



KHOA CÔNG NGHỆ THÔNG TIN – ĐẠI HỌC ĐÀ LẠT



Chương 2
Tìm kiếm và sắp xếp trong



Mục tiêu



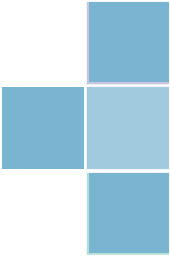
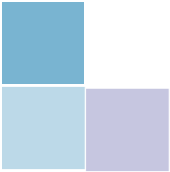
- ❖ Giới thiệu một số thuật giải tìm kiếm và sắp xếp trong.
- ❖ Phân tích, đánh giá độ phức tạp của cc thuật giải tìm kiếm, sắp xếp trong các trường hợp xấu nhất, tốt nhất.



Nội dung

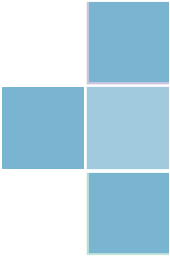
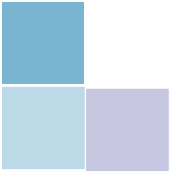


- I. Nhu cầu tìm kiếm và sắp xếp dữ liệu trong một hệ thống thông tin
- II. Các thuật giải tìm kiếm trong
- III. Các thuật giải sắp xếp trong



I. Nhu cầu tìm kiếm và sắp xếp dữ liệu trong một hệ thống thông tin

- ❖ Thực hiện thao tác tìm kiếm để khai thác thông tin.
- ❖ Xây dựng các thuật giải cho phép tìm kiếm nhanh sẽ có nhiều ý nghĩa trong khối lượng dữ liệu lớn.
- ❖ Có nhiều thuật giải tìm kiếm và sắp xếp, chọn lựa thuật giải thích hợp tương ứng với ứng dụng hoặc dựa vào đặc trưng lưu trữ của dữ liệu



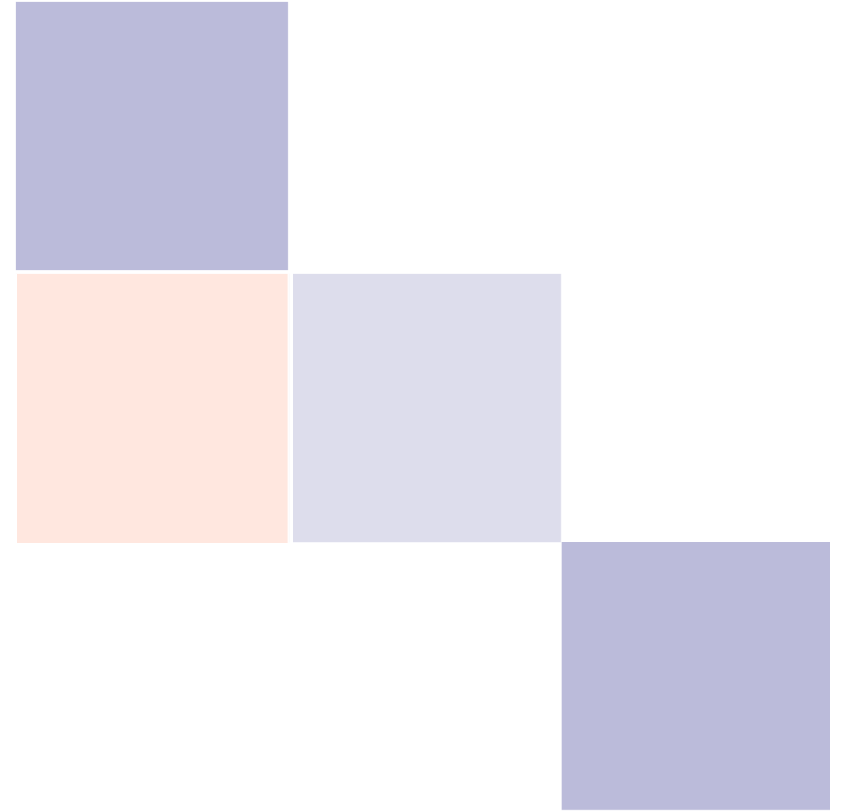
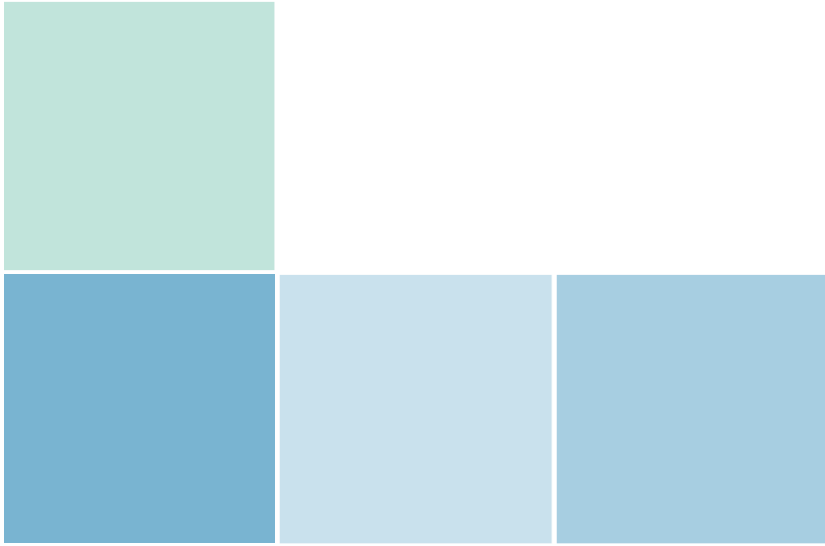
I. Nhu cầu tìm kiếm và sắp xếp dữ liệu trong một hệ thống thông tin

❖ Đặc trưng lưu trữ của dữ liệu:

- trong bộ nhớ trong
- trong bộ nhớ phụ (thiết bị lưu trữ ngoài)

do đặc điểm khác nhau của thiết bị lưu trữ, các thuật giải tìm kiếm và sắp xếp được xây dựng cho các cấu trúc lưu trữ trên bộ nhớ trong hoặc bộ nhớ phụ cũng có những đặc thù khác nhau.

