
            while(l<=r)       // tìm vị trí chèn x
            {       m = (l+r)/2;     // tìm vị trí thích hợp m
                    if(x < a[m]) r = m-1;
                    else      l = m+1;
            }
            for(int j = i-1 ; j >=l ; j--)
                    a[j+1] = a[j];// dời các phần tử sẽ đứng sau x
            a[l] = x;                    // chèn x vào dãy
    }
}
```

Các Thuật Toán Sắp Xếp

1. Đổi chỗ trực tiếp – Interchange Sort
2. Chọn trực tiếp – Selection Sort
3. Nổi bọt – Bubble Sort
4. Shaker Sort
5. Chèn trực tiếp – Insertion Sort
6. Chèn nhị phân – Binary Insertion Sort
7. **Shell Sort**
8. Heap Sort
9. Quick Sort
10. Merge Sort
11. Radix Sort

Shell Sort

- Cải tiến của phương pháp chèn trực tiếp
- Ý tưởng:
  - Phân hoạch dãy thành các dãy con
  - Sắp xếp các dãy con theo phương pháp chèn trực tiếp
  - Dùng phương pháp chèn trực tiếp sắp xếp lại cả dãy.

Shell Sort

- Phân chia dãy ban đầu thành những dãy con gồm các phần tử ở cách nhau **h** vị trí
- Dãy ban đầu : $a_1, a_2, ..., a_n$ được xem như sự xen kẽ của các dãy con sau :
  - Dãy con thứ nhất : $a_1\ a_{h+1}\ a_{2h+1}\ ...$
  - Dãy con thứ hai : $a_2\ a_{h+2}\ a_{2h+2}\ ...$
  - ....
  - Dãy con thứ h : $a_h\ a_{2h}\ a_{3h}\ ...$

Shell Sort

- Tiến hành sắp xếp các phần tử trong cùng dãy con sẽ làm cho các phần tử được đưa về vị trí đúng tương đối
- Giảm khoảng cách **h** để tạo thành các dãy con mới
- Dừng khi h=1

Shell Sort

- Giả sử quyết định sắp xếp **k** bước, các khoảng cách chọn phải thỏa điều kiện :

  $h_i > h_{i+1}$ và $h_k = 1$

- $h_i = (h_{i-1} - 1)/3$ và $h_k = 1$, $k = \log_3 n - 1$

  Ví dụ :127, 40, 13, 4, 1

- $h_i = (h_{i-1} - 1)/2$ và $h_k = 1$, $k = \log_2 n - 1$

  Ví dụ : 15, 7, 3, 1

Shell Sort

- h có dạng 3i+1: 364, 121, 40, 13, 4, 1
- Dãy fibonaci: 34, 21, 13, 8, 5, 3, 2, 1
- h là dãy các số nguyên tố giảm dần đến 1: 13, 11, 7, 5, 3, 1.

Shell Sort

- Bước 1: Chọn **k** khoảng cách h[1], h[2], ..., h[k];

  i = 1;

- Bước 2: Phân chia dãy ban đầu thành các dãy con cách nhau h[i] khoảng cách.

  Sắp xếp từng dãy con bằng phương pháp chèn trực tiếp;

- Bước 3 : i = i+1;

  Nếu i > k : Dừng

  Ngược lại : Lặp lại Bước 2.

Shell Sort

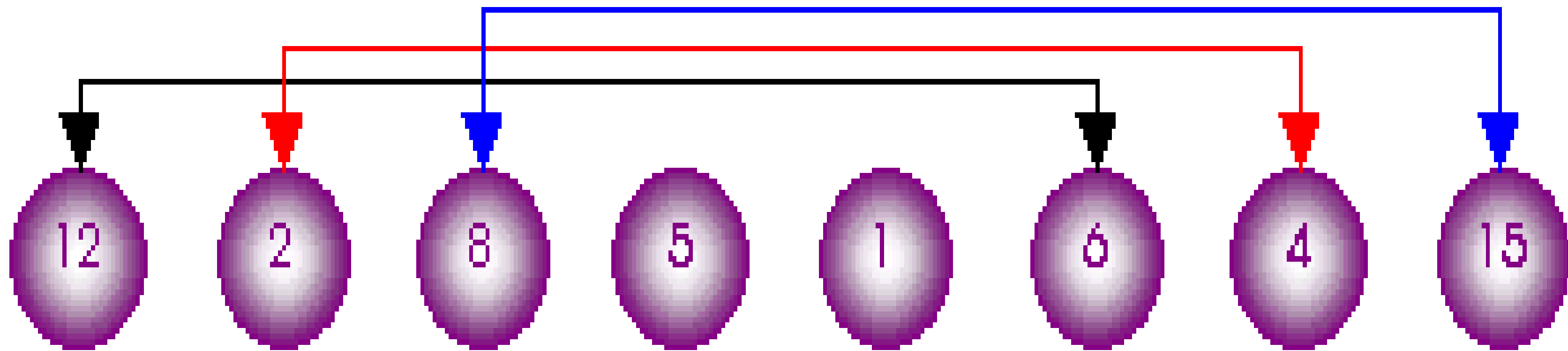
- Cho dãy số a:

  12 2 8 5 1 6 4 15

- Giả sử chọn các khoảng cách là 5, 3, 1

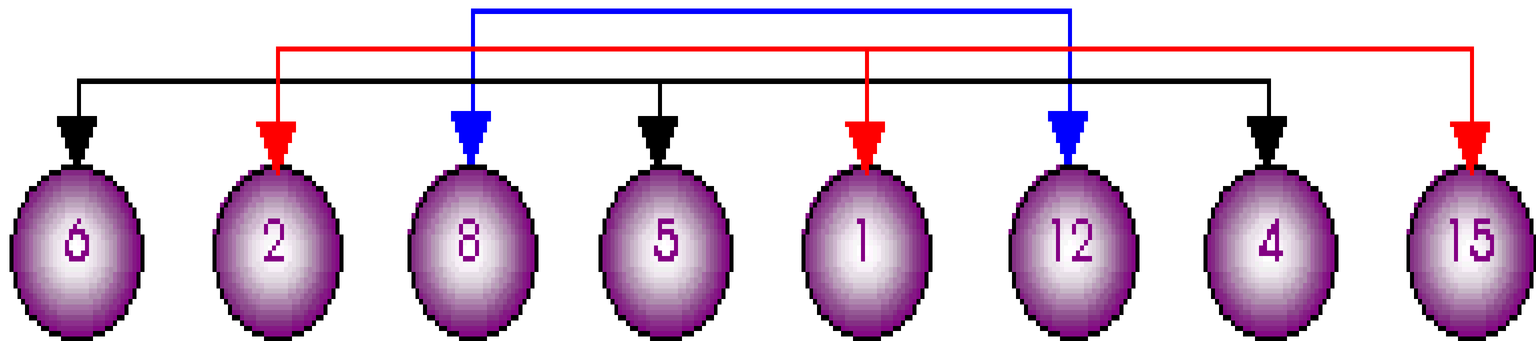
Shell Sort

- h = 5 : xem dãy ban đầu như các dãy con



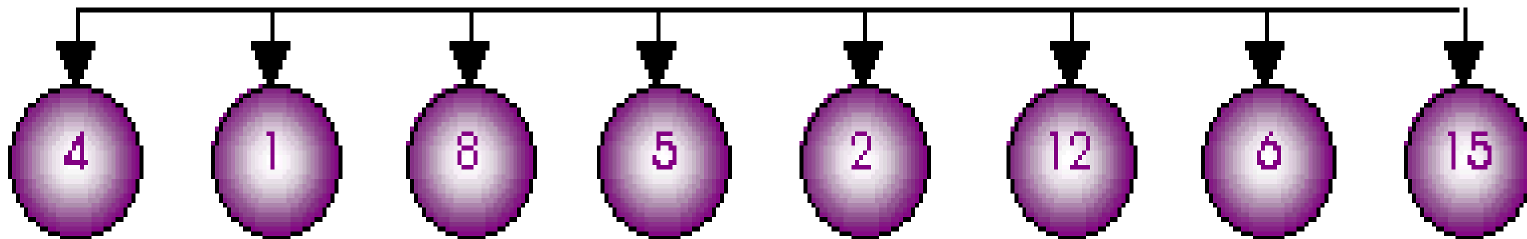
Shell Sort

- h = 3 : (sau khi đã sắp xếp các dãy con ở bước trước)



Shell Sort

- <u>h = 1 :</u> (sau khi đã sắp xếp các dãy con ở bước trước



Shell Sort

4 1 8 5 2 12 6 15

1 4 8 5 2 12 6 15

1 4 5 8 2 12 6 15

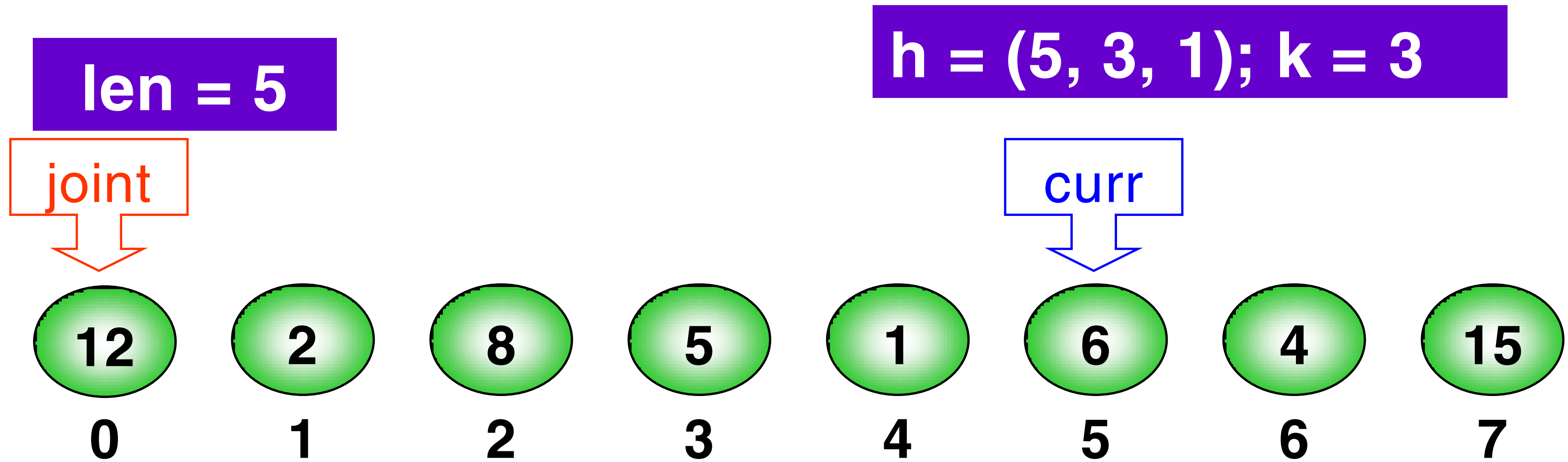
1 2 4 5 8 12 6 15

1 2 4 5 6 8 12 15

Shell Sort