❖ Chương này trình bày các thuật giải sắp xếp và tìm kiếm dữ liệu được lưu trữ trên bộ nhớ trong - gọi là các thuật giải *tìm kiếm và sắp xếp trong*.



II. Các thuật giải tìm kiếm trong

II.1. Tìm kiếm tuyến tính.

II.2. Tìm kiếm nhị phân.



Bài toán tìm kiếm



❖ Tập dữ liệu được lưu trữ là dãy số a_0, a_2, \dots, a_{N-1}

int a[MAX];

❖ Khoá cần tìm là x

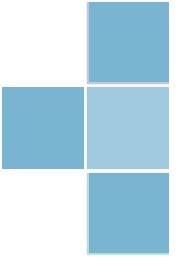
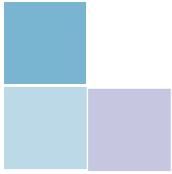
int x;



Tìm kiếm tuyến tính



- ❖ Bước 1: $i = 0$; // bắt đầu từ phần tử đầu tiên của dãy
- ❖ Bước 2: So sánh $a[i]$ với x , có 2 khả năng :
 - $a[i] = x$: Tìm thấy. Dừng
 - $a[i] \neq x$: Sang Bước 3.
- ❖ Bước 3 : $i = i + 1$; // xét tiếp phần tử kế trong mảng
 - Nếu $i > N - 1$: Hết mảng, không tìm thấy. Dừng
 - Ngược lại: Lặp lại Bước 2.



Tìm kiếm tuyến tính

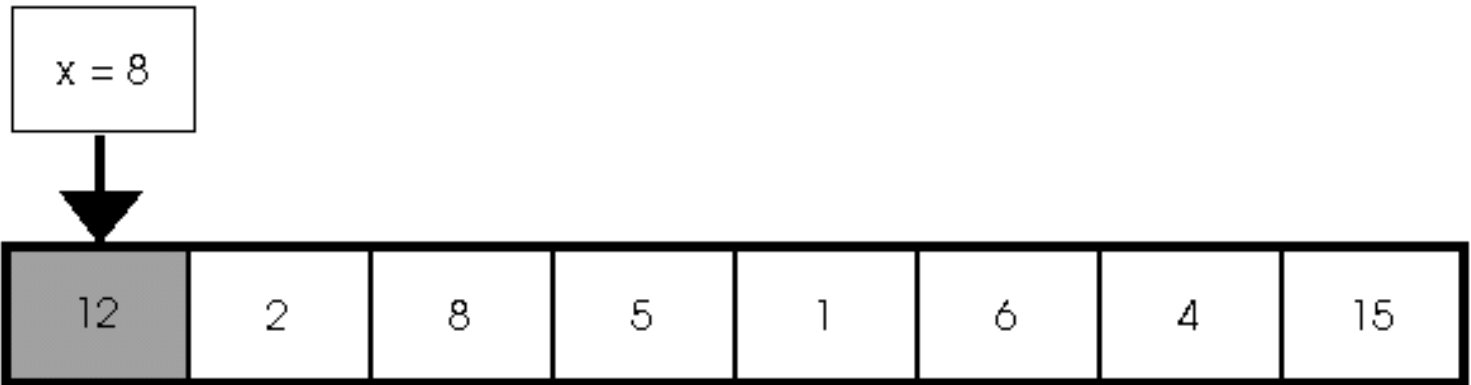
❖ Ví dụ: Cho dãy số a

12 2 8 5 1 6 4 15

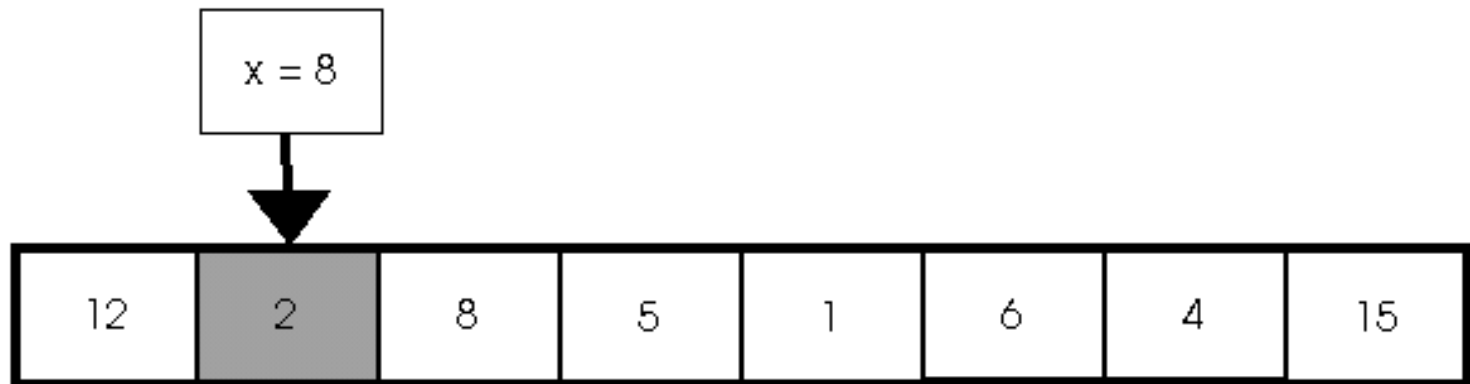
■ Giá trị cần tìm: 8



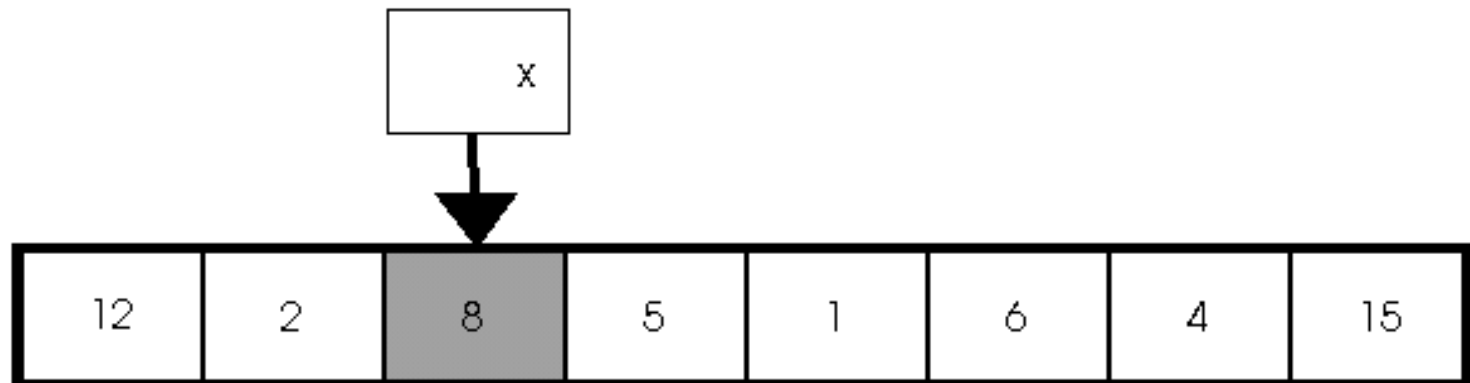
$i = 0$



$i = 1$



$i = 2$



Tìm kiếm tuyến tính

❖ Cài đặt

```
int LinearSearch(int a[], int n, int x)
{
    int i = 0;
    while ((i < n) && (a[i]) != x)
        i++;
    if (i == n)
        return 0; // tìm hết mảng nhưng không có x
    return 1; // tìm thấy x tại vị trí x
}
```

❖ Độ phức tạp thuật toán $O(n)$

Tìm kiếm tuyến tính

- ❖ Để giảm bớt số phép so sánh chỉ số trong biểu thức điều kiện của lệnh if và lệnh while, ta dùng một biến phụ đóng vai trò là lính canh bên phải (hay trái) $x_n = \text{item cần tìm}$ (x_0 trong trường hợp dãy đánh số từ 1).

```
int LinearSearchNew(int a[], int n, int x)
{
    int i = 0;
    a[n] = x; // đặt phần tử lính canh
    while (a[i] != x)
        i++;
    if (i == n)
        return 0; // tìm hết mảng nhưng không có x
    return 1; // tìm thấy x
}
```

Tìm kiếm nhị phân

- ❖ Thuật giải tìm kiếm nhị phân chỉ dùng cho các dãy số có thứ tự (tăng hay giảm). Không mất tính tổng quát ta xét các dãy có thứ tự tăng.
- ❖ Phát biểu bài toán: Tìm x có trong dãy tăng a ?
 - Input : $a_0, a_1, \dots, a_{n-1}, x$
int a[n], x;
 - Output :
 - Nếu có, trả về 1 ;
 - Nếu không có, trả về 0
- ❖ Ý tưởng (chia để trị): Giả sử dãy đã có thứ tự tăng : $i < j \Rightarrow a_i \leq a_j$
 - Nếu $x > a_k$ thì x chỉ có thể xuất hiện trong đoạn $[a_{k+1}, a_{n-1}]$
 - Nếu $x < a_k$ thì x chỉ có thể xuất hiện trong đoạn $[a_0, a_{k-1}]$

Tìm kiếm nhị phân

- ❖ Thuật giải tìm kiếm nhị phân chỉ dùng cho các dãy số có thứ tự (tăng hay giảm). Không mất tính tổng quát ta xét các dãy có thứ tự tăng.
- ❖ Phát biểu bài toán: Tìm x có trong dãy tăng a ?
 - Input : $a_0, a_1, \dots, a_{n-1}, x$
int a[n], x;
 - Output :
 - Nếu có, trả về 1 ;
 - Nếu không có, trả về 0
- ❖ **Ý tưởng (chia để trị):** Giả sử dãy đã có thứ tự tăng: $i < j \Rightarrow a_i \leq a_j$
 - Nếu $x > a_k$ thì x chỉ có thể xuất hiện trong đoạn $[a_{k+1}, a_{n-1}]$
 - Nếu $x < a_k$ thì x chỉ có thể xuất hiện trong đoạn $[a_0, a_{k-1}]$

Tìm kiếm nhị phân

- ❖ Bước 1: $\text{left} = 0$; $\text{right} = N-1$; // tìm trên tất cả các phần tử
- ❖ Bước 2: $\text{midle} = (\text{left} + \text{right}) / 2$; // lấy mốc so sánh
 - So sánh $a[\text{midle}]$ với x , có 3 khả năng :
 - $a[\text{midle}] = x$: Tìm thấy. Dừng
 - $a[\text{midle}] > x$: //tìm tiếp x trong dãy con $a_{\text{left}} \dots a_{\text{midle} - 1}$
 - $\text{right} = \text{midle} - 1$;
 - $a[\text{midle}] < x$: //tìm tiếp x trong dãy con $a_{\text{midle} + 1} \dots a_{\text{right}}$
 - $\text{left} = \text{midle} + 1$;
- ❖ Bước 3:
 - Nếu $\text{left} \leq \text{right}$ //còn phần tử chưa xét \Rightarrow tìm tiếp.
 - Lặp lại Bước 2.
 - Ngược lại: Dừng; //Đã xét hết tất cả các phần tử.