```
void ShellSort(int a[],int n, int h[], int k)
{     int  step,i,j, x,len;
      for (step = 0 ; step <k; step++)
      {     len = h[step];
            for (i = len; i<n; i++)
            {
                        x = a[i];
                        j = i-len; // a[j] đứng kề trước a[i] trong cùng dãy con
                        while ((x<a[j])&&(j>=0))// sắp xếp dãy con chứa  x
                        {                       // bằng phương pháp chèn trực tiếp
                                    a[j+len]  = a[j];
                                    j = j - len;
                        }
                        a[j+len] = x;
            }
      }
}
```

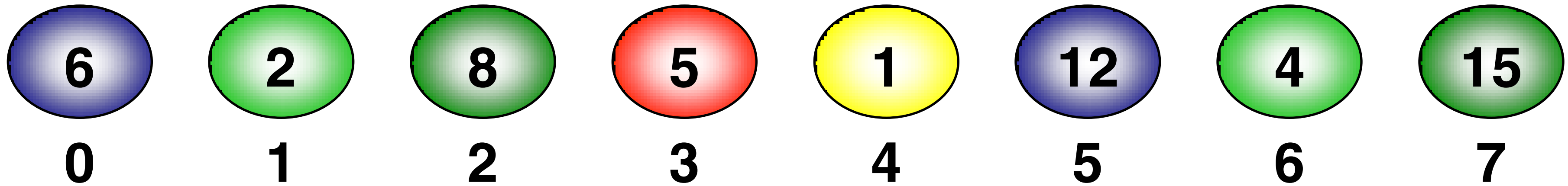
Shell Sort – Ví Dụ



Shell Sort – Ví Dụ

len = 5;

h = (5, 3, 1); k = 3



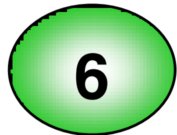
Shell Sort – Ví Dụ

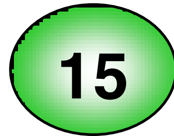
h = (5, 3, 1); k = 3

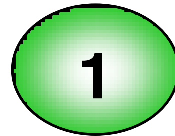
len = 3

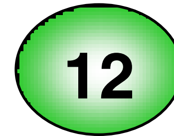
joint

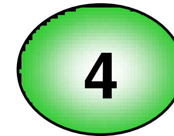
curr

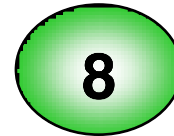






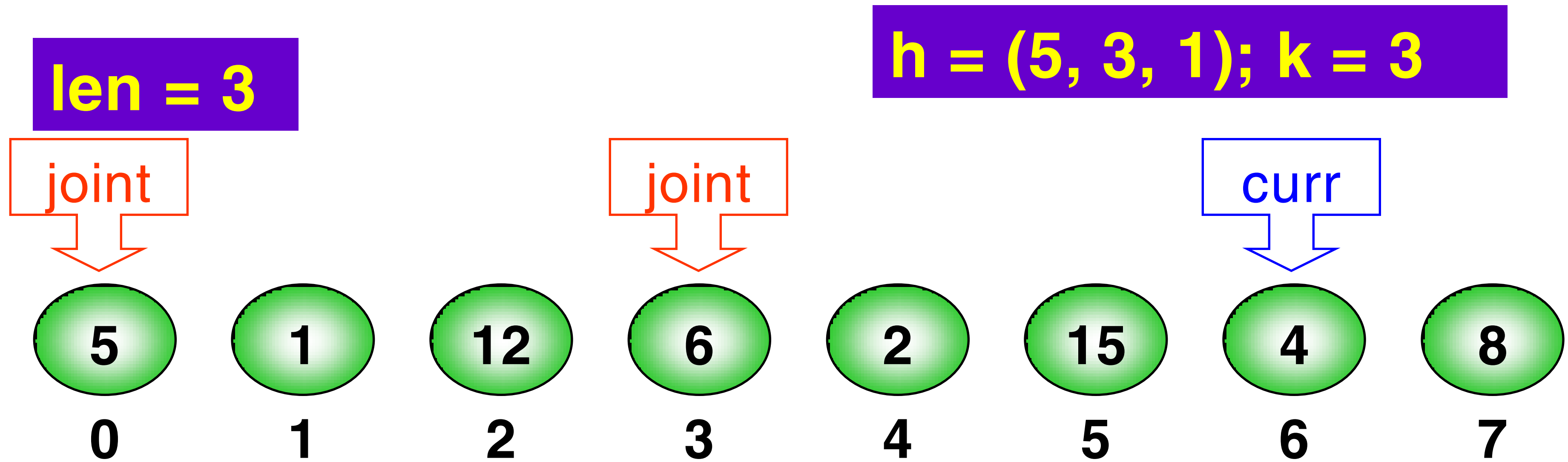






0 1 2 3 4 5 6 7

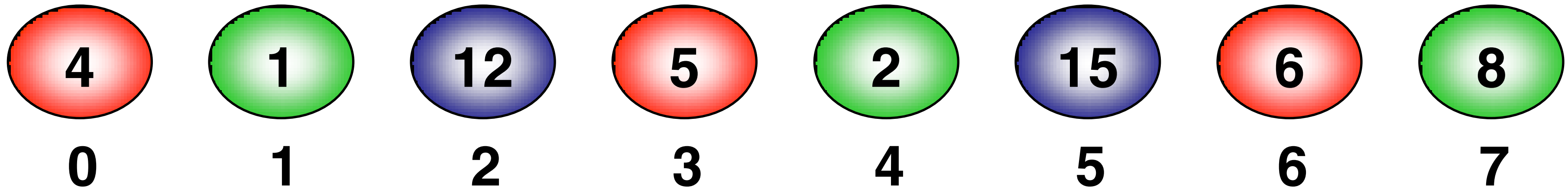
Shell Sort – Ví Dụ



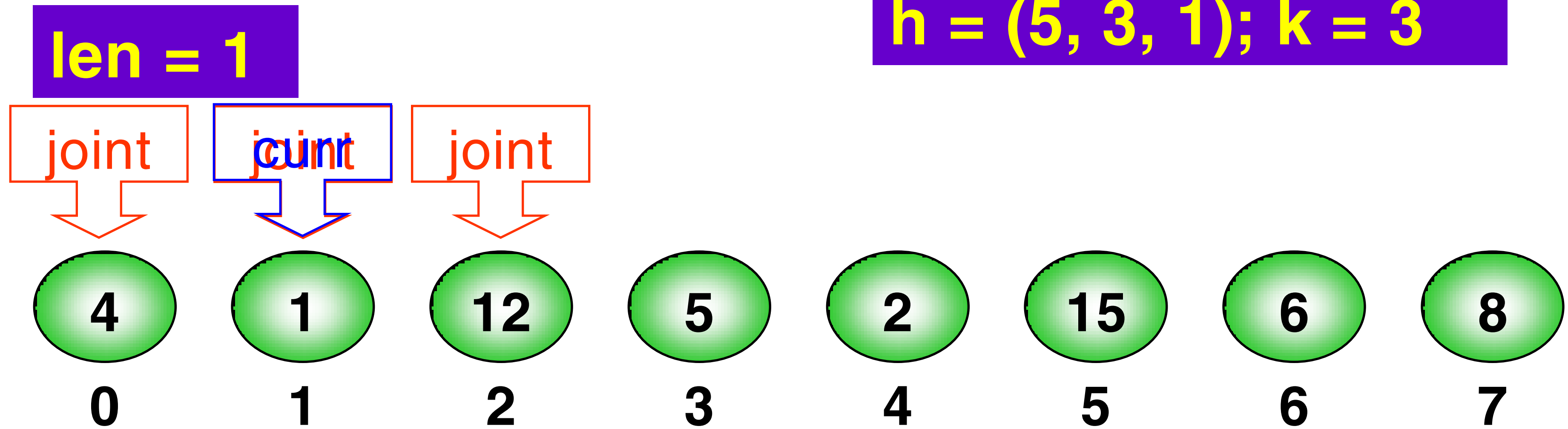
Shell Sort – Ví Dụ

len = 3

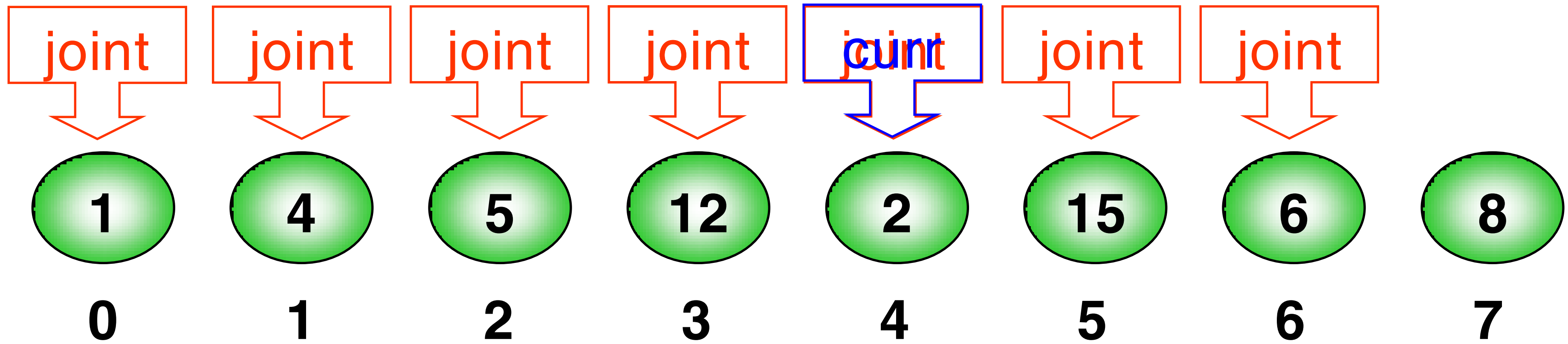
h = (5, 3, 1); k = 3



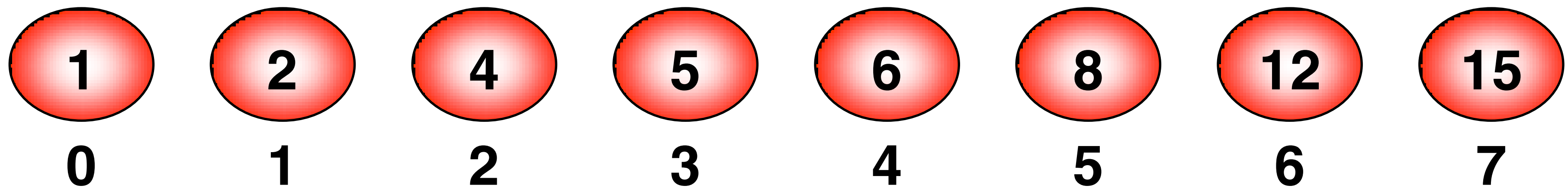
Shell Sort – Ví Dụ



Shell Sort – Ví Dụ



Shell Sort – Ví Dụ



Các Thuật Toán Sắp Xếp

1. Đổi chỗ trực tiếp – Interchange Sort
2. Chọn trực tiếp – Selection Sort
3. Nổi bọt – Bubble Sort
4. Shaker Sort
5. Chèn trực tiếp – Insertion Sort
6. Chèn nhị phân – Binary Insertion Sort
7. Shell Sort
8. **Heap Sort**
9. Quick Sort
10. Merge Sort
11. Radix Sort

Thuật Toán Sắp Xếp Heap Sort

- Heap Sort tận dụng được các phép so sánh ở bước i-1 mà thuật toán sắp xếp chọn trực tiếp không tận dụng được
- Để làm được điều này Heap sort thao tác dựa trên cây.

Thuật Toán Sắp Xếp Heap Sort

- Cho dãy số : 12 2 8 5 1 6 4 15

12	2	8	5	1	6	4	15
0	1	2	3	4	5	6	7

12 a[0]
2 a[1]
8 a[2]
5 a[3]
1 a[4]
6 a[5]
4 a[6]
15 a[7]

Thuật toán sắp xếp Heap Sort

- Ở cây trên, phần tử ở mức i chính là phần tử lớn trong cặp phần tử ở mức i +1, do đó phần tử ở nút gốc là phần tử lớn nhất.
- Nếu loại bỏ gốc ra khỏi cây, thì việc cập nhật cây chỉ xảy ra trên những nhánh liên quan đến phần tử mới loại bỏ, còn các nhánh khác thì bảo toàn.
- Bước kế tiếp có thể sử dụng lại kết quả so sánh của bước hiện tại.
- Vì thế độ phức tạp của thuật toán O(nlog$_2$n)

Các Bước Thuật Toán

- Giai đoạn 1 : Hiệu chỉnh dãy số ban đầu thành heap
- Giai đoạn 2: Sắp xếp dãy số dựa trên heap:
  - Bước 1:Đưa phần tử lớn nhất về vị trí đúng ở cuối dãy:

    r = n-1; Swap (a_1 , a_r);
  - Bước 2: Loại bỏ phần tử lớn nhất ra khỏi heap: r = r-1;

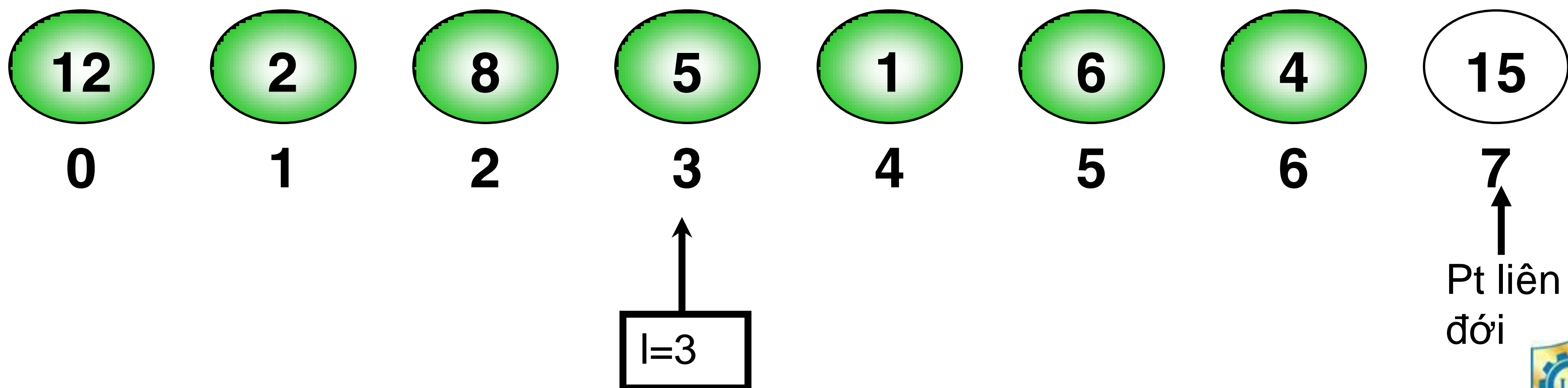
    Hiệu chỉnh phần còn lại của dãy từ a1 , a2 ... ar thành một heap.
  - Bước 3:

    Nếu r>1 (heap còn phần tử): Lặp lại Bước 2

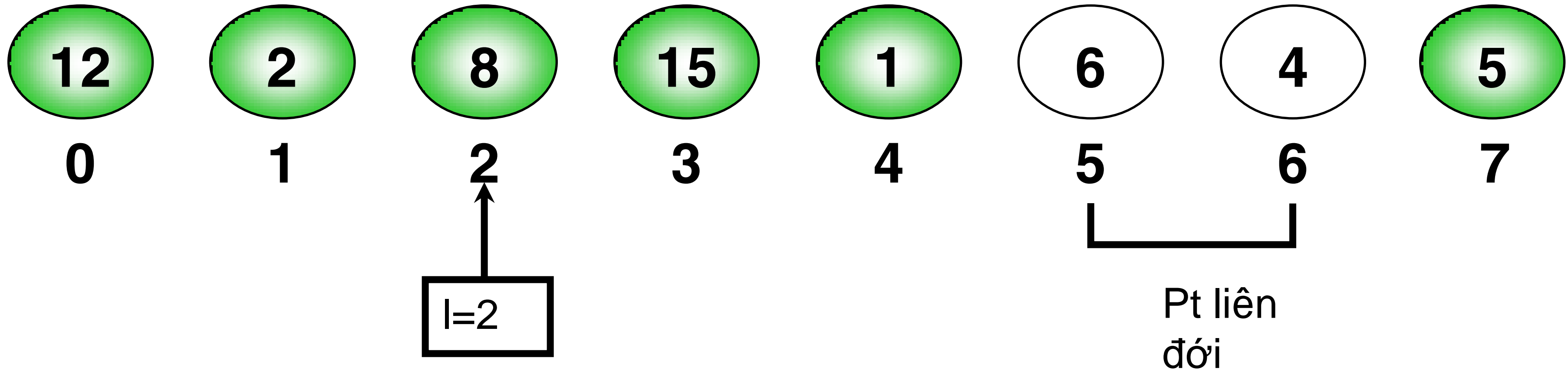
    Ngược lại : Dừng

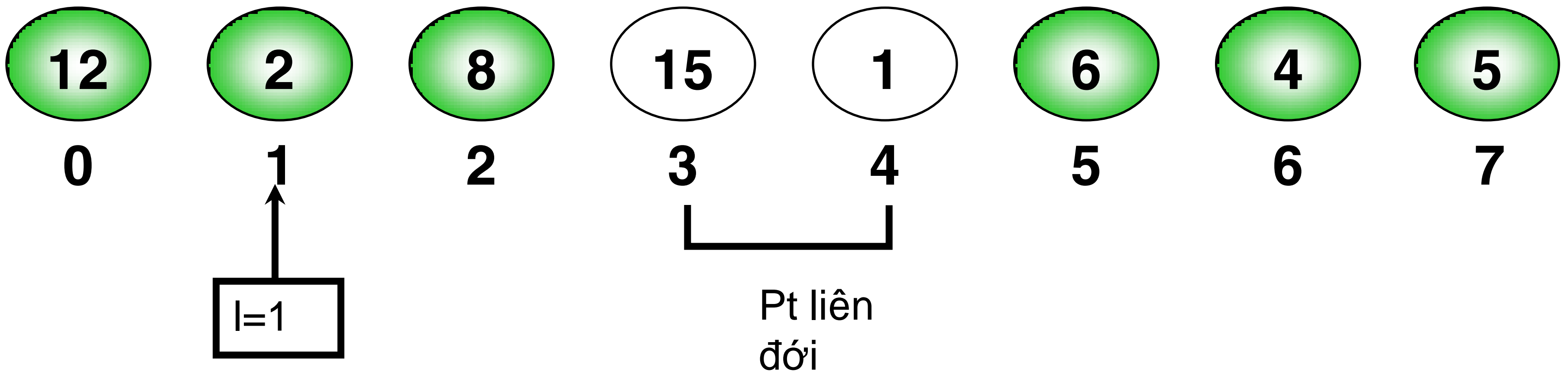
Minh Họa Thuật Toán

- **Heap:** Là một dãy các phần tử $a_l, a_{l+1}, \ldots, a_r$ thoả các quan hệ với mọi i , r]:
  - $a_i \quad a_{2i+1}$
  - $a_i \quad a_{2i+2}$ // (a_i , a_{2i+1}), (a_i , a_{2i+2}) là các cặp phần tử liên đới
- Cho dãy số : 12 2 8 5 1 6 4 15
- Giai đoạn 1: Hiệu chỉnh dãy ban đầu thành Heap

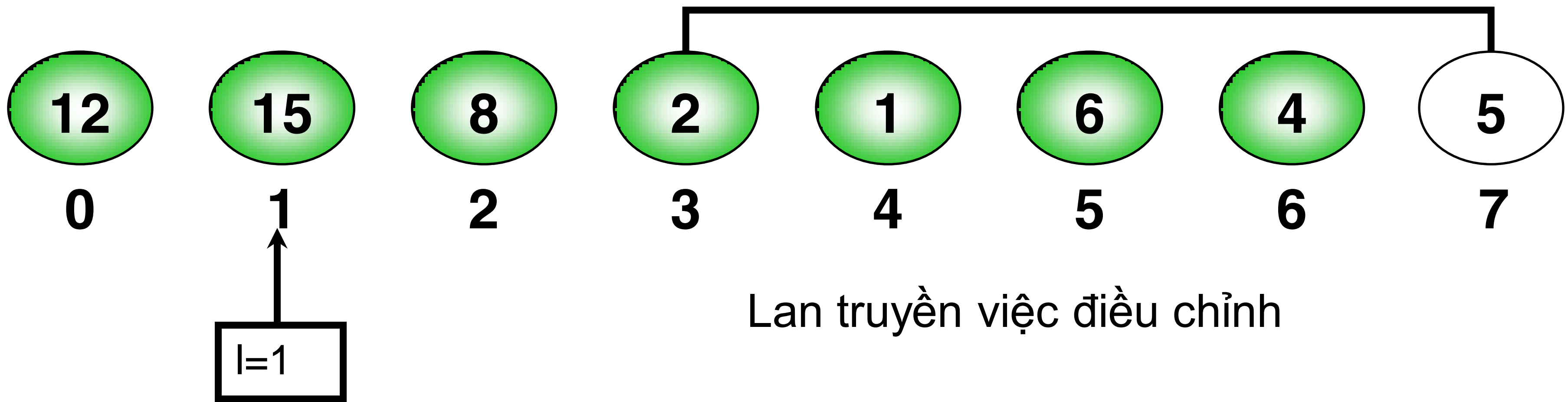


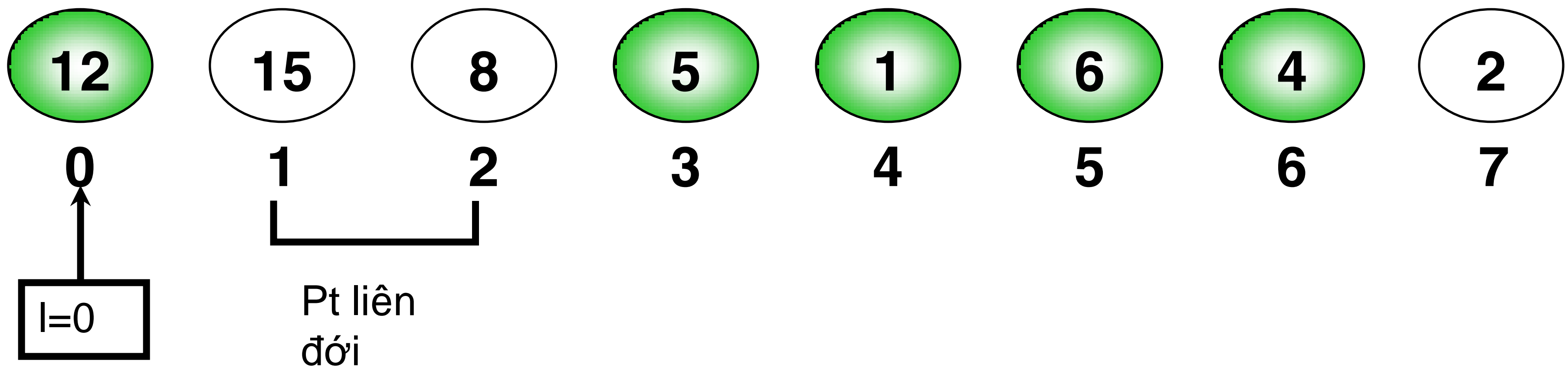
Minh Họa Thuật Toán



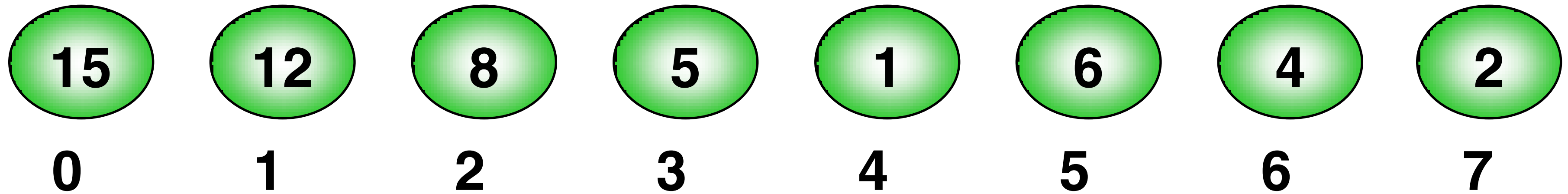


Minh Họa Thuật Toán

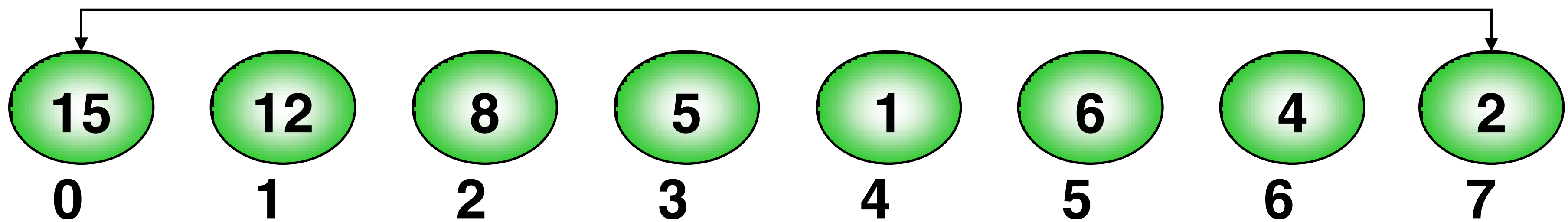


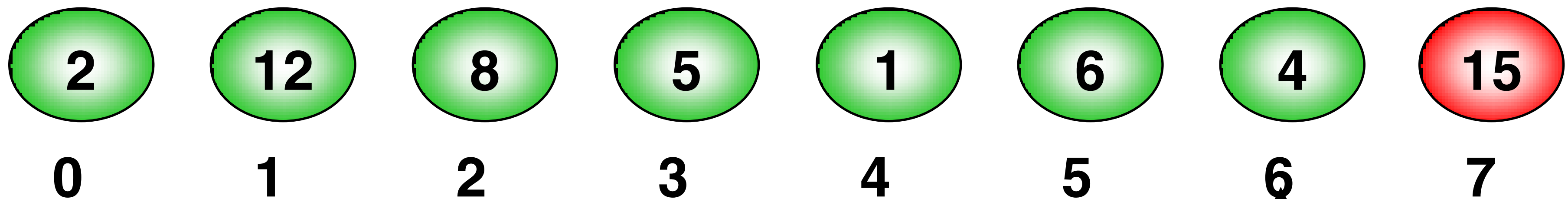


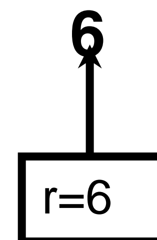
Minh Họa Thuật Toán



➢ Giai đoạn 2: Sắp xếp dãy số dựa trên Heap







Minh Họa Thuật Toán

➢Hiệu chỉnh Heap

2	12	8	5	1	6	4	15
0	1	2	3	4	5	6	7

l=2

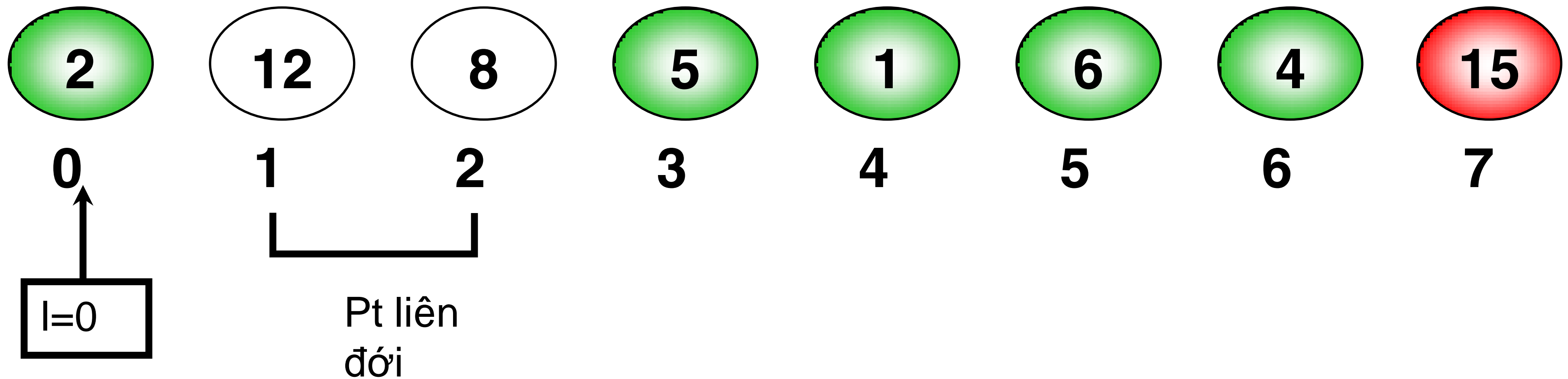
Pt liên đới

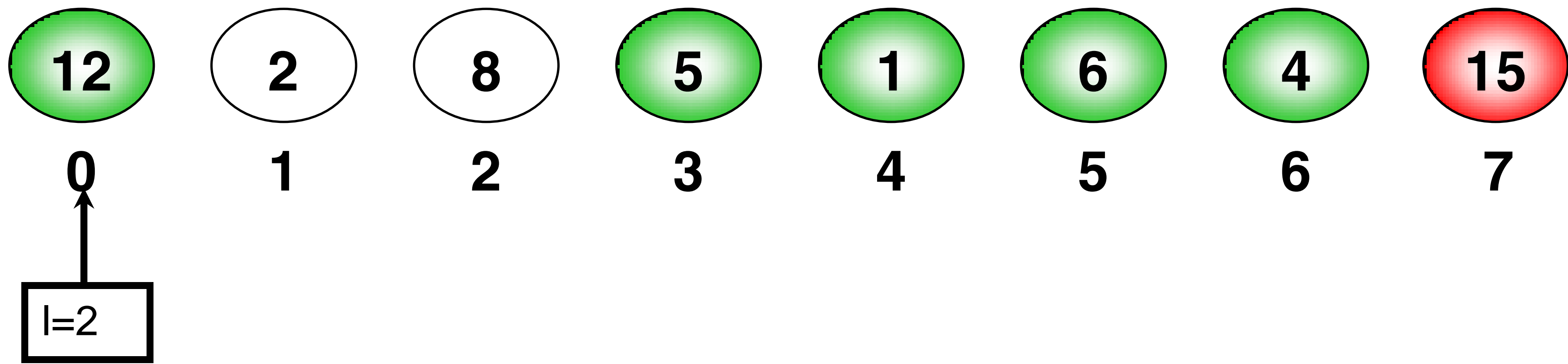
2	12	8	5	1	6	4	15
0	1	2	3	4	5	6	7

l=2

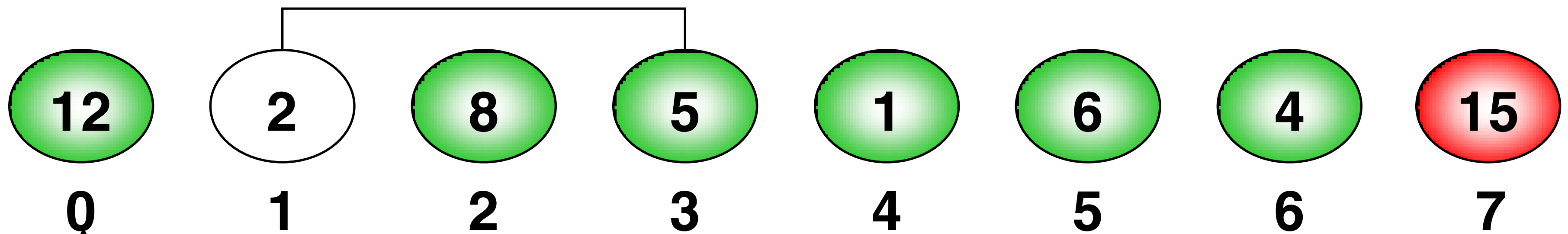
Pt liên đới

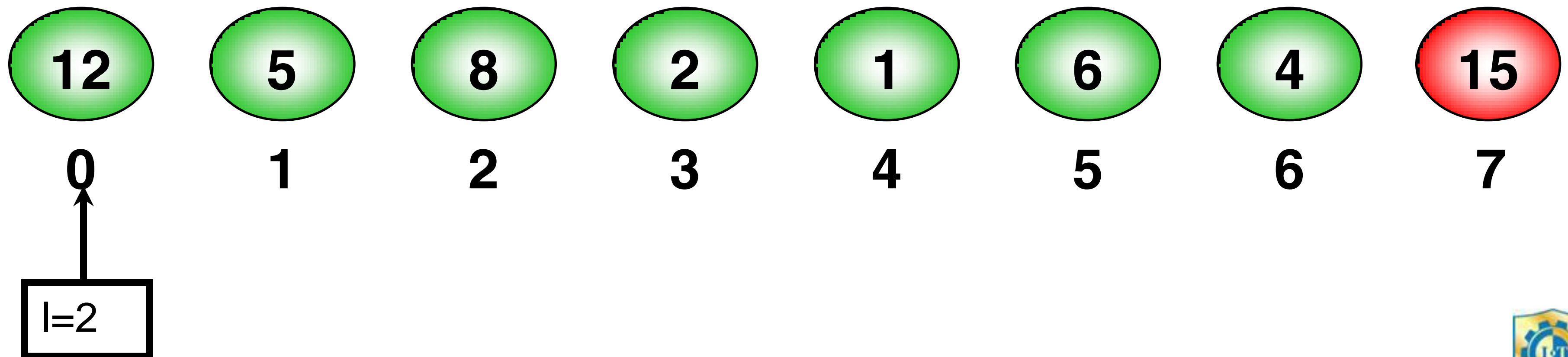
Minh Họa Thuật Toán



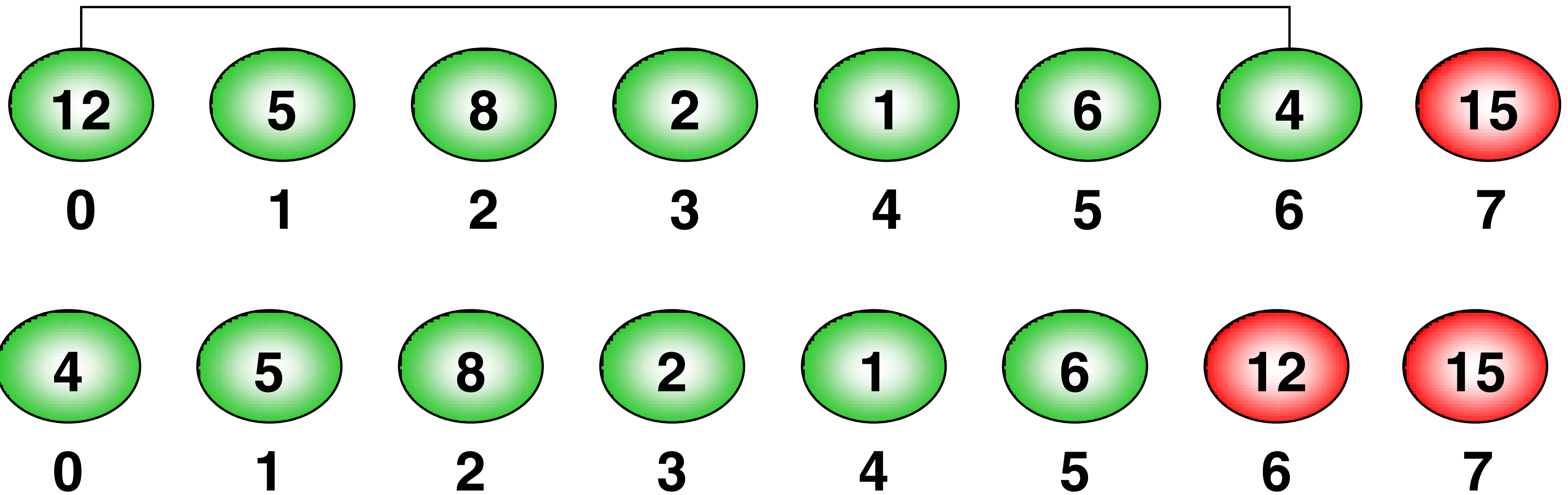


Minh Họa Thuật Toán

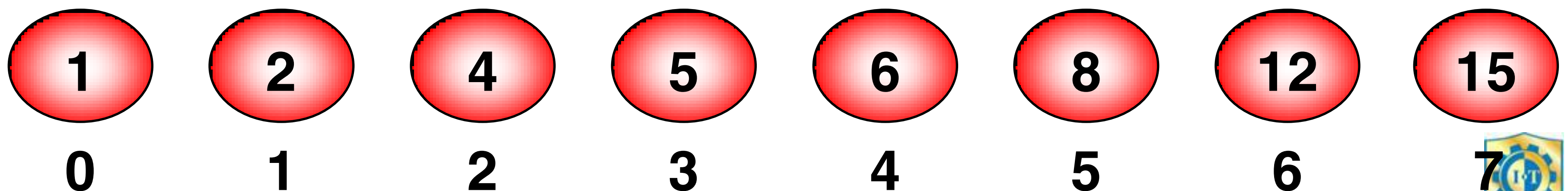




Minh Họa Thuật Toán



- Thực hiện với r= 5,4,3,2 ta được



Cài Đặt Thuật Toán

- Hiệu chỉnh $a_l, a_{l+1},..,a_r$ thành Heap

```
void shift(int a[],int l,int r)
{
    int x,i,j;
    i=l;
    j=2*i+1;
    x=a[i];
    while(j<=r)
    {       if(j<r)
            if(a[j]<a[j+1]) //tim phan tu lon nhat a[j] va a[j+1]
```

Cài Đặt Thuật Toán

```
j++; //luu chi so cua phan tu nho nhat trong hai phan tu
if(a[j]<=x) return;
else
{       a[i]=a[j];
        a[j]=x;
        i=j;
        j=2*i+1;
        x=a[i];
 }
    }
}
```

Cài Đặt Thuật Toán

- Hiệu chỉnh $a_0,..a_{n-1}$ Thành Heap

```
void CreateHeap(int a[],int n)
{    int l;
     l=n/2-1;
     while(l>=0)
     {
          shift(a,l,n-1);
          l=l-1;
     }
}
```

Cài Đặt Thuật Toán

- Hàm HeapSort

```
void HeapSort(int a[],int n)
{   int r;
    CreateHeap(a,n);
    r=n-1;
    while(r>0)
    {
            Swap(a[0],a[r]);//a[0] la nút gốc
            r--;
            if(r>0)
                    shift(a,0,r);
    }
}
```

Các Thuật Toán Sắp Xếp

1. Đổi chỗ trực tiếp – Interchange Sort
2. Chọn trực tiếp – Selection Sort
3. Nổi bọt – Bubble Sort
4. Shaker Sort
5. Chèn trực tiếp – Insertion Sort
6. Chèn nhị phân – Binary Insertion Sort
7. Shell Sort
8. Heap Sort
9. **Quick Sort**
10. Merge Sort
11. Radix Sort

Quick Sort

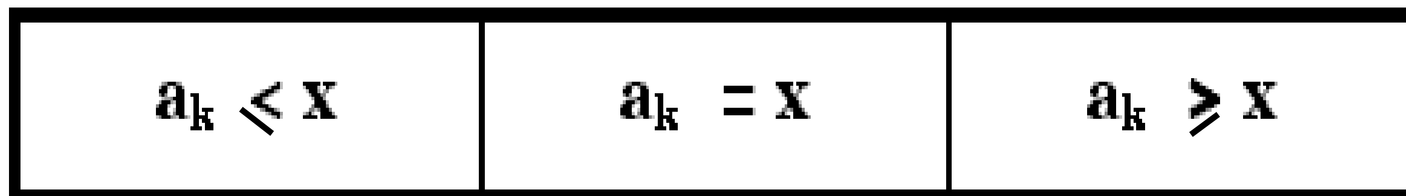
➢ Ý tưởng:

- Giải thuật QuickSort sắp xếp dãy $\mathbf{a_1, a_2 ..., a_N}$ dựa trên việc phân hoạch dãy ban đầu thành 3 phần :
  - Phần 1: Gồm các phần tử có giá trị bé hơn **x**
  - Phần 2: Gồm các phần tử có giá trị bằng **x**
  - Phần 3: Gồm các phần tử có giá trị lớn hơn **x**

với x là giá trị của một phần tử tùy ý trong dãy ban đầu.

Quick Sort - Ý Tưởng

- Sau khi thực hiện phân hoạch, dãy ban đầu được phân thành 3 đoạn:
  - 1. $a_k \leq x$, với k = 1 .. j
  - 2. $a_k = x$, với k = j+1 .. i-1
  - 3. a_k [illegible]x , với k = i..N



Quick Sort – Ý Tưởng

$a_k \le x$	$a_k = x$	$a_k \ge x$

- Đoạn thứ 2 đã có thứ tự.
- Nếu các đoạn 1 và 3 chỉ có 1 phần tử : đã có thứ tự

  → khi đó dãy con ban đầu đã được sắp.

Quick Sort – Ý Tưởng

$a_k \leq x$	$a_k = x$	$a_k \geq x$

- Đoạn thứ 2 đã có thứ tự.
- Nếu các đoạn 1 và 3 có nhiều hơn 1 phần tử thì dãy ban đầu chỉ có thứ tự khi các đoạn 1, 3 được sắp.
- Để sắp xếp các đoạn 1 và 3, ta lần lượt tiến hành việc phân hoạch từng dãy con theo cùng phương pháp phân hoạch dãy ban đầu vừa trình bày …

Giải Thuật Quick Sort

- <u>Bước 1</u>: Nếu left ≥ right //dãy có ít hơn 2 phần tử

  Kết thúc; //dãy đã được sắp xếp

- <u>Bước 2</u>: Phân hoạch dãy a_{left} … a_{right} thành các đoạn: a_{left}.. a_j, a_{j+1}.. a_{i-1}, a_i.. a_{right}

  Đoạn 1 ■ x

  Đoạn 2: a_{j+1}.. a_{i-1} = x

  Đoạn 3: a_i.. a_{right} ■ x

- <u>Bước 3</u>: **Sắp xếp đoạn 1**: a_{left}.. a_j
- <u>Bước 4</u>: **Sắp xếp đoạn 3**: a_i.. a_{right}

Giải Thuật Quick Sort

- Bước 1 : Chọn tùy ý một phần tử a[k] trong dãy là giá trị mốc ($l \le k \le r$):

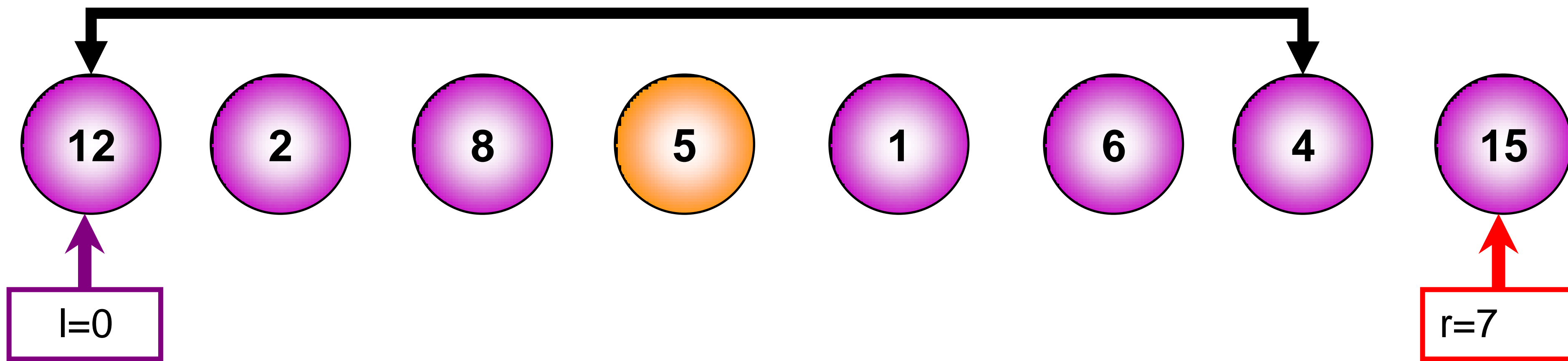
  x = a[k]; i = l; j = r;

- Bước 2 : Phát hiện và hiệu chỉnh cặp phần tử a[i], a[j] nằm sai chỗ :
  - Bước 2a : Trong khi (a[i]<x) i++;
  - Bước 2b : Trong khi (a[j]>x) j--;
  - Bước 2c : Nếu i< j Swap(a[i],a[j]);
- Bước 3 : Nếu i < j: Lặp lại Bước 2.
  Ngược lại: Dừng

Quick Sort – Ví Dụ

- Cho dãy số a:

12 2 8 5 1 6 4 15



Quick Sort – Ví Dụ

4 2 8 5 1 6 12 15

i = 0

l=0

j = 6

r=7

4 2 8 5 1 6 12 15

l=0

i = 1

i = 2

j = 3

j = 4

j = 5

r=7

Quick Sort – Ví Dụ

- Phân hoạch đoạn l = 0, r = 2:

4 2 1 5 8 6 12 15

l = 0

r =3

i = 0

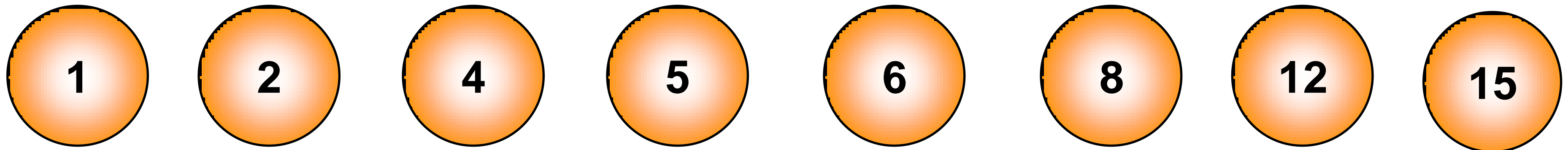
j = 2

Quick Sort – Ví Dụ

- Phân hoạch đoạn l =4, r = 7:

Quick Sort – Ví Dụ

- Phân hoạch đoạn l =6, r = 7:

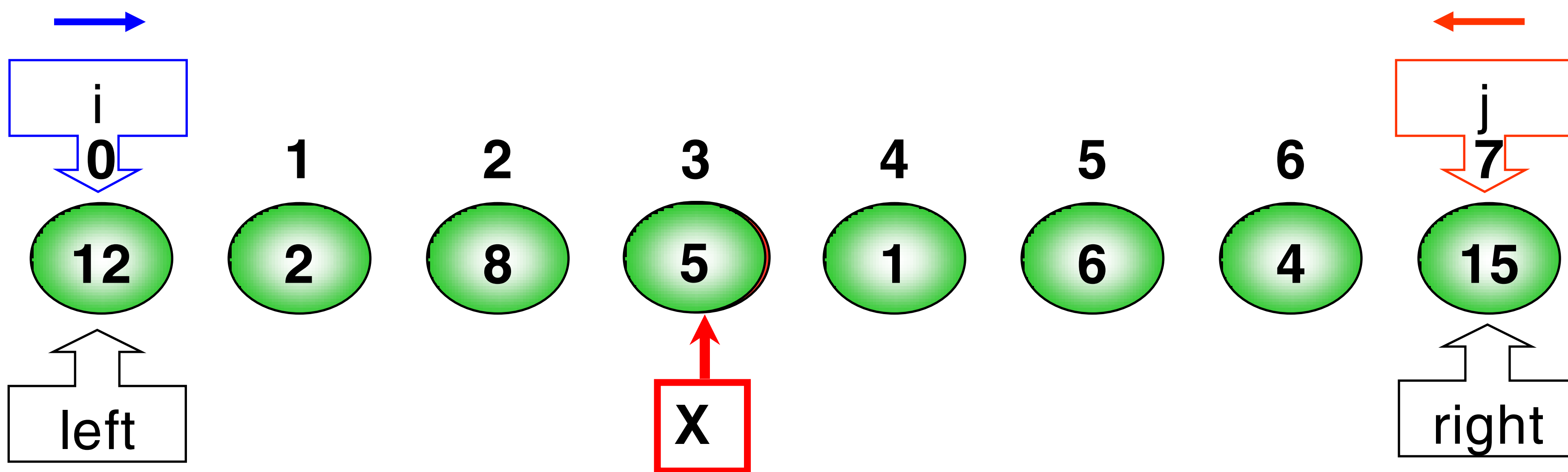


Quick Sort

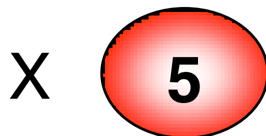
```
void QuickSort(int a[], int left, int right)
{     int  i, j, x;
      x = a[(left+right)/2];
      i = left; j = right;
      do
      {
            while(a[i] < x) i++;
            while(a[j] > x) j--;
            if(i <= j)
            {
                        Swap(a[i],a[j]);
                        i++ ; j--;
            }
      } while(i <= j);