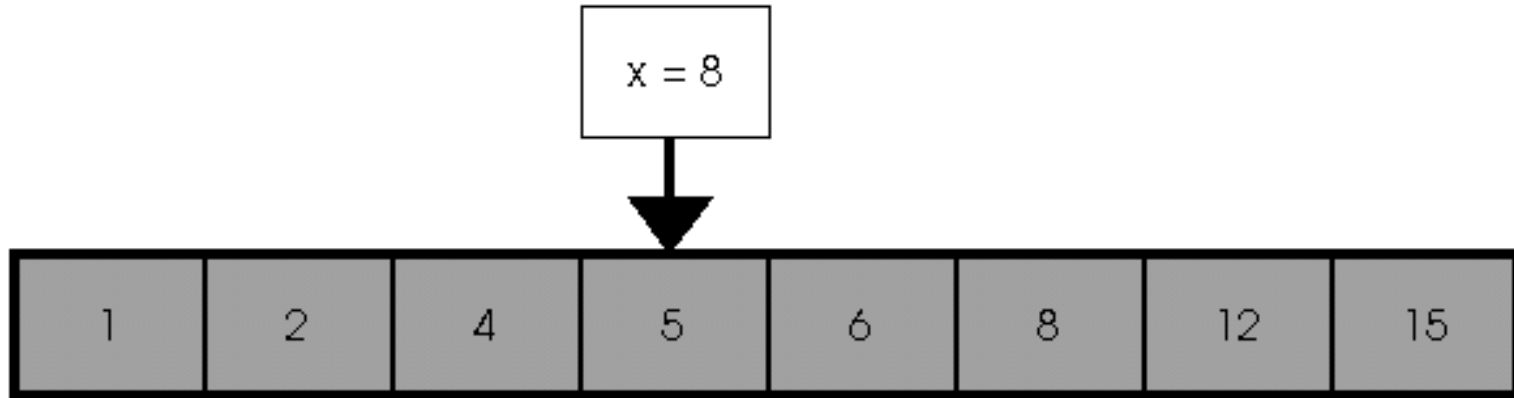
Tìm kiếm nhị phân

❖ Ví dụ: Cho dãy số a gồm 8 phần tử:

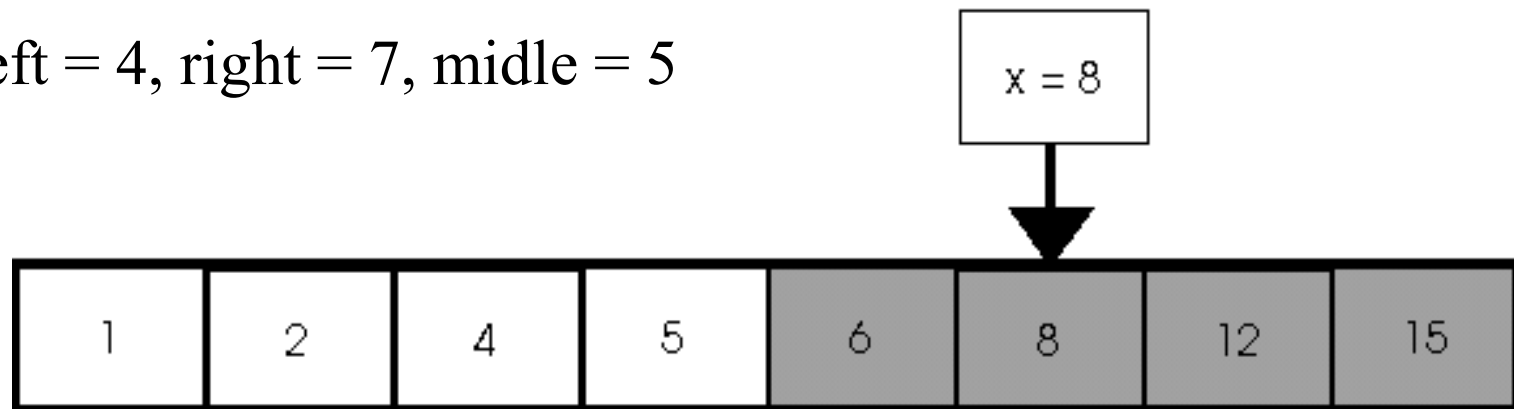
1 2 4 5 6 8 12 15

▪ Giá trị cần tìm: $x = 8$

left = 0, right = 7, middle = 3



left = 4, right = 7, middle = 5



Tìm kiếm nhị phân

```
int BinarySearch(int a[], int n, int x)
{
    int left = 0, right = n - 1, mid;
    do
    {
        mid = (left + right) / 2;
        if (x == a[mid])
            return 1; // tìm thấy x tại vị trí mid
        else
            if (x < a[mid])
                right = mid - 1;
            else
                left = mid + 1;
    } while (left <= right);
    return 0; // tìm hết dãy mà không có x
}
```

Tìm kiếm nhị phân

❖ Có thể cài đặt đệ quy như sau :

```
int BinarySearch_R(int a[], int x, int l, int r)
{
    int mid;
    if (l > r)
        return 0;
    mid = (l + r) / 2;
    if (x == a[mid])
        return 1;
    else
        if (x > a[mid])
            return BinarySearch_R(a, x, mid + 1, r);
        else
            return BinarySearch_R(a, x, l, mid - 1);
}
```



III. Các thuật giải sắp xếp trong

- III.1. Chọn trực tiếp (Selection Straight sort)
- III.2. Chèn trực tiếp (Insertion Straight sort)
- III.3. Đổi chỗ trực tiếp (Interchange Straight sort)
- III.4. Nổi bọt (Bubble sort)
- III.5. Chèn nhị phân (Binary Insertion Sort)
- III.6. Heap Sort
- III.7. Quick sort
- III.8. Merge Sort
- III.9. Radix Sort



Bài toán sắp xếp



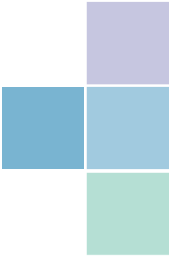
- ❖ Sắp xếp là quá trình xử lý một danh sách các phần tử để đặt chúng theo một thứ tự thỏa mãn một tiêu chuẩn nào đó dựa trên nội dung thông tin lưu trữ tại mỗi phần tử.
- ❖ Khái niệm nghịch thế:
 - Xét một mảng các số a_0, a_1, \dots, a_{n-1} .
 - Nếu có $i < j$ và $a_i > a_j$, thì ta gọi đó là một nghịch thế.
- ❖ Mảng chưa sắp xếp sẽ có nghịch thế.
- ❖ Mảng đã có thứ tự (tăng, giảm) sẽ không chứa nghịch thế, chẳng hạn:
 - $a_0 \leq a_1 \leq \dots \leq a_{n-1}$



Chọn trực tiếp (Selection Straight sort)



- ❖ Thuật giải tiến hành trong $n-1$ bước, Với mọi $i = 0, \dots, n-2$, tìm min trong đoạn $[a_i, a_{n-1}]$ rồi đưa Min về đầu dãy $a_i, a_{i+1}, \dots, a_{n-1}$
- ❖ Các bước:
 - Bước 1 : $i = 0$;
 - Bước 2 : Tìm phần tử $a[\text{min}]$ nhỏ nhất trong dãy hiện hành từ $a[i]$ đến $a[N-1]$
 - Bước 3 : Hoán vị $a[\text{min}]$ và $a[i]$
 - Bước 4 :
 - Nếu $i < N-1$ thì $i = i+1$; Lặp lại Bước 2
 - Ngược lại: Dừng. // $N-1$ phần tử đã nằm đúng vị trí.

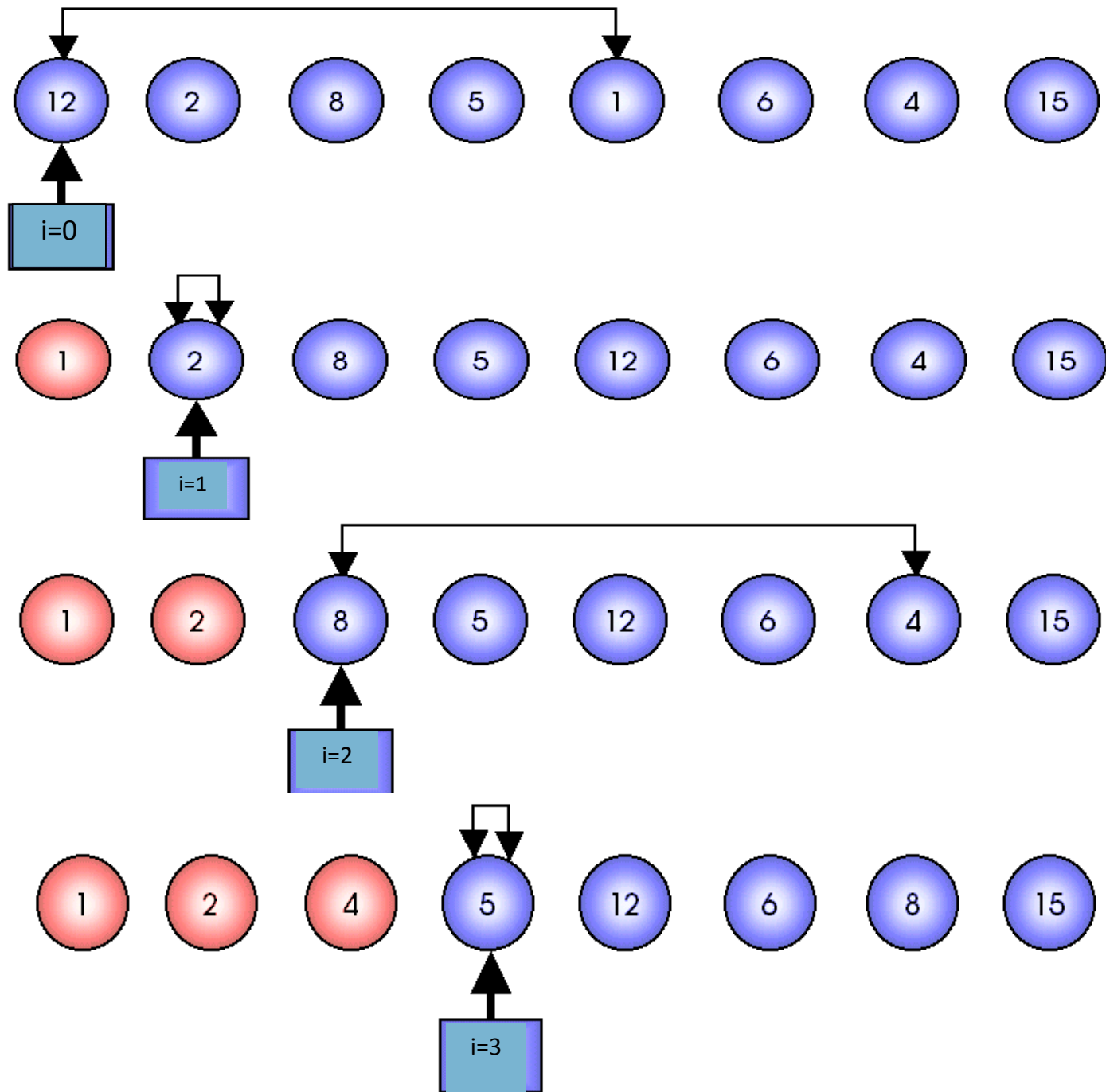
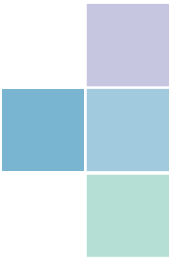