      if(left<j)
            QuickSort(a, left, j);
      if(i<right)
            QuickSort(a, i, right);
}
```

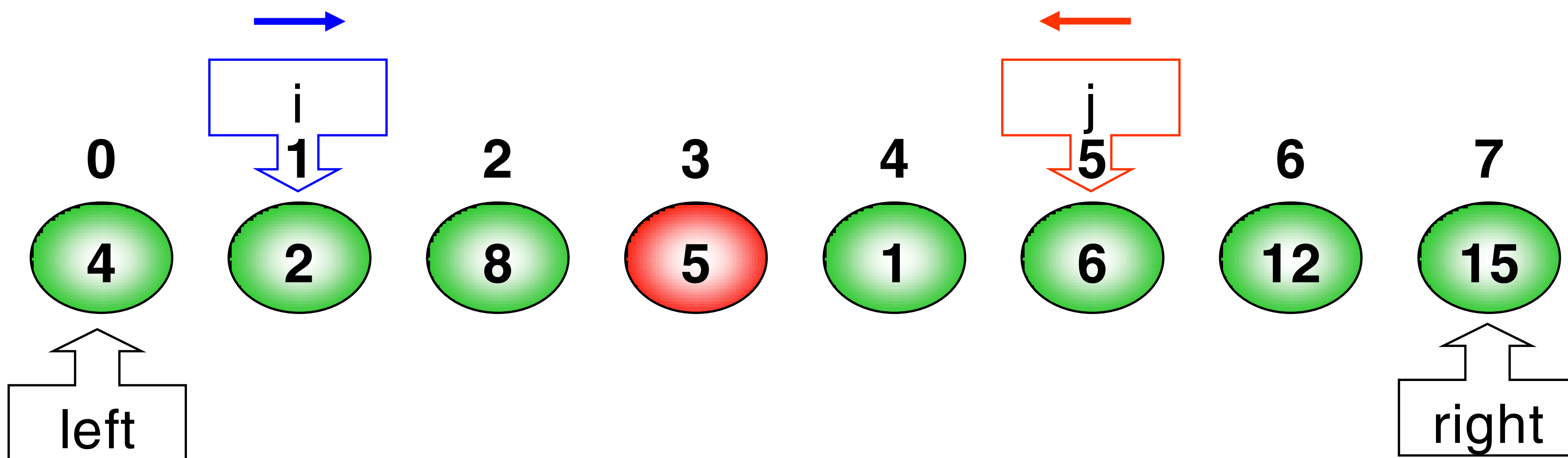
Quick Sort – Ví Dụ

➢ Phân hoạch đọan [0,7]



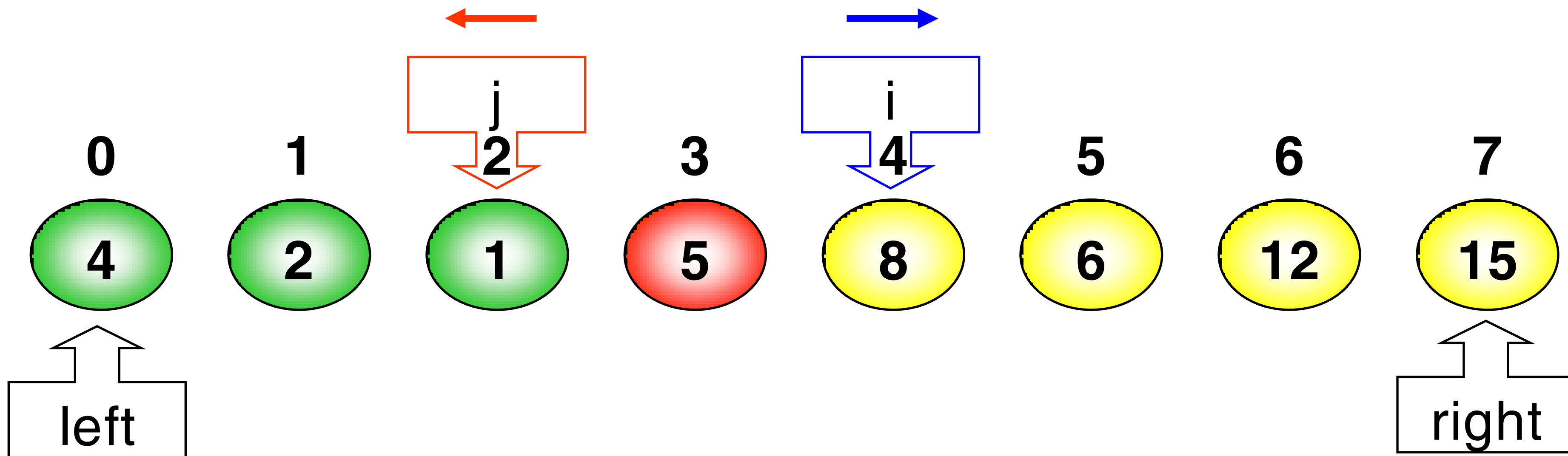
- Phân hoạch đọan [0,7]



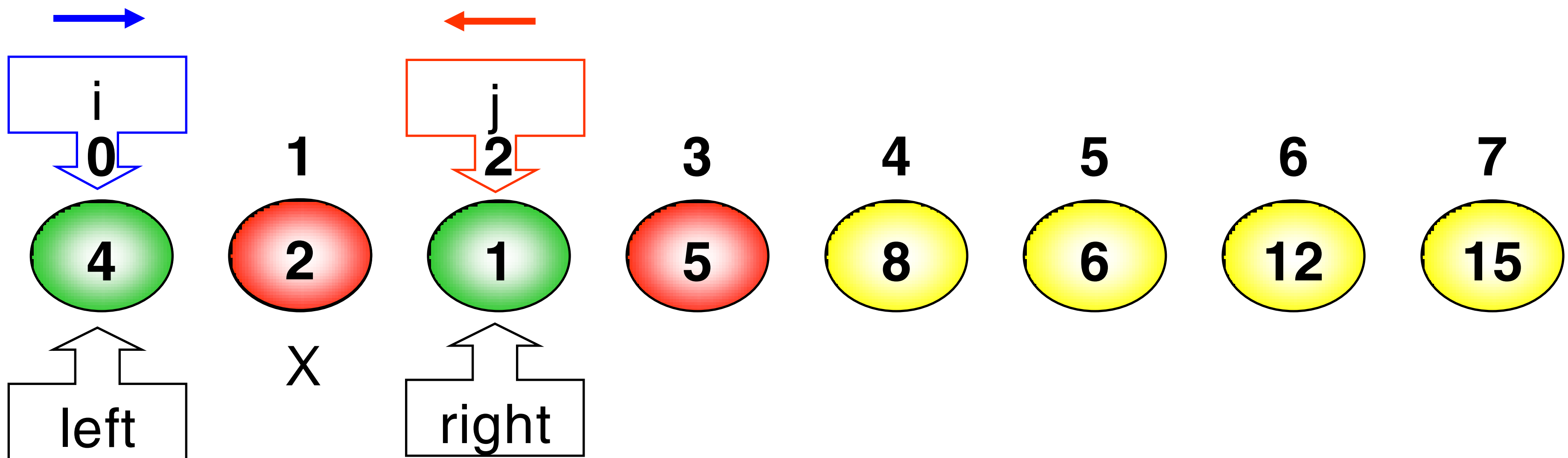


Quick Sort – Ví Dụ

➤ Phân hoạch đọan [0,2]

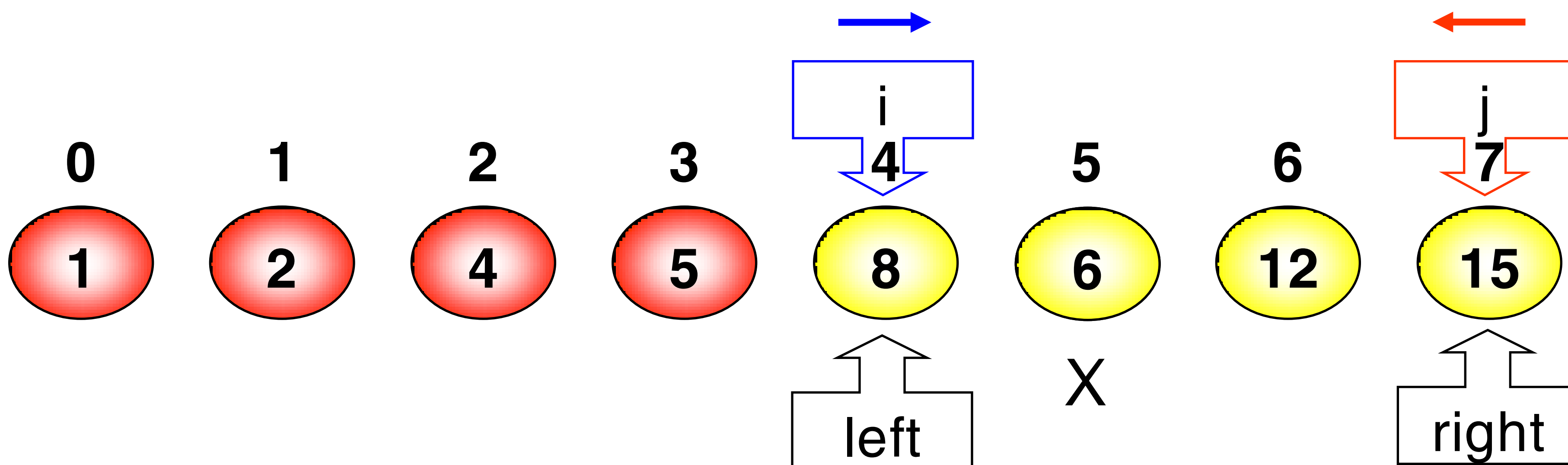


➢ Phân hoạch đọan [0,2]



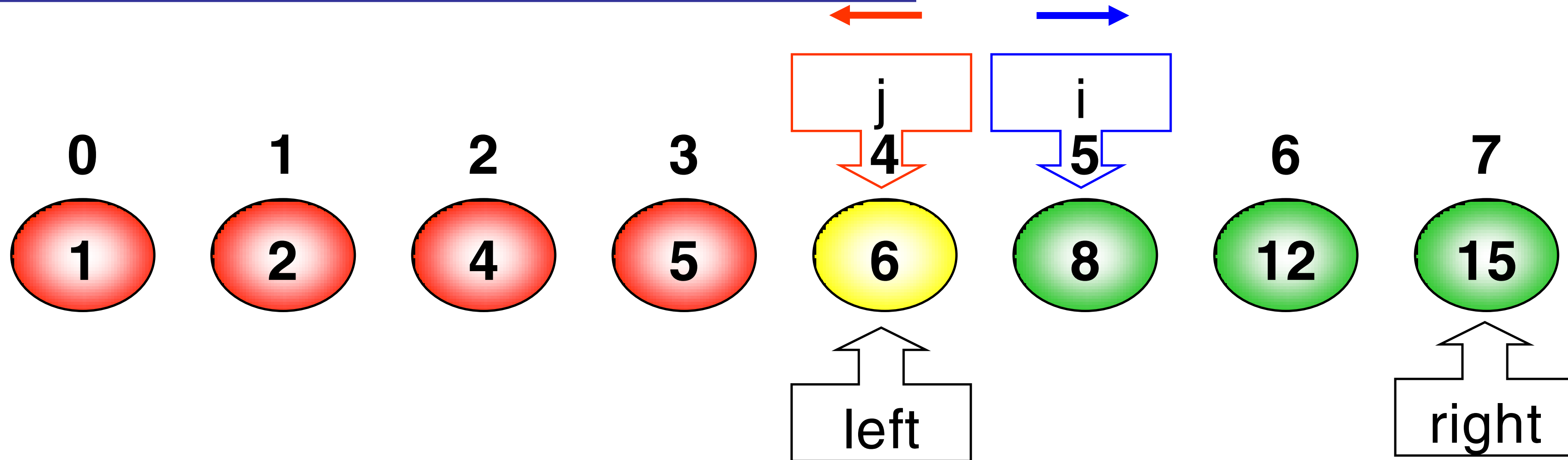
Quick Sort – Ví Dụ

- Phân hoạch đọan [4,7]



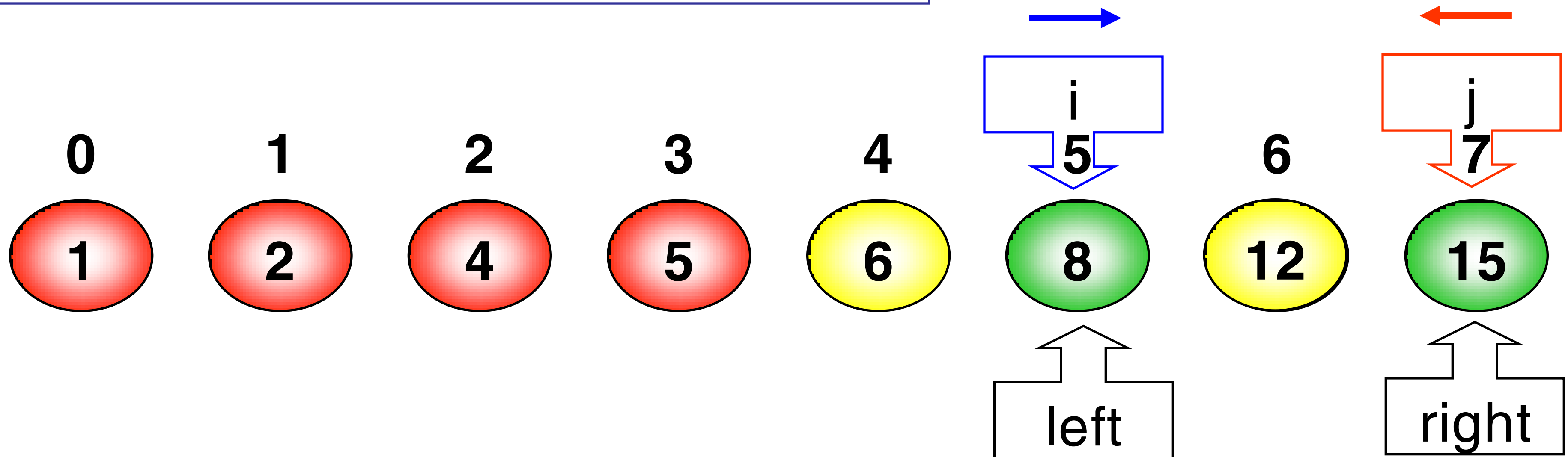
Quick Sort – Ví Dụ

➤ Phân hoạch đọan [5,7]

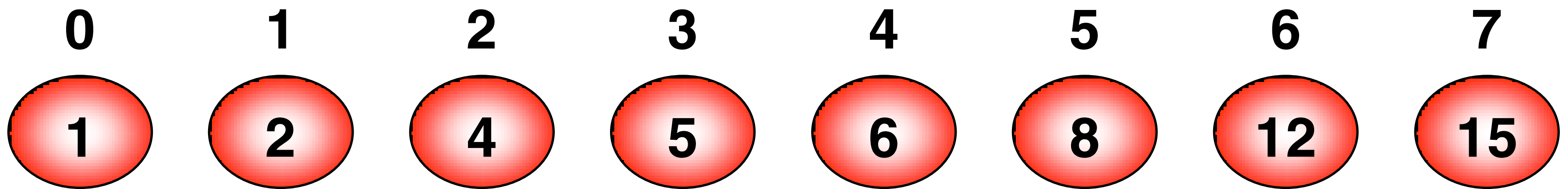


Quick Sort – Ví Dụ

- Phân hoạch đọan [5,7]



Quick Sort – Ví Dụ



Độ Phức Tạp Của Quick Sort

Trường hợp	Độ phức tạp
Tốt nhất	n*log(n)
Trung bình	n*log(n)
Xấu nhất	n^2

Các Thuật Toán Sắp Xếp

1. Đổi chỗ trực tiếp – Interchange Sort
2. Chọn trực tiếp – Selection Sort
3. Nổi bọt – Bubble Sort
4. Shaker Sort
5. Chèn trực tiếp – Insertion Sort
6. Chèn nhị phân – Binary Insertion Sort
7. Shell Sort
8. Heap Sort
9. Quick Sort
10. **Merge Sort**
11. Radix Sort

Merge Sort – Ý Tưởng

- Giải thuật Merge sort sắp xếp dãy $a_1, a_2, ..., a_n$ dựa trên nhận xét sau:
  - Mỗi dãy $a_1, a_2, ..., a_n$ bất kỳ là một tập hợp các dãy con liên tiếp mà mỗi dãy con đều đã có thứ tự.
    - Ví dụ: dãy 12, 2, 8, 5, 1, 6, 4, 15 có thể coi như gồm 5 dãy con không giảm (12); (2, 8); (5); (1, 6); (4, 15).
  - Dãy đã có thứ tự coi như có 1 dãy con.

➔ Hướng tiếp cận: tìm cách làm giảm số dãy con không giảm của dãy ban đầu.

Merge Sort – thuật toán

Bước 1 : // Chuẩn bị

k = 1; // k là chiều dài của dãy con trong bước hiện hành

Bước 2 :

Tách dãy a_0, a_1, ., a_{n-1} thành 2 dãy b, c theo nguyên tắc luân phiên từng nhóm k phần tử:

b = a_0, ., a_k, a_{2k}, ., a_{3k}, .

c = a_{k+1}, ., a_{2k+1}, a_{3k+1}, .

Bước 3 :

Trộn từng cặp dãy con gồm k phần tử của 2 dãy b, c vào a.

Bước 4 :

k = k*2;

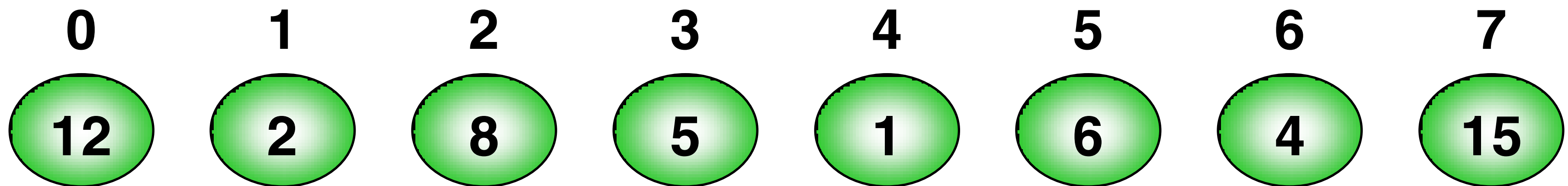
Nếu k < n thì trở lại bước 2.

Ngược lại: Dừng

Merge Sort – Ví Dụ

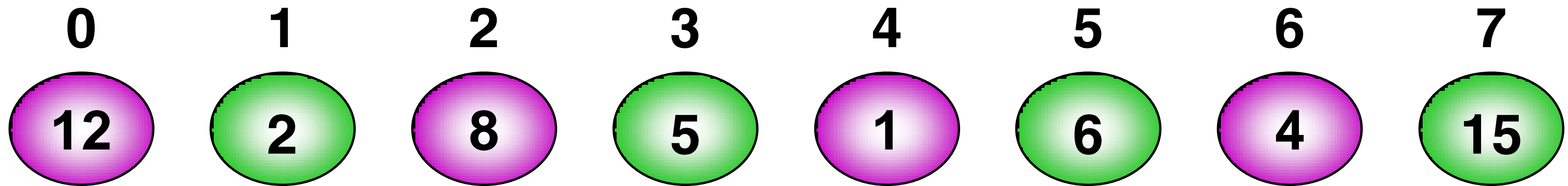
- k=1

- Phân phối luân phiên



Merge Sort – Ví Dụ

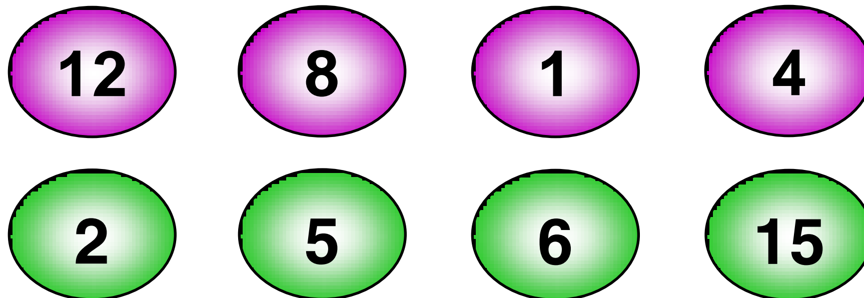
- k = 1
- Phân phối luân phiên



Merge Sort – Ví Dụ

➢Trộn từng cặp đường chạy

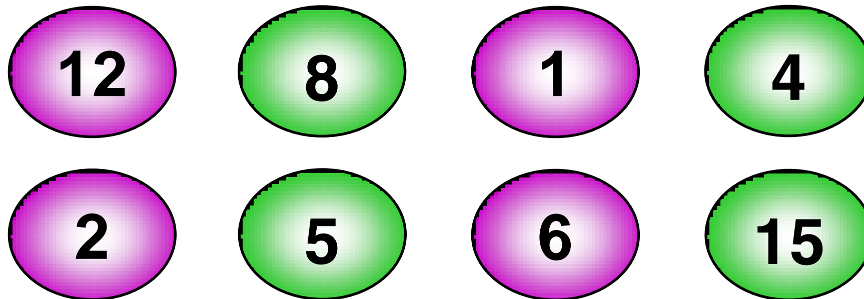
0	1	2	3	4	5	6	7



Merge Sort – Ví Dụ

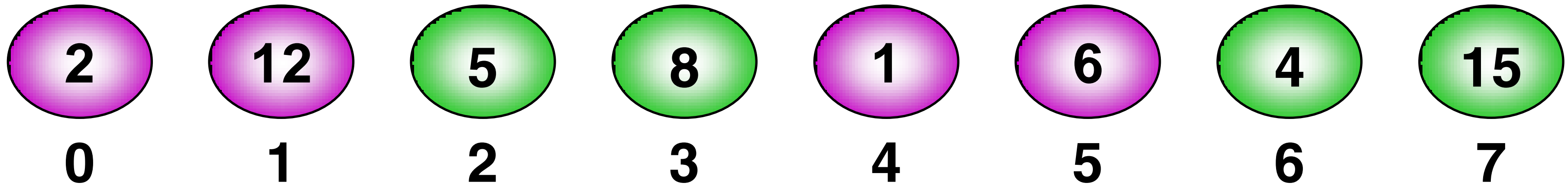
- k = 1
- Trộn từng cặp đường chạy

0	1	2	3	4	5	6	7



Merge Sort – Ví Dụ

- k = 2
- Phân phối luân phiên

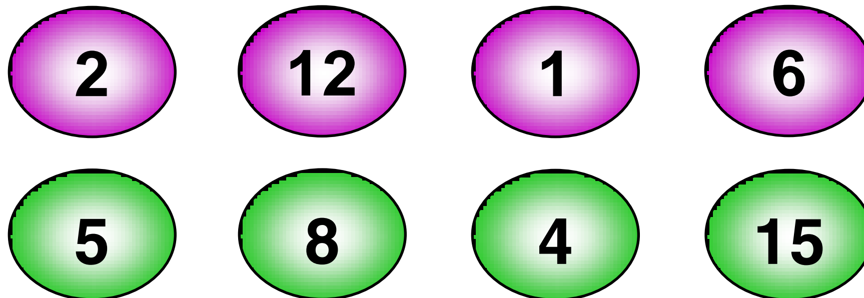


Merge Sort – Ví Dụ

➤k = 2

➤Trộn từng cặp đường chạy

0	1	2	3	4	5	6	7

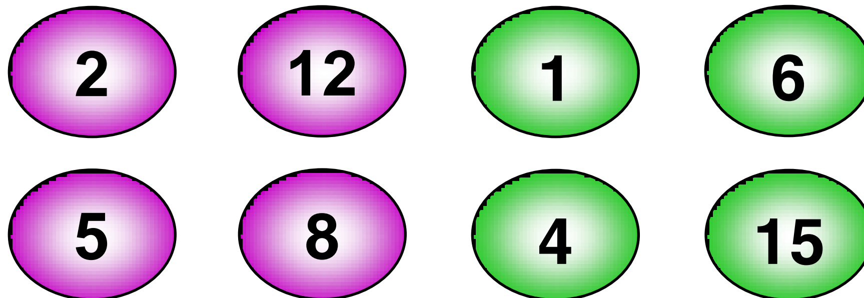


Merge Sort – Ví Dụ

➢k = 2

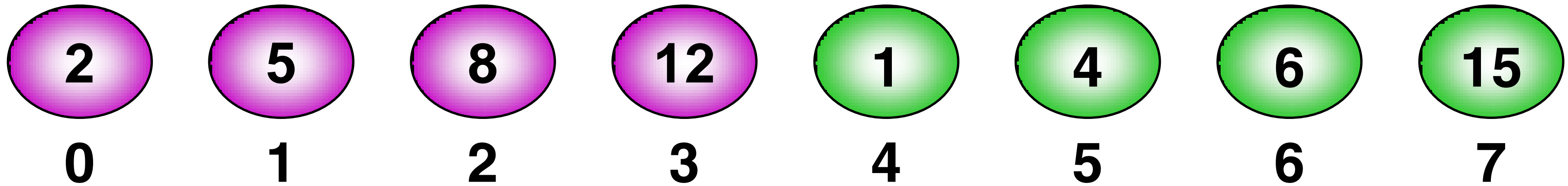
➢Trộn từng cặp đường chạy

0	1	2	3	4	5	6	7



Merge Sort – Ví Dụ

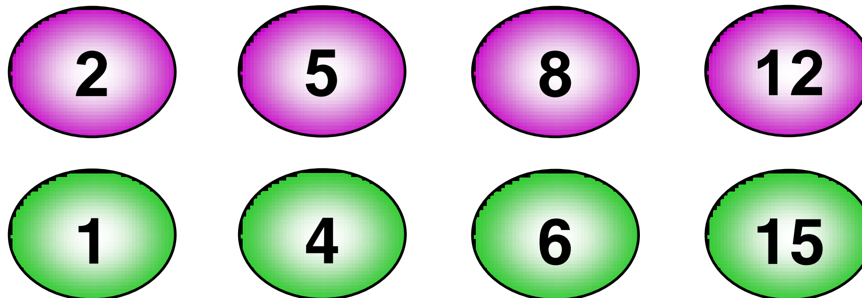
- k = 4
- Phân phối luân phiên



Merge Sort – Ví Dụ

- k = 4
- Trộn từng cặp đường chạy

0	1	2	3	4	5	6	7

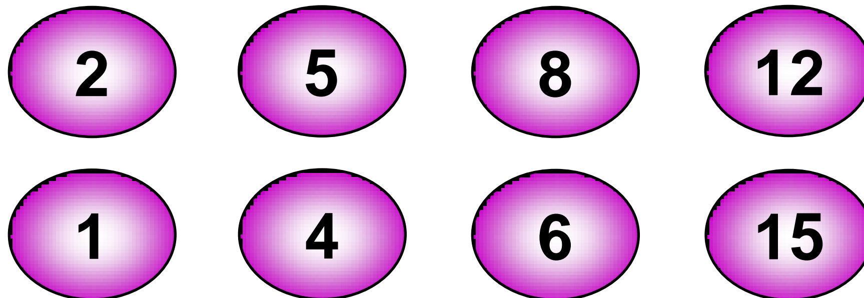


Merge Sort – Ví Dụ

➢k = 4

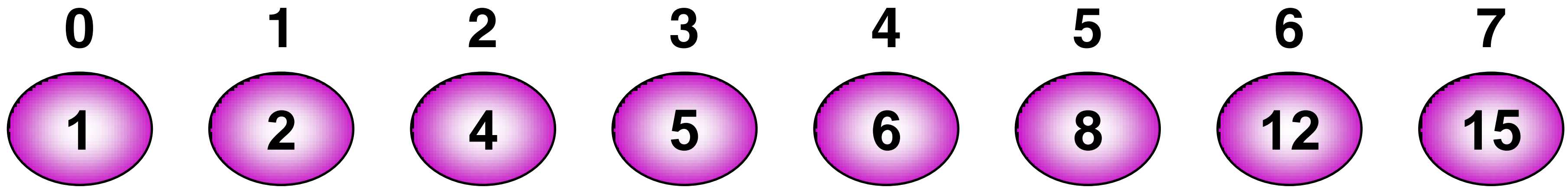
➢Trộn từng cặp đường chạy

0	1	2	3	4	5	6	7

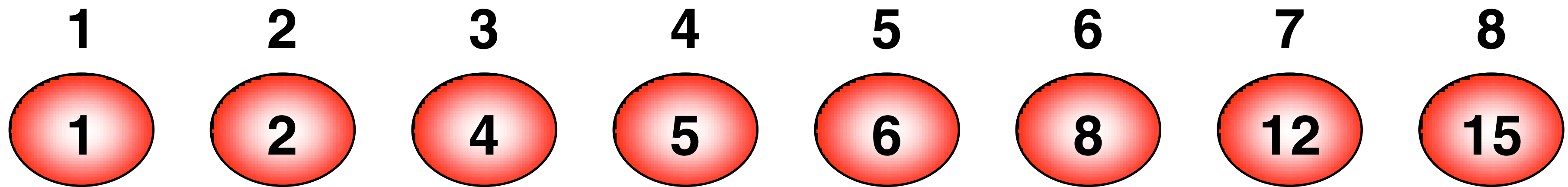


Merge Sort – Ví Dụ

➢k = 8



Merge Sort – Ví Dụ



Merge Sort – Cài Đặt

- Dữ liệu hỗ trợ: 2 mảng b, c:

```
int  b[MAX], c[MAX], nb, nc;
```

- Các hàm cần cài đặt:
  - **void MergeSort(int a[], int N);** : Sắp xếp mảng (a, N) tăng dần
  - **void Distribute(int a[], int N, int &nb, int &nc, int k);** Phân phối đều luân phiên các dãy con độ dài k từ mảng a vào hai mảng con b và c
  - **void Merge(int a[], int nb, int nc, int k);** : Trộn mảng b và mảng c vào mảng a
  - **void MergeSubarr(int a[], int nb, int nc, int &pa, int &pb, int &pc, int k);** : Trộn một cặp dãy con từ b và c vào a

Merge Sort – Cài Đặt

```
int b[MAX], c[MAX], nb, nc;

void MergeSort(int a[], int N)
{
	int  k;
	for (k = 1; k < N; k *= 2)
	{
		Distribute(a, N, nb, nc, k);
		Merge(a, nb, nc, k);
	}
}
```

Merge Sort – Cài Đặt

```
void Distribute(int a[], int N, int &nb, int &nc, int k)
{
	int   i, pa, pb, pc;
	pa = pb = pc = 0;
	while (pa < N)
	{
		for (i=0; (pa<N) && (i<k); i++, pa++, pb++)
				b[pb] = a[pa];
		for (i=0; (pa<N) && (i<k); i++, pa++, pc++)
				c[pc] = a[pa];
	}
	nb = pb;	nc = pc;
}
```

Merge Sort – Cài Đặt

```
void Merge(int a[],int nb, int nc,int k)
{   int p, pb, pc, ib, ic, kb, kc;
    p=pb=pc=0; ib=ic=0;
    while((nb>0)&&(nc>0))
    {   kb=min(k,nb); kc=min(k,nc);
        if(b[pb+ib]<=c[pc+ic])
        {       a[p++]=b[pb+ib]; ib++;
                if(ib==kb)
                {       for(;ic<kc;ic++     a[p++]=c[pc+ic];
                        pb+=kb; pc+=kc; ib = ic=0;
                        nb-=kb; nc-=kc;
                }
        }
```

Merge Sort – Cài Đặt

```
        else
        {       a[p++]=c[pc+ic]; ic++;
                if(ic==kc)
                {
                        for(;ib<kb;ib++)    a[p++]=b[pb+ib];
                        pb+=kb;  pc+=kc; ib = ic=0;
                        nb-=kb; nc-=kc;
                }
        }
    }
}
```

Merge Sort – Cài Đặt

```
int min(int a,int b)
{
    if(a>b) return b;
    else        return a;
}
```

Độ phức tạp của Merge Sort

- Số lần lặp của Bước 2, 3 là $\log_2 n$ do sau mỗi lần lặp giá trị k tăng gấp đôi. Chi phí thực hiện ở bước 2 và 3 tỉ lệ thuật với n. Do dó chi phí của dãy thuật MergeSort là $O(n\log_2 n)$

Các Thuật Toán Sắp Xếp

1. Đổi chỗ trực tiếp – Interchange Sort
2. Nổi bọt – Bubble Sort
3. Shaker Sort
4. Chèn trực tiếp – Insertion Sort
5. Chèn nhị phân – Binary Insertion Sort
6. Shell Sort
7. Chọn trực tiếp – Selection Sort
8. Quick Sort
9. Merge Sort
10. Heap Sort
11. **Radix Sort**

Sắp Xếp Theo Phương Pháp Cơ Số Radix Sort

- Radix Sort là một thuật toán tiếp cận theo một hướng hoàn toàn khác.
- Nếu như trong các thuật toán khác, cơ sở để sắp xếp luôn là việc so sánh giá trị của 2 phần tử thì Radix Sort lại dựa trên nguyên tắc phân loại thư của bưu điện. Vì lý do đó Radix Sort còn có tên là Postman's Sort.
- Radix Sort không hề quan tâm đến việc so sánh giá trị của phần tử mà bản thân việc phân loại và trình tự phân loại sẽ tạo ra thứ tự cho các phần tử.

- Mô phỏng lại qui trình trên, để sắp xếp dãy a_1, a_2, ..., a_n, giải thuật Radix Sort thực hiện như sau:
  - Trước tiên, ta có thể giả sử mỗi phần tử a_i trong dãy a_1, a_2, ..., a_n là một số nguyên có tối đa m chữ số.
  - Ta phân loại các phần tử lần lượt theo các chữ số hàng đơn vị, hàng chục, hàng trăm, … tương tự việc phân loại thư theo tỉnh thành, quận huyện, phường xã, ….

Sắp Xếp Theo Phương Pháp Cơ Số Radix Sort

- Bước 1 :// k cho biết chữ số dùng để phân loại hiện hành
  - k = 0; // k = 0: hàng đơn vị; k = 1: hàng chục; …
- Bước 2 : //Tạo các lô chứa các loại phần tử khác nhau
  - Khởi tạo 10 lô B_0, B_1, …, B_9 rỗng;

Sắp Xếp Theo Phương Pháp Cơ Số Radix Sort

- Bước 3 :
  - For i = 1 .. n do
    - Đặt a_i vào lô B_t với t: chữ số thứ k của a_i;
- Bước 4 :
  - Nối B_0, B_1, …, B_9 lại (theo đúng trình tự) thành a.
- Bước 5 :
  - k = k+1;Nếu k < m thì trở lại bước 2. Ngược lại: Dừng

Sắp Xếp Theo Phương Pháp Cơ Số Radix Sort

CS	A	0	1	2	3	4	5	6	7	8	9
12	0701										
11	1725										
10	0999										
9	9170										
8	3252										
7	4518										
6	7009										
5	1424										
4	0428										
3	1239										0999
2	8425						1725			4518	7009
1	7013	9170	0701	3252	7013	1424	8425			0428	1239

Sắp Xếp Theo Phương Pháp Cơ Số Radix Sort

CS	A	0	1	2	3	4	5	6	7	8	9
12	0999										
11	7009										
10	1239										
9	4518										
8	0428										
7	1725										
6	8425										
5	1424										
4	7013			0428							
3	3252			1725							
2	0701	7009	4518	8425							
1	9170	0701	7013	1424	1239		3252		9170		0999

Sắp Xếp Theo Phương Pháp Cơ Số Radix Sort

CS	A	0	1	2	3	4	5	6	7	8	9
12	0999										
11	9170										
10	3252										
9	1239										
8	0428										
7	1725										
6	8425										
5	1424										
4	4518										
3	7013					0428					
2	7009	7013		3252		8425			1725		
1	0701	7009	9170	1239		1424	4518		0701		0999

Sắp Xếp Theo Phương Pháp Cơ Số Radix Sort

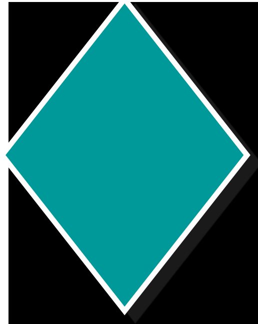
CS	A	0	1	2	3	4	5	6	7	8	9
12	0999										
11	1725										
10	0701										
9	4518										
8	0428										
7	8425										
6	1424										
5	3252										
4	1239										
3	9170	0999	1725								
2	7013	0701	1424						7013		
1	7009	0428	1239		3252	4518			7009	8425	9170

Sắp Xếp Theo Phương Pháp Cơ Số Radix Sort

CS	A	0	1	2	3	4	5	6	7	8	9
12	9170										
11	8425										
10	7013										
9	7009										
8	4518										
7	3252										
6	1725										
5	1424										
4	1239										
3	0999										
2	0701										
1	0428										

Bài Tập

- Nhập một dãy số nguyên n phần tử.
- Sắp xếp lại dãy sao cho:
  - số nguyên dương đầu ở đầu dãy và theo thứ tự giảm.
  - số nguyên âm tăng ở cuối dãy và theo thứ tự tăng.
  - số 0 ở giữa.
- *Lưu ý: Không dùng đổi chỗ trực tiếp.*



CẤU TRÚC DỮ LIỆU ĐỘNG

Biến Tĩnh

- Được khai báo tường minh, có tên gọi
- Tồn tại trong phạm vi khai báo
- Được cấp phát trong stack
- Kích thước không đổi => không tận dụng hiệu quả bộ nhớ
- Ví dụ : int x,y;