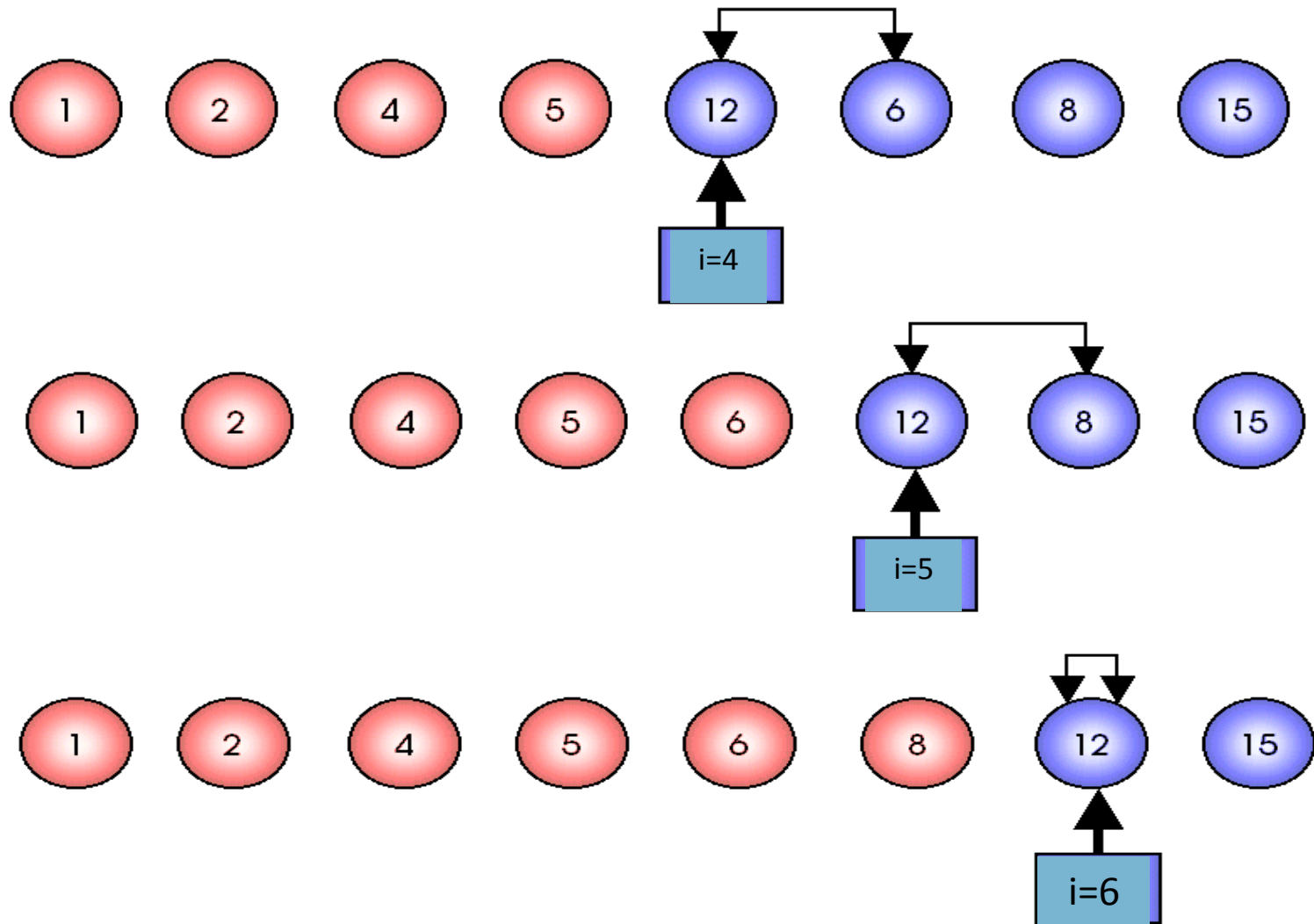
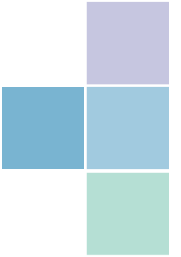
Chọn trực tiếp (Selection Straight sort)

Cho mảng a:

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

Sắp tăng a.





Chọn trực tiếp (Selection Straight sort)

❖ Cài đặt

```
void SelectionSort(int a[], int N)
{
    int i, j, Cs_min; // chỉ số phần tử nhỏ nhất trong dãy hiện hành
    for (i = 0; i < N - 1; i++)
    {
        Cs_min = i;
        for (j = i + 1; j < N; j++)
            if (a[j] < a[Cs_min])
                Cs_min = j; // ghi nhận vị trí phần tử hiện nhỏ nhất
        HoanVi(a[Cs_min], a[i]);
    }
}
```

❖ Đánh giá thuật giải:

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



Sắp xếp chèn (Insertion Sort)



❖ Mô tả thuật toán:

Bước 1: $i = 1$; // đoạn có 1 phần tử $a[0]$ đã được sắp

Bước 2: $x = a[i]$; Tìm vị trí pos thích hợp trong đoạn $a[0]$ đến $a[i-1]$ để chèn x vào.

Bước 3: Dời 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 $a[0]..a[i]$ đã được sắp

Bước 5: $i = i + 1$;

Nếu $i < N - 1$: Lặp lại Bước 2.

Ngược lại : Dừng.