```
int     x,y;
char    c;
float   f[5];
```

- Khi biết chắc nhu cầu sử dụng đối tượng trước khi thực sự xử lý : dùng biến không động

Ví Dụ Hạn Chế Của Biến Tinh

- Tổ chức danh sách lớp học
- Dùng mảng tĩnh :

```
typedef        struct
{
  char ten[20];
   int   maso;
}Hocvien;

Hocvien        danhsach[50];
```

- Số lượng học viên <50 => lãng phí
- Số lượng học viện > 50 => thiếu chỗ !

Biến Động

- Không được khai báo tường minh, không có tên gọi
- Xin khi cần, giải phóng khi sử dụng xong
- Được cấp phát trong heap
- Linh động về kích thước
- Vấn đề : biến động không có tên gọi tường minh, làm sao thao tác ?

Kiểu con trỏ

- Kiểu con trỏ dùng lưu địa chỉ của một đối tượng dữ liệu khác.
- Biến thuộc kiểu con trỏ Tp là biến mà giá trị của nó là địa chỉ cuả một vùng nhớ ứng với một biến kiểu T, hoặc là giá trị NULL.
- Khai báo trong C :

```
typedef     int      *intpointer;
intpointer           p;
```

- Bản thân biến con trỏ là không động
- Dùng biến con trỏ để lưu giữ địa chỉ của biến động => truy xuất biến động thông qua biến con trỏ

Các thao tác trên kiểu con trỏ

- Tạo ra một biến động và cho con trỏ ‘p’ chỉ đến nó:
  - void* malloc(size);
  - void* calloc(n,size);
  - new // hàm cấp phát bộ nhớ trong C++
- Hủy một biến động do p chỉ đến :
  - Hàm free(p) huỷ vùng nhớ cấp phát bởi hàm malloc hoặc calloc do p trỏ tới
  - Hàm delete p huỷ vùng nhớ cấp phát bởi hàm new do p trỏ tới

Sử dụng biến tĩnh, con trỏ và biến động

```
int   x;
x = 5 ;
```

Biến không động x

5

```
int *p;

p =
new(int);
*p = 5
```

Biến con trỏ p

0xFF

0xFF

5

Biến động có địa chỉ 0xFF

Kiểu danh sách

- Danh sách = { các phần tử có cùng kiểu}
- Danh sách là một kiểu dữ liệu tuyến tính :
  - Mỗi phần tử có nhiều nhất 1 phần tử đứng trước
  - Mỗi phần tử có nhiều nhất 1 phần tử đứng sau
- Là kiểu dữ liệu quen thuộc trong thực tế :
  - Danh sách học sinh
  - Danh mục sách trong thư viện
  - Danh bạ điện thoại
  - Danh sách các nhân viên trong công ty
  - …

Các hình thức tổ chức danh sách

- CTDL cho mỗi phần tử ?
- Thể hiện liên kết của các phần tử ?
- Hai hình thức cơ bản :
  - Liên kết ngầm : Mảng
  - Liên kết tường minh : Danh sách liên kết

Danh sách liên kết ngầm(mảng)

- Mối liên hệ giữa các phần tử được thể hiện ngầm:
  - xi : phần tử thứ i trong danh sách
  - xi , xi_{+1} là kế cận trong danh sách

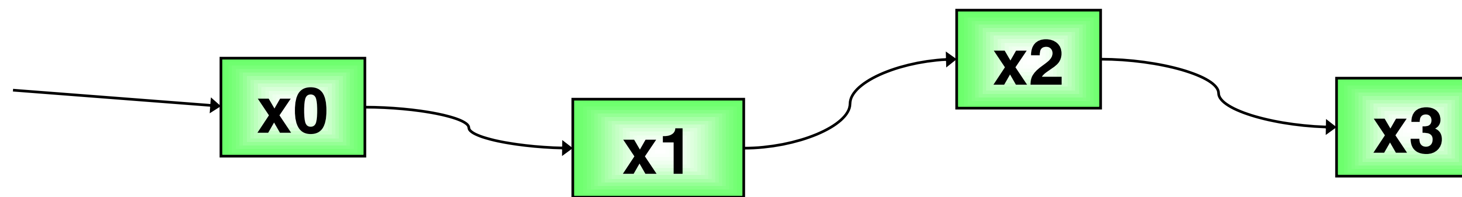
x0	...	xi	xi+1

- Phải lưu trữ liên tiếp các phần tử trong bộ nhớ
  - công thức xác định địa chỉ phần tử thứ i:

    address(i) = address(1) + (i-1)*sizeof(T)
- **Ưu điểm** : Truy xuất trực tiếp, nhanh chóng
- **Nhược điểm**:
  - Sử dụng bộ nhớ kém hiệu quả
  - Kích thước cố định
  - Các thao tác thêm vào , loại bỏ không hiệu quả

Liên kết tường minh(Danh sánh liên kết)

- CTDL cho một phần tử
  - Thông tin bản thân
  - Địa chỉ của phần tử kế trong danh sách



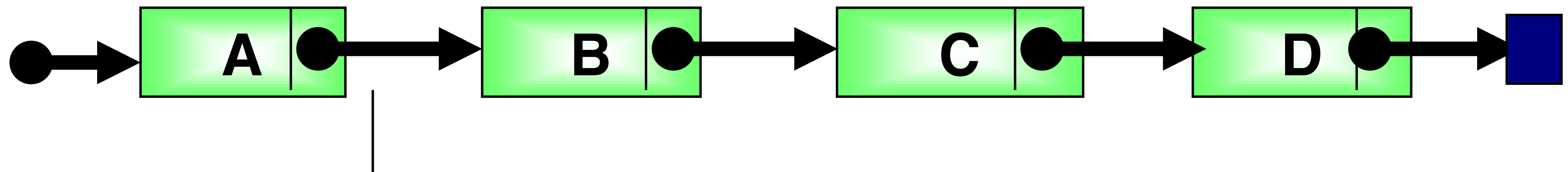
- Mỗi phần tử là một biến động
- **Ưu điểm**

  + Sử dụng hiệu quả bộ nhớ

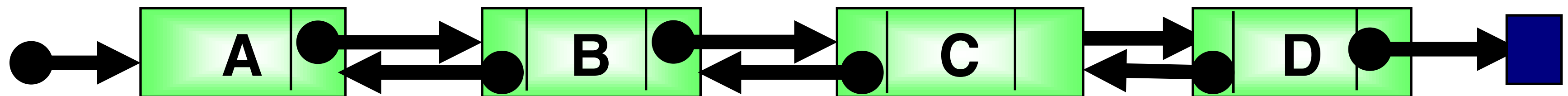
  + Linh động về số lượng phần tử

Các loại danh sách liên kết

➢ **Danh sách liên kết đơn**: Mỗi phần tử liên kết với phần tử đứng sau nó trong danh sách



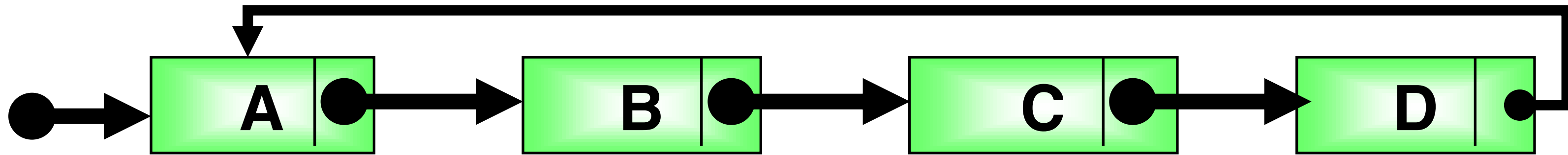
➢ **Danh sách liên kết kép**: Mỗi phần tử liên kết với phần tử đứng trước và sau nó trong danh sách



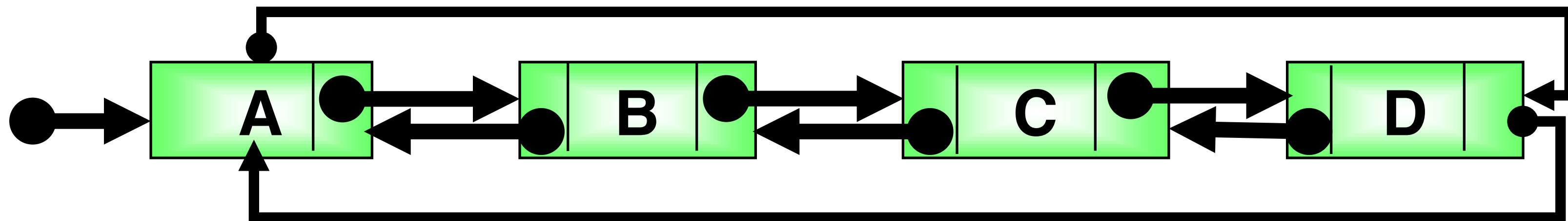
➢ **Danh sách liên Vòng**: Phần tử cuối danh sách liên với phần tử đầu danh sách

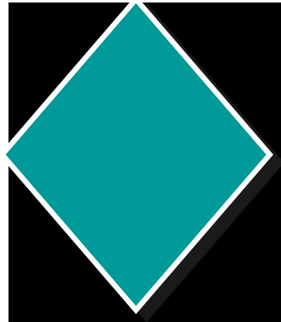
Các loại danh sách liên kết (tt)

- **<u>Danh sách liên Vòng</u>:** Phần tử cuối danh sách liên với phần tử đầu danh sách
  - Danh sách liên kết đơn vòng



  - Danh sách liên kết đôi vòng

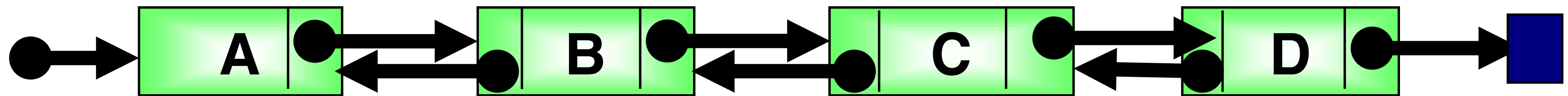




DANH SÁCH LIÊN KẾT kép

Định Nghĩa

- Mỗi phần tử liên kết với phần tử đứng trước và sau nó trong danh sách
- Hình vẽ minh họa danh sách liên kết kép:



Cấu Trúc Dữ Liệu

- *Cấu trúc dữ liệu 1 nút*

```
typedef struct tagDnode
{   Data Info;
    struct tagDnode *pPre;
    struct tagDnode *pNext;
}DNode;
```

- *Cấu trúc List kép*

```
Typedef struct tagDList
{   DNode *pHead;
    DNode *pTail;
}DList;
```

Các Thao Tác Trên List Kép

- Khởi tạo danh sách liên kết kép rỗng
- Tạo 1 nút có thành phần dữ liệu = x
- Chèn 1 phần tử vào danh sách
  - Chèn vào đầu
  - Chèn sau phần tử Q
  - Chèn vào trước phần tử Q
  - Chèn vào cuối danh sách
- Huỷ 1 phần tử trong danh sách
  - Hủy phần tử đầu danh sách
  - Hủy phần tử cuối danh sách
  - Hủy 1 phần tử có khoá bằng x
- Tìm 1 phần tử trong danh sách
- Sắp xếp danh sách

Tạo 1 Danh Sách Rỗng

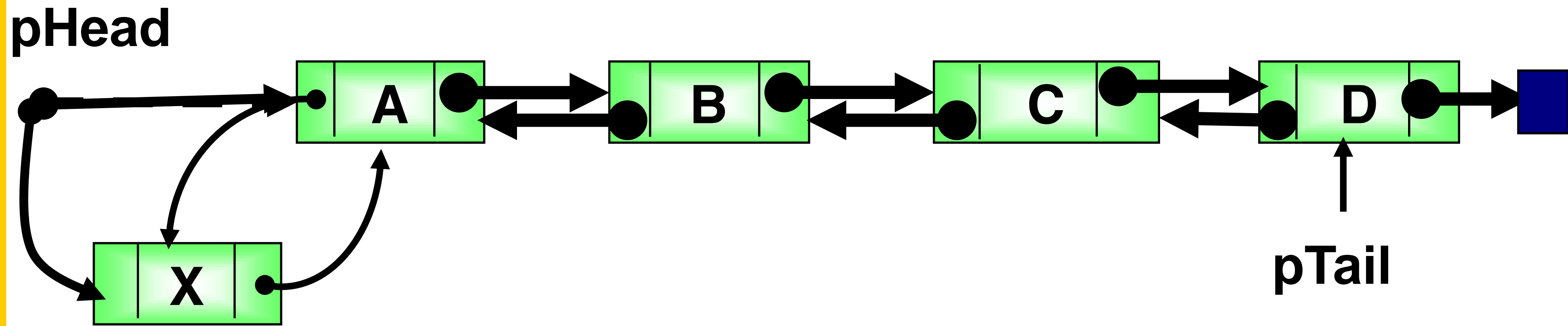
```
void CreateDList(DList &l)
{
        l.DHead=NULL;
        l.DTail=NULL;
}
```

Tạo 1 Nút Có Thành Phần Dữ Liệu = X

```
DNode *CreateDNode(int x)
{   DNode *tam;
    tam=new DNode;
    if(tam==NULL)
    {     printf("khong con du bo nho");
          exit(1);
    }
    else
    {     tam->Info=x;
          tam->pNext=NULL;
          tam->pPre=NULL;
          return tam;
    }
}
```

Thêm 1 Nút Vào Đầu Danh Sách

- Minh họa hình vẽ

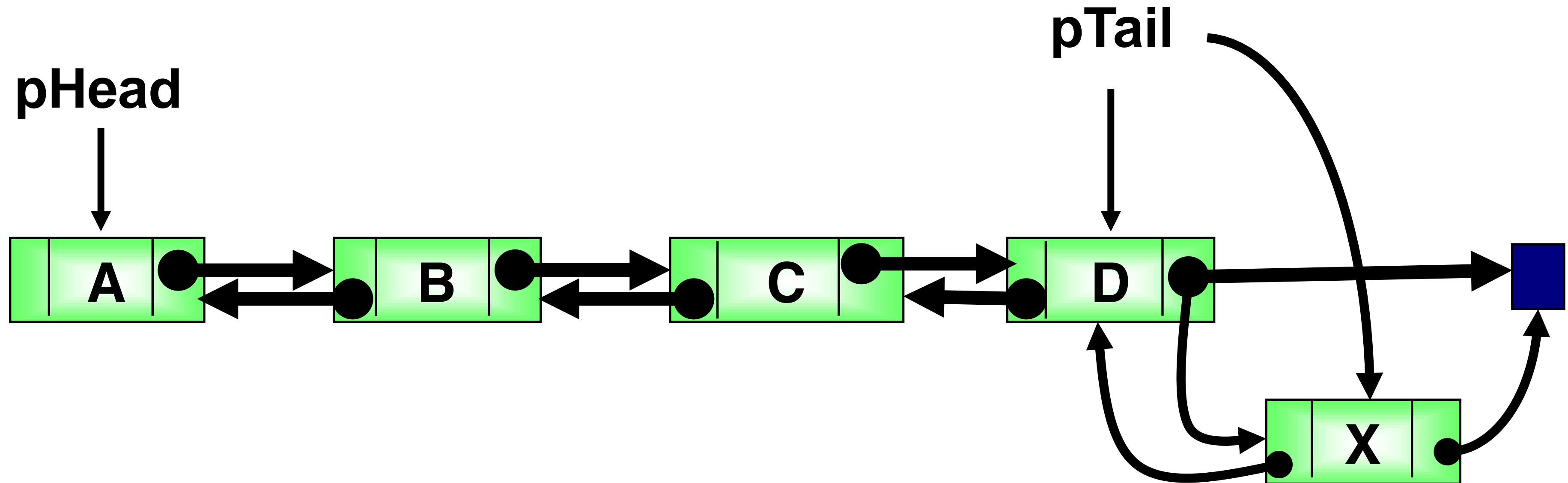


Cài Đặt Thêm 1 Nút Vào Đầu Danh Sách

```
void AddFirst(DList &l, DNode *tam)
{
    if(l.pHead==NULL)//xau rong
    {
        l.pHead=tam;
        l.pTail=l.pHead;
    }
    else
    {
        tam->pNext=l.pHead;
        l.pHead->pPre=tam;
        l.pHead=tam;
    }
}
```

Thêm Vào Cuối Danh Sách

- Minh họa thêm 1 phần tử vào sau danh sách



Cài Đặt Thêm 1 Nút Vào Cuối Danh Sách


```
void AddEnd(DList &l,DNode *tam)
{
    if(l.pHead==NULL)
    {
        l.pHead=tam;
        l.pTail=l.pHead;
    }
    else
    {
        tam->pPre=l.pTail;
        l.pTail->pNext=tam;
        tam=l.pTail;
    }
}
```

Thêm Vào Sau Nút Q

- Minh họa thêm nút X vào sau nút q

pHead
q
pTail
A
B
C
D
X

Cài Đặt Thêm 1 Nút Vào Sau Nút Q

```
void AddLastQ(DList &l,DNode *tam, DNode *q)
{
    DNode *p;
    p=q->pNext;
    if(q!=NULL)//them vao duoc
    {
        tam->pNext=p;
        tam->pPre=q;
        q->pNext=tam;
        if(p!=NULL)
            p->pPre=tam;
        if(q==l.pTail) //them vao sau danh sach lien ket.
            l.pTail=tam;
    }
    else
        AddFirst(l,tam);
}
```

Thêm 1 Nút Vào Trước Nút Q

- Minh hoạ thêm 1 nút vào trước nút q

pHead
q
pTail
A
B
C
D
X

Cài Đặt Thêm 1 Nút Vào Trước Nút Q