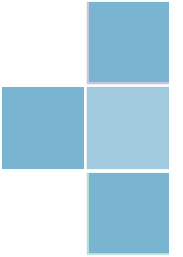
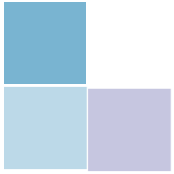


Sắp xếp chèn (Insertion Sort)



❖ Ý tưởng:

- Giả sử có một dãy a_0, a_1, \dots, a_{N-1} trong đó i phần tử đầu tiên a_0, a_1, \dots, a_{i-1} đã có thứ tự. Ý tưởng chính của phương pháp chèn trực tiếp là tìm cách chèn phần tử a_i vào vị trí thích hợp của đoạn đã được sắp để có dãy mới a_0, a_1, \dots, a_i có thứ tự.
- Vị trí này có thể là:
 - Trước a_0
 - Sau a_{i-1}
 - Giữa hai phần tử a_{k-1} và a_k thỏa $a_{k-1} \leq a_i < a_k$ ($1 \leq k \leq i-1$).

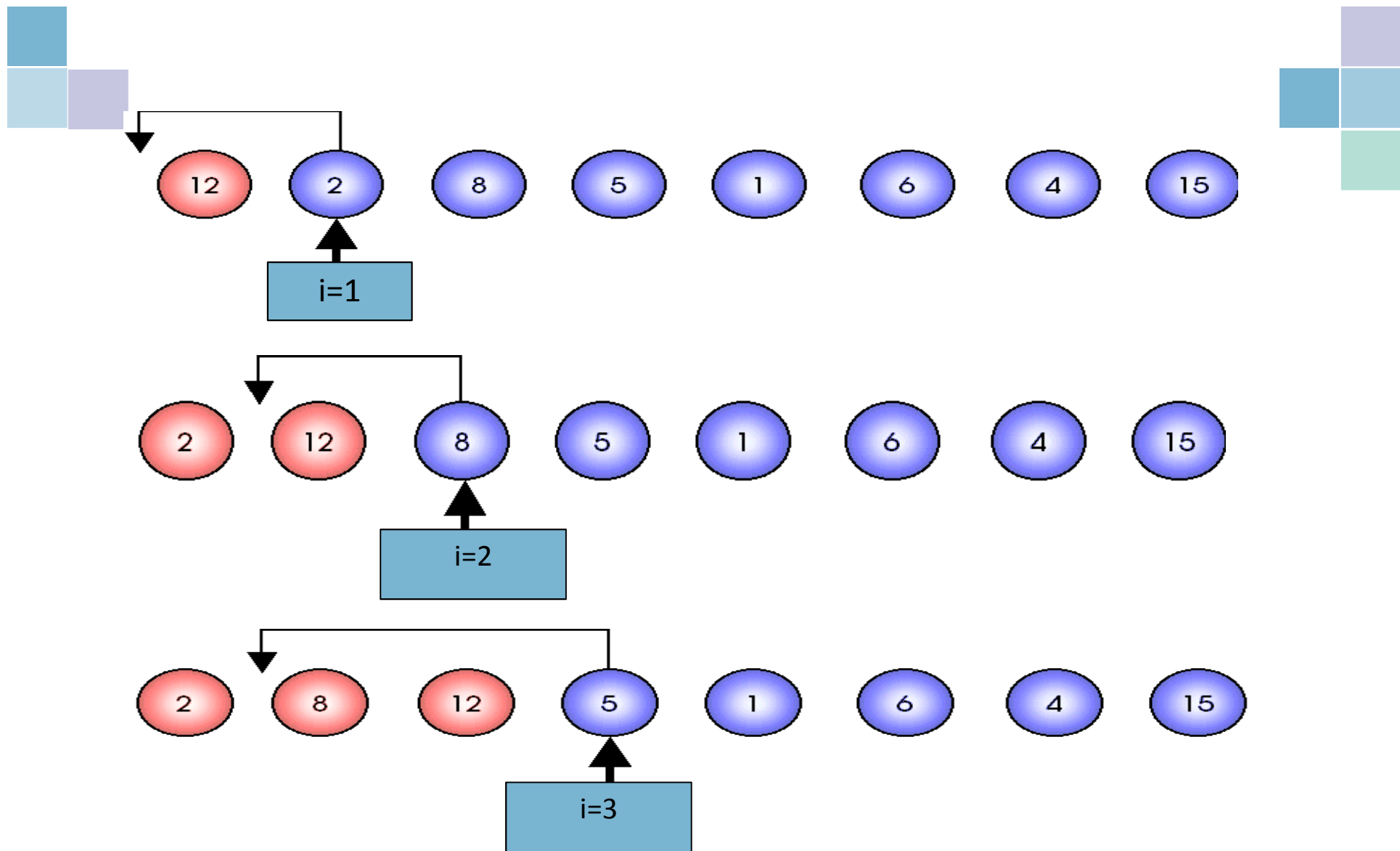


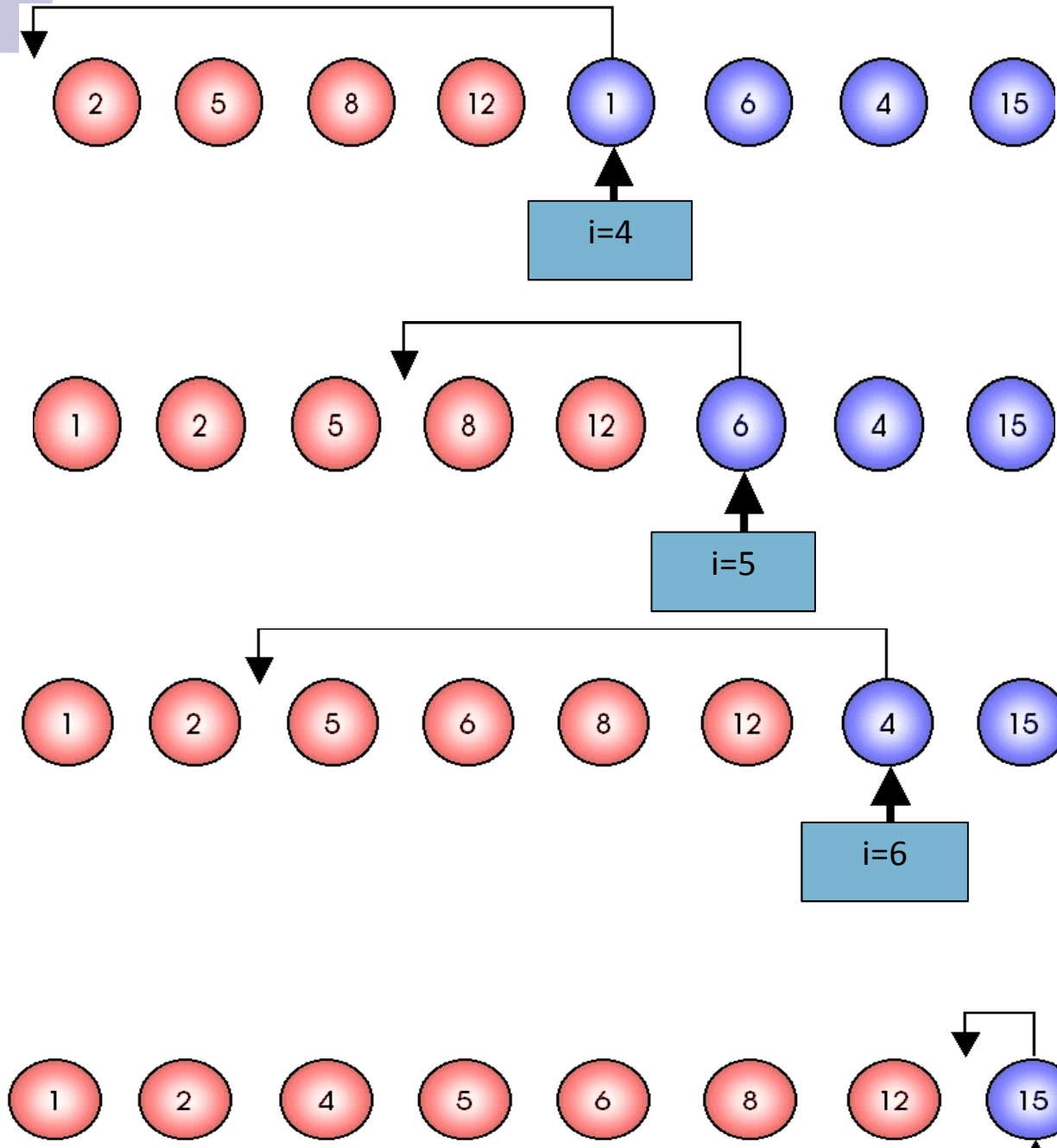
Sắp xếp chèn (Insertion Sort)

Cho mảng a:

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

Sắp tăng a.





Sắp xếp chèn (Insertion Sort)

❖ Cài đặt:

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

❖ Đánh giá thuật toán

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

Sắp xếp đổi chỗ trực tiếp (Interchange Sort)

❖ **Ý tưởng:** xuất phát từ đầu dãy, tìm tất cả các nghịch thế chứa phần tử này, triệt tiêu chúng bằng cách đổi chỗ phần tử này với phần tử tương ứng trong cặp nghịch thế. Lặp lại việc xử lý trên với các phần tử tiếp theo trong dãy.

❖ **Mô tả thuật giải:**

Bước 1: $i=0$; // bắt đầu từ đầu dãy

Bước 2: $j=i+1$; // tìm các phần tử $a[j] < a[i]$, $j > i$

Bước 3:

Trong khi $j < N$ thực hiện

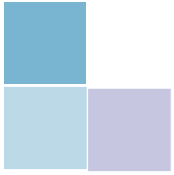
Nếu $a[j] < a[i]$: Hoán vị ($a[i], a[j]$)

$j=j+1$

Bước 4: $i=i+1$;

Nếu $i < N-1$: lặp lại bước 2

Ngược lại: dừng

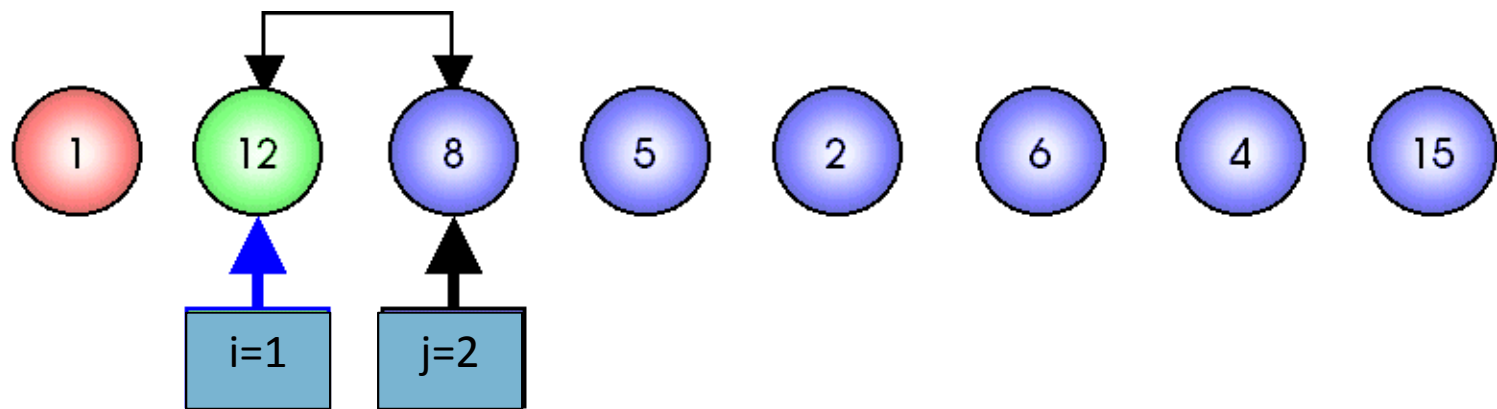