```
void AddBeforeQ(DList &l,DNode *tam,DNode *q)
{   DNode *p;
    p=q->pPre;
    if(q!=NULL)
    {
            tam->pNext=q;
            q->pPre=tam;
            tam->pPre=p;
            if(p!=NULL)
                    p->pNext=tam;
            if(q==l.pHead)
                    l.pHead = tam;
    }
    else
            AddEnd(l,tam);
}
```

Xoá Phần Tử Đầu Danh Sách

```
void DeleteFirst(DList &l)
{
   DNode *p;
   if(l.pHead!=NULL)
   {
      p=l.pHead;
      l.pHead=l.pHead->pNext;
      l.pHead->pPre=NULL;
      delete p;
      if(l.pHead==NULL)
         l.pTail=NULL;
   }
}
```

Xoá 1 Phần Tử Cuối Danh Sách

```
void DeleteEnd(DList &l )
{
    DNode *p;
    if(l.pHead!=NULL) //tuc xau co hon mot phan tu
    {
        p=l.pTail;
        l.pTail=l.pTail->Pre;
        l.pTail->pNext=NULL;
        delete p;
        if(l.pTail==NULL)
            l.pHead=NULL;
    }
}
```

Hủy 1 Nút Sau Nút Q

```
void DeleteLastQ(DList &l,DNode *q)
{
    DNode *p;//luu node dung sau node q
    if(q!=NULL)
    {
        p=q->pNext;
        if(p!=NULL)
        {
            q->pNext=p->pNext;
            if(p==l.pTail)//xoa dung nu't cuoi
                l.pTail=q;
            else    //Nut xoa khong phai nut cuoi
                p->pNext->pPre=q;
            delete p;
        }
    }
    else
        DeleteFirst(l);
}
```

Huỷ 1 Nút Đứng Trước Nút Q

```
void DeleteBeforeQ(DList &l,DNode *q)
{
    DNode *p;
    if(q!=NULL) //tuc ton tai node q
    {
        p=q->pPre;
        if(p!=NULL)
        {
            q->pPre=p->pPre;
            if(p==l.pHead)//p la Node dau cua danh sach
                l.pHead=q;
            else //p khong phai la node dau
                p->pPre->pNext=q;
            delete p;
        }
    }
    else
        DeleteEnd(l);
}
```

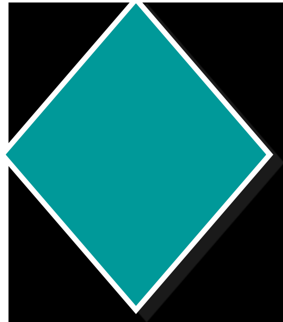
Xoá 1 Phần Tử Có Khoá = X

```
int DeleteX(DList &l,int x)
{
    DNode *p;
    DNode *q;
    q=NULL;
    p=l.pHead;
    while(p!=NULL)
    {
        if(p->Info==x)
            break;
        q=p;//q la Node co truong Info = x
        p=p->pNext;
    }
    if(q==NULL) return 0;//khong tim thay Node nao co truong Info =x
    if(q!=NULL)
        DeleteLastQ(l,q);
    else
        DeleteFirst(l);
    return 1;
}
```

Sắp Xếp

```
void DoiChoTrucTiep(DList &l)
{   DNode *p,*q;
    p=l.pHead;
    while(p!=l.pTail)
    {
        q=p->pNext;
        while(q!=NULL)
        {
            if(p->Info>q->Info)
                HV(p,q);
            q=q->pNext;
        }
        p=p->pNext;
    }}
```

NỘI DUNG



CÂY VÀ CÂY NHỊ PHÂN

Định Nghĩa Cây

- Cây là một tập hợp T các phần tử (gọi là nút của cây), trong đó có một nút đặc biệt gọi là nút gốc, các nút còn lại được chia thành những tập rời nhau T_1, T_2, …,T_n theo quan hệ phân cấp, trong đó T_i cũng là 1 cây. Mỗi nút ở cấp i sẽ quản lý một số nút ở cấp i+1. Quan hệ này người ta gọi là quan hệ cha – con.

Một Số Khái Niệm

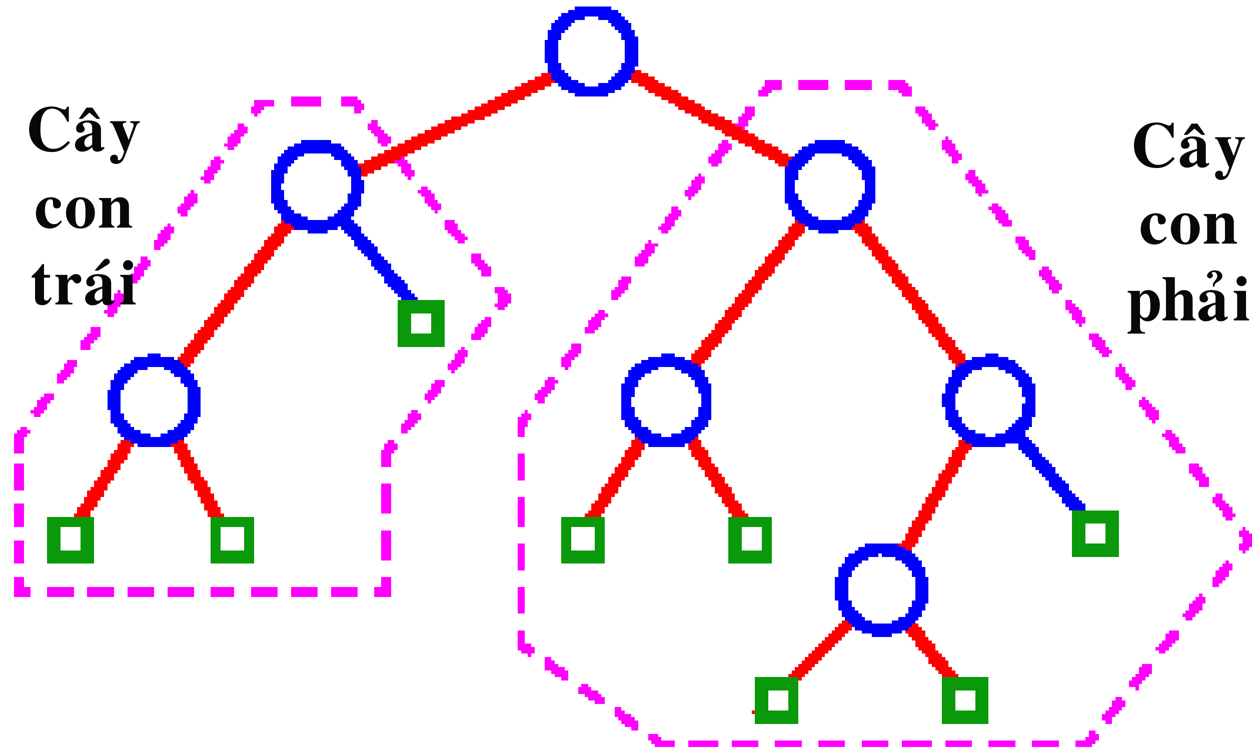
- Bậc của một nút: là số cây con của nút đó .
- Bậc của một cây: là bậc lớn nhất của các nút trong cây
- Nút gốc: là nút không có nút cha.
- Nút lá: là nút có bậc bằng 0 .
- Mức của một nút:
  - Mức (gốc (T)) = 0.
  - Gọi T1, T2, T3, ... , Tn là các cây con của T0 : Mức (T1) = Mức (T2) = . . . = Mức (Tn) = Mức (T0) + 1.
- Độ dài đường đi từ gốc đến nút x: là số nhánh cần đi qua kể từ gốc đến x.

Ví Dụ 1 Tổ Chức Dạng Cây

- BB-Electronic Corp.
  - R&D
  - Kinh doanh
    - Nội địa
    - Quốc tế
      - Châu âu
      - Mỹ
      - Các nước
  - Tài vụ
  - Sản xuất
    - TV
    - CD
    - Amplier

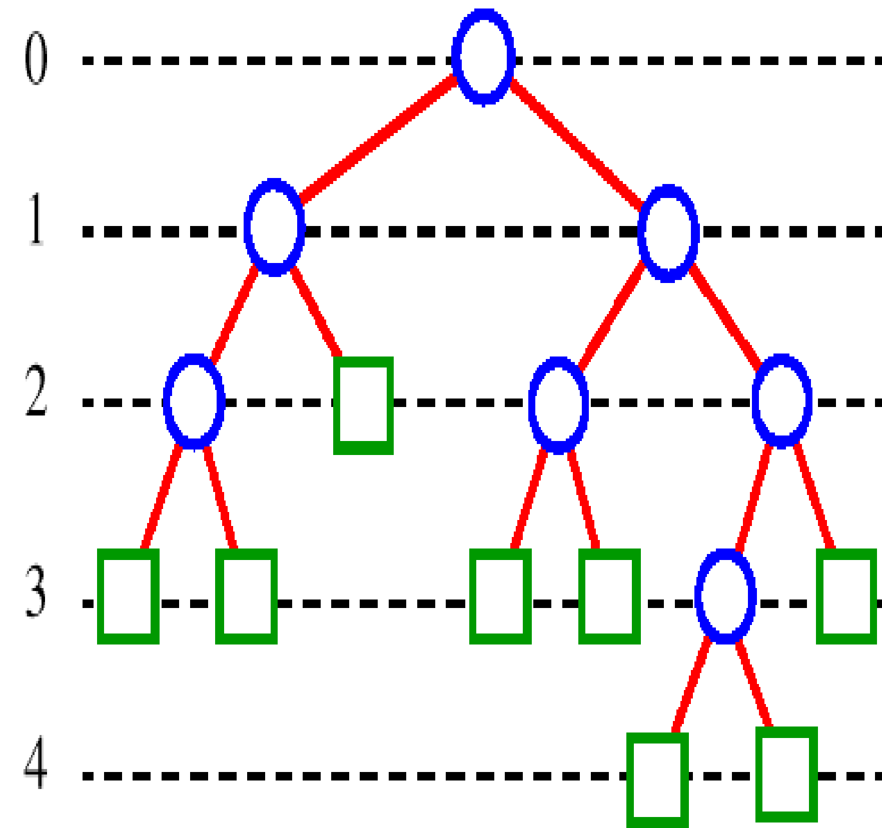
Cây Nhị Phân

- **Mỗi nút có tối đa 2 cây con**



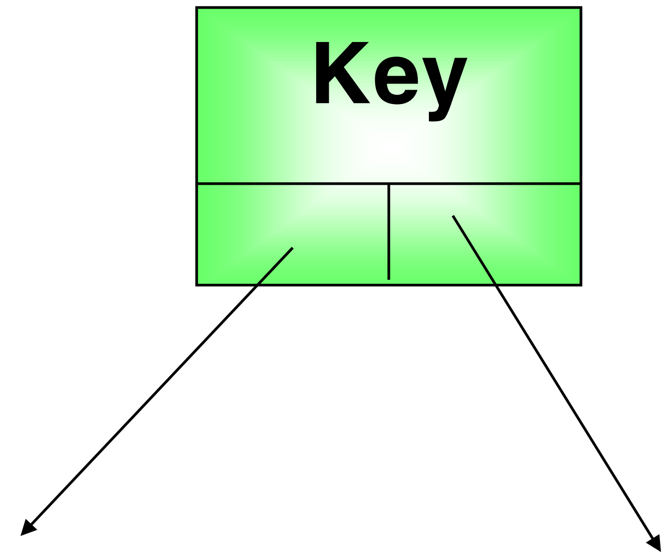
Một Số Tính Chất Của Cây Nhị Phân

- **Số nút nằm ở mức i ≤ 2i.**
- **Số nút lá ≤ 2h-1, với h là chiều cao của cây.**
- **Chiều cao của cây h ≥ log2(N)**
  - **N = số nút trong cây**
- **Số nút trong cây ≤ 2h-1.**



Cấu Trúc Dữ Liệu Của Cây Nhị Phân

```
typedef struct tagTNode
{
   Data      Key;
   struct tagTNode *pLeft;
   struct tagTNode *pRight;
}TNode;

typedef TNode *TREE;
```



Ví Dụ Cây Được Tổ Chức Trong Bộ Nhớ Trong

1f

2f	6	3f

2f

N	4	5f

3f

7f	9	N

5f

N	5	N

7f

N	8	N

Duyệt Cây Nhị Phân

- Có 3 trình tự thăm gốc :
  - Duyệt trước
  - Duyệt giữa
  - Duyệt sau
- Độ phức tạp O (log2(h))

  Trong đó h là chiều cao cây

Ví Dụ Kết Quả Của Phép Duyệt Cây

- NLR: 9, 2, 6, 1, 10, 8, 5, 3, 7, 12, 4.
- LNR: 6, 2, 10, 1, 9, 3, 5, 8, 12, 7, 4.
- Kết quả của phép duyệt : LRN, NRL,LRN, LNR?

Duyệt Trước

```
void  NLR(TREE Root)
{
   if (Root != NULL)
   {
         <Xử lý Root>; //Xử lý tương ứng theo nhu cầu
         NLR(Root->pLeft);
         NLR(Root->pRight);
   }
}
```

Duyệt Giữa

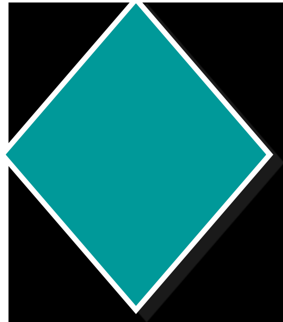
```
void LNR(TREE Root)
{
   if (Root != NULL)
   {
        LNR(Root->pLeft);
        <Xử lý Root>; // Xử lý tương ứng theo nhu
                                        cầu
        LNR(Root->pRight);
   }
}
```

Duyệt Sau


```
void    LRN(TREE Root)
{
  if (Root != NULL)
  {
      LRN(Root->pLeft);
      LRN(Root->pRight);
      <Xử lý Root>; // Xử lý tương ứng theo nhu cầu
  }
}
```

Biểu Diễn Cây Tổng Quát Bằng Cây Nhị Phân

NỘI DUNG

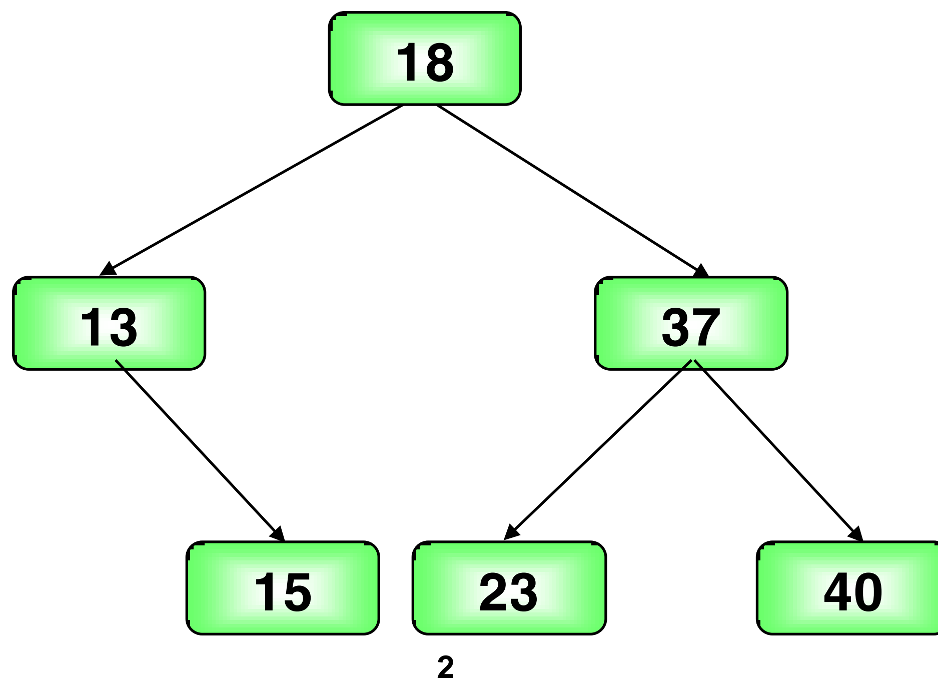


CÂY NHỊ PHÂN TÌM KIẾM

Định nghĩa cây nhị phân tìm kiếm

- Cây nhị phân
- Bảo đảm nguyên tắc bố trí khoá tại mỗi nút:
  - Các nút trong cây trái nhỏ hơn nút hiện hành
  - Các nút trong cây phải lớn hơn nút hiện hành

Ví dụ:



Ưu điểm của cây nhị phân tìm kiếm

- Nhờ trật tự bố trí khóa trên cây :
  - Định hướng được khi tìm kiếm
- Cây gồm N phần tử :
  - Trường hợp tốt nhất h = $\log_2 N$
  - Trường hợp xấu nhất h = Ln
  - Tình huống xảy ra trường hợp xấu nhất ?

Cấu trúc dữ liệu của cây nhị phân tìm kiếm

- *Cấu trúc dữ liệu của 1 nút*

```
typedef struct tagTNode
{
    int     Key; //trường dữ liệu là 1 số nguyên
    struct tagTNode *pLeft;
    struct tagTNode *pRight;
}TNode;
```

- *Cấu trúc dữ liệu của cây*

```
typedef TNode *TREE;
```

Các thao tác trên cây nhị phân tìm kiếm

- Tạo 1 cây rỗng
- Tạo 1 nút có trường Key bằng x
- Thêm 1 nút vào cây nhị phân tìm kiếm
- Xoá 1 nút có Key bằng x trên cây
- Tìm 1 nút có khoá bằng x trên cây

Tạo cây rỗng

- Cây rỗng -> địa chỉ nút gốc bằng NULL

```
void CreateTree(TREE &T)
{
    T=NULL;
}
```

Tạo 1 nút có Key bằng x

```
TNode *CreateTNode(int x)
{
	TNode *p;
	p = new TNode; //cấp phát vùng nhớ động
	if(p==NULL)
		exit(1); // thoát
	else
	{
		p->key = x; //gán trường dữ liệu của nút = x
		p->pLeft = NULL;
		p->pRight = NULL;
	}
	return p;
}
```

Thêm một nút x

- **<u>Rằng buộc</u>**: Sau khi thêm cây đảm bảo là cây nhị phân tìm kiếm.

```
int insertNode(TREE &T, Data X)
{ if(T)
  {     if(T->Key == X)       return 0;
        if(T->Key > X) return insertNode(T->pLeft, X);
        else    return insertNode(T->pRight, X);}
        T         = new TNode;
        if(T == NULL)         return -1;
        T->Key          = X;
        T->pLeft =T->pRight = NULL;
  return 1;
}
```

Minh họa thêm 1 phần tử vào cây

Thêm X=50

44 < X

88 > X

59 > X

55 > X

44

18 88

13 37 59 108

15 23 40 55 71

50

Tìm nút có khoá bằng x (không dùng đệ quy)

```
TNode * searchNode(TREE Root,  Data x)
{       Node *p = Root;
        while (p != NULL)
        {       if(x == p->Key) return p;
                else
                if(x < p->Key)   p = p->pLeft;
                else  p = p->pRight;
        }
        return NULL;
}
```

Tìm nút có khoá bằng x (dùng đệ quy)

```
TNode *SearchTNode(TREE T, int x)
{
    if(T!=NULL)
    {
        if(T->key==x)
            return T;
        else
            if(x>T->key)
                return SearchTNode(T->pRight,x);
            else
                return SearchTNode(T->pLeft,x);
    }
    return NULL;
}
```

Minh hoạ tìm một nút

Tìm X=55

55

44
18
88
13
37
59
108
15
23
40
55
71

Tìm thấy X=55

Minh hoạ thành lập 1 cây từ dãy số

9, 5, 4, 8, 6, 3, 14,12,13

9
5
14
4
8
12
3
6
13

Hủy 1 nút có khoá bằng X trên cây

- Hủy 1 phần tử trên cây phải đảm bảo điều kiện ràng buộc của Cây nhị phân tìm kiếm
- Có 3 trường hợp khi hủy 1 nút trên cây
  - TH1: X là nút lá
  - TH2: X chỉ có 1 cây con (cây con trái hoặc cây con phải)
  - TH3: X có đầy đủ 2 cây con
- TH1: Ta xoá nút lá mà không ảnh hưởng đến các nút khác ttrên cây
- TH2: Trước khi xoá x ta móc nối cha của X với con duy nhất cùa X.
- TH3: Ta dùng cách xoá gián tiếp

Minh hoạ hủy phần tử x có 1 cây con

Hủy X=37

44
18 88
13 37 59 108
15 23 55 71

Hủy 1 nút có 2 cây con

- Ta dùng cách hủy gián tiếp, do X có 2 cây con
- Thay vì hủy X ta tìm phần tử thế mạng Y. Nút Y có tối đa 1 cây con.
- Thông tin lưu tại nút Y sẽ được chuyển lên lưu tại X.
- Ta tiến hành xoá hủy nút Y (xoá Y giống 2 trường hợp đầu)
- Cách tìm nút thế mạng Y cho X: Có 2 cách
  - C1: Nút Y là nút có khoá nhỏ nhất (trái nhất) bên cây con phải X
  - C2: Nút Y là nút có khoá lớn nhất (phải nhất) bên cây con trái của X

Minh họa hủy phần tử X có 2 cây con

Xoá nút có trường

Key = 18, lúc đó nút có khoá 23 là nút thế mạng

44
18
88
13
37
59
108
15
23
40
55
71
30

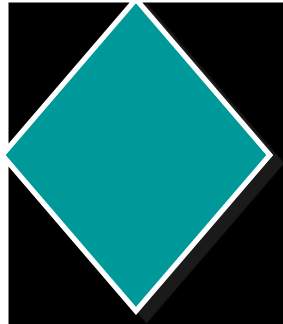
Cài đặt thao tác xoá nút có trường Key = x

```
void DeleteNodeX1(TREE &T,int x)
{
    if(T!=NULL)
    {
        if(T->Key<x)        DeleteNodeX1(T->Right,x);
        else
        {
            if(T->Key>x)        DeleteNodeX1(T->Left,x);
            else  //tim thấy  Node có trường dữ liệu = x
            {       TNode *p;
                    p=T;
                    if (T->Left==NULL)          T = T->Right;
                    else
                    {       if(T->Right==NULL)          T=T->Left;
                            else        ThayThe1(p, T->Right);// tìm bên cây con
    phải
                    }
                    delete p;
            }
        }
    }
    else  printf("Khong tim thay phan can xoa tu");}
```

Hàm tìm phần tử thế mạng

```
void ThayThe1(TREE &p, TREE &T)
{   if(T->Left!=NULL)
        ThayThe1(p,T->Left);
    else
    {
        p->Key = T->Key;
        p=T;
        T=T->Right;
    }
}
```

NỘI DUNG



CÂY NHỊ PHÂN TÌM KIẾM CÂN BẰNG

Định nghĩa

- Cây nhị phân tìm kiếm cân bằng là cây mà tại mỗi nút của nó độ cao của cây con trái và của cây con phải chênh lệch không quá một

Ví dụ:

Tổ chức dữ liệu

- **Chỉ số cân bằng = độ lệch giữa cây trái và cây phải của một nút**
- **Các giá trị hợp lệ :**
  - **CSCB(p) = 0 ⇔ Độ cao cây trái (p) = Độ cao cây phải (p)**
  - **CSCB(p) = 1 ⇔ Độ cao cây trái (p) < Độ cao cây phải (p)**
  - **CSCB(p) = -1 ⇔ Độ cao cây trái (p) > Độ cao cây phải (p)**

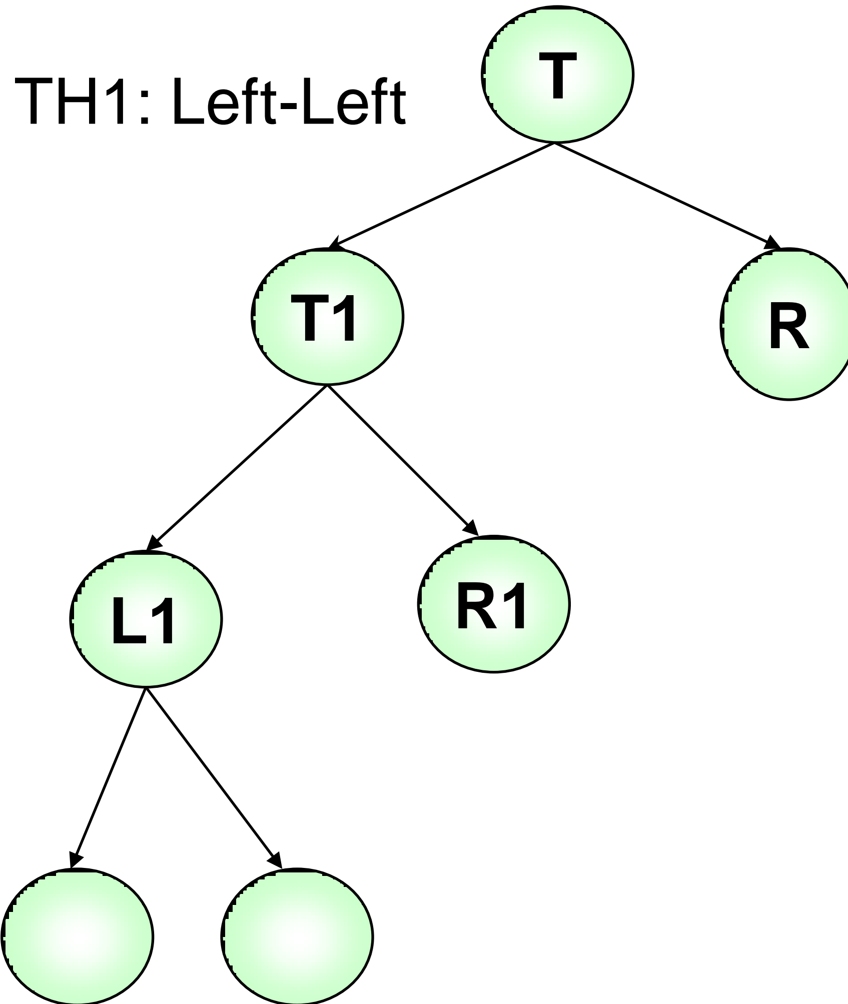
Tổ chức dữ liệu(tt)


```
#define LH -1 //cây con trái cao hơn
#define EH 0 //cây con trái bằng cây con phải
#define RH 1 //cây con phải cao hơn
typedef struct tagAVLNode
{  char    balFactor; //chỉ số cân bằng
   Data    key;
   struct tagAVLNode*      pLeft;
   struct tagAVLNode*      pRight;
}AVLNode;
typedef AVLNode    *AVLTree;
```

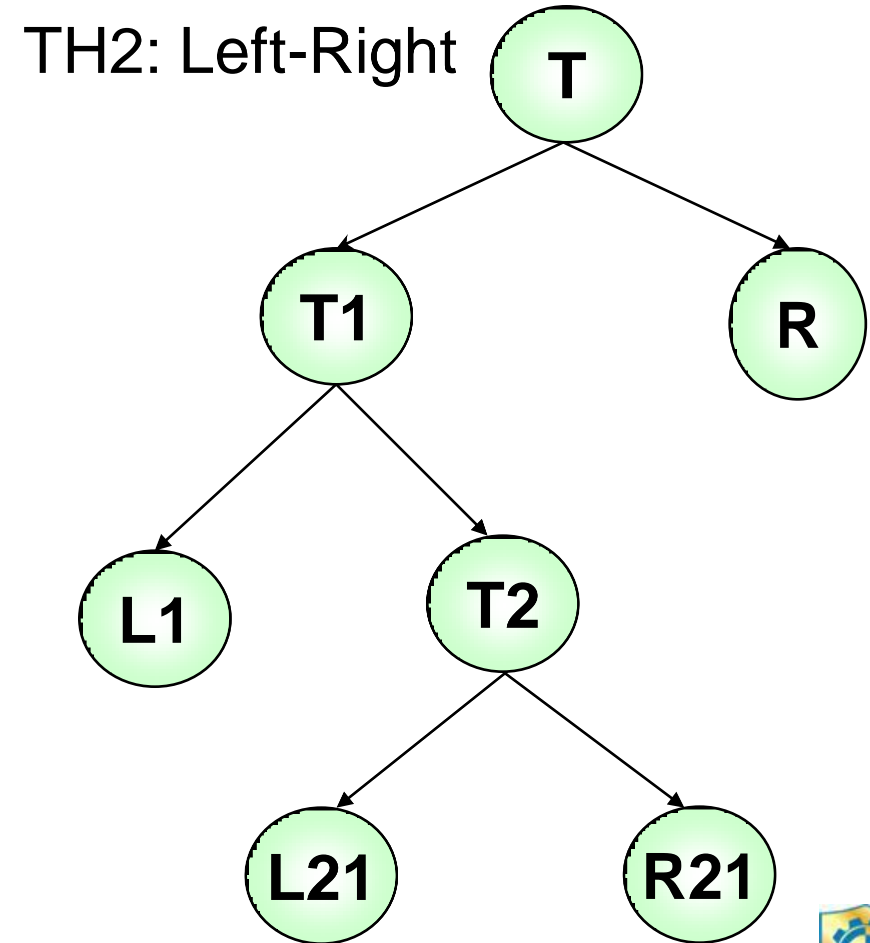
Các trường hợp mất cân bằng do lệch trái

- Cây mất cân bằng tại nút T

TH1: Left-Left



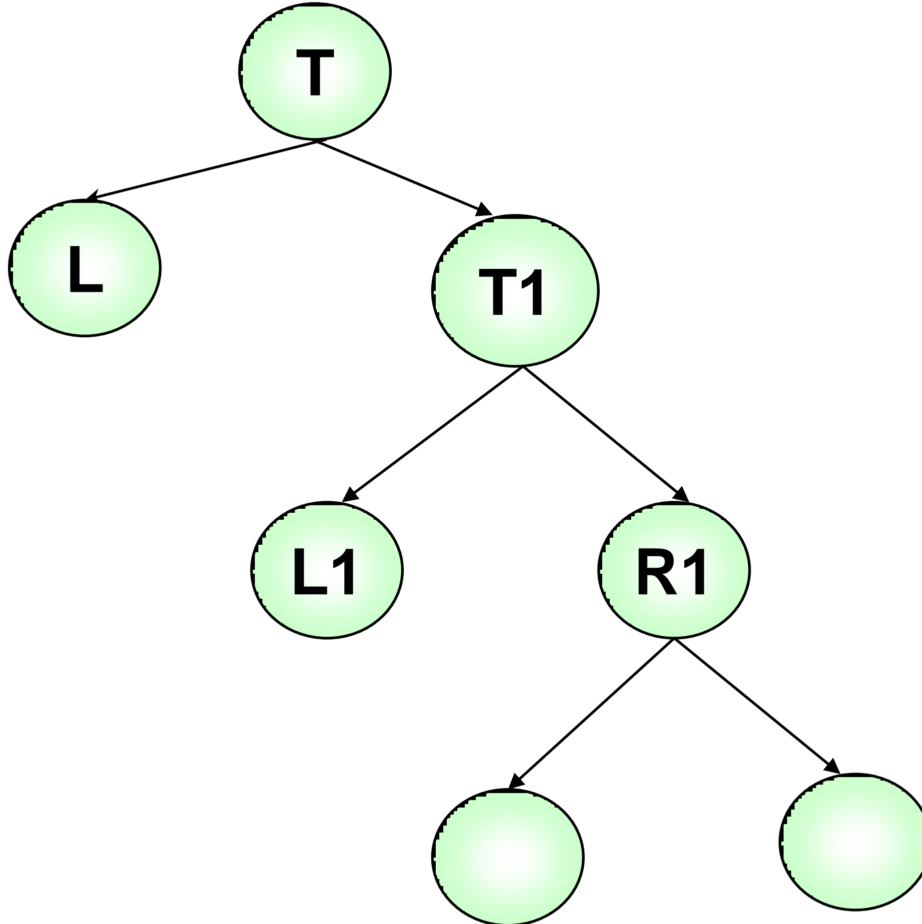
TH2: Left-Right



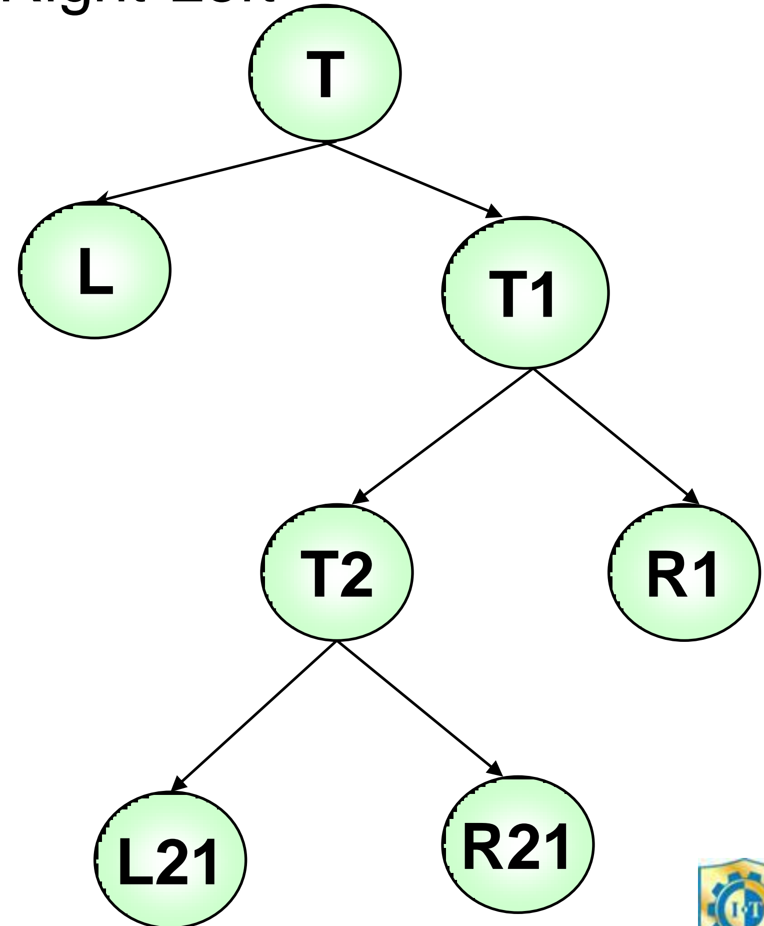
Các trường hợp mất cân bằng do lệch phải

- Cây mất cân bằng tại nút T

TH3: Right-Right



TH4: Right-Left



Các thao tác trên cây cân bằng

- Khi thêm hay xoá 1 nút trên cây, cĩ thể làm cho cây mất tính cân bằng, khi ấy ta phải tiến hành cân bằng lại.
- Cây có khả năng mất cân bằng khi thay đổi chiều cao:
  - Lệch nhánh trái, thêm bên trái
  - Lệch nhánh phải, thêm bên phải
  - Lệch nhánh trái, hủy bên phải
  - Lệch nhánh phải, hủy bên trái
- Cân bằng lại cây : tìm cách bố trí lại cây sao cho chiều cao 2 cây con cân đối:
  - Kéo nhánh cao bù cho nhánh thấp
  - Phải bảo đảm cây vẫn là Nhị phân tìm kiếm

Cân bằng lại trường hợp 1

Cài đặt cân bằng lại cho trường hợp 1

```
void LL(AVLTree &T)
{
   AVLNode *T1=T->pLeft;
   T->pLeft = T1->pRight;
   T1->pRight=T;
   switch(T1-> balFactor)
   { case LH:    T-> balFactor =EH;
                 T1->balFactor=EH; break;
     case EH:    T->balFactor=LH;
                 T1->balFactor =RH; break;
   }
   T=T1;
}
```

Cân bằng lại trường hợp 2

T

T1 R

L1 T2

L21 R21

T2

T1 T

L1 L21 R21 R

Cài đặt cân bằng lại cho trường hợp 2

```
void LR(AVLTree &T)
{   AVLNode *T1=T->pLeft;
    AVLNode *T2=T1->pRight;
    T->pLeft=T2->pRight;
    T2->pRight=T;
    T1->pRight= T2->pLeft;
    T2->pLeft = T1;
    switch(T2->balFactor)
    {     case LH:     T->balFactor=RH;
                       T1->balFactor=EH; break;
          case EH:     T->balFactor = EH;
                       T1->balFactor=EH; break;
          case RH:     T->balFactor =EH;
                       T1->balFactor= LH; break;
    }T2->balFactor =EH; T=T2}
```

Cân bằng lại trường hợp 3

Cài đặt cân bằng lại cho trường hợp 3

```
void RR(AVLTree &T)
{  AVLNode  *T1= T->pRight;
   T->pRight=T1->pLeft;
   T1->pLeft=T;
   switch(T1-> balFactor)
   {
       case RH:   T-> balFactor = EH;
                  T-> balFactor = EH; break;
       case EH:   T-> balFactor = RH;
                  T1-> balFactor = LH; break;
   }
   T=T1
}
```

Cân bằng lại trường hợp 4

Cài đặt cân bằng lại cho trường hợp 4

```
void RR(AVLTree &T)
{   AVLNode  *T1= T->pRight;
    AVLNode *T2=T1->pLeft;
    T->pRight = T2->pLeft;
    T2->pLeft = T;
    T1->pLeft = T2->pRight;
    T2->pRight = T1;
    switch(T2-> balFactor)
    {     case RH:        T-> balFactor = LH;
                          T1-> balFactor = EH; break;
          case EH:        T-> balFactor = EH;
                          T1-> balFactor = EH; break;
          case LH:        T-> balFactor = EH;
                          T1-> balFactor = RH; break;
    }
    T2-> balFactor =EH; T=T2;}
```

Thêm 1 nút

- Thêm bình thường như trường hợp cây NPTK
- Nếu cây tăng trưởng chiều cao
  - Lần ngược về gốc để phát hiện nút bị mất cân bằng
  - Tiến hành cân bằng lại nút đó bằng thao tác cân bằng thích hợp
- Việc cân bằng lại chỉ cần thực hiện 1 lần nơi mất cân bằng

Hủy 1 nút

- Hủy bình thường như trường hợp cây NPTK
- Nếu cây giảm chiều cao:
  - Lần ngược về gốc để phát hiện nút bị mất cân bằng
  - Tiến hành cân bằng lại nút đó bằng thao tác cân bằng thích hợp
  - Tiếp tục lần ngược lên nút cha…
- Việc cân bằng lại co thể lan truyền lên tận gốc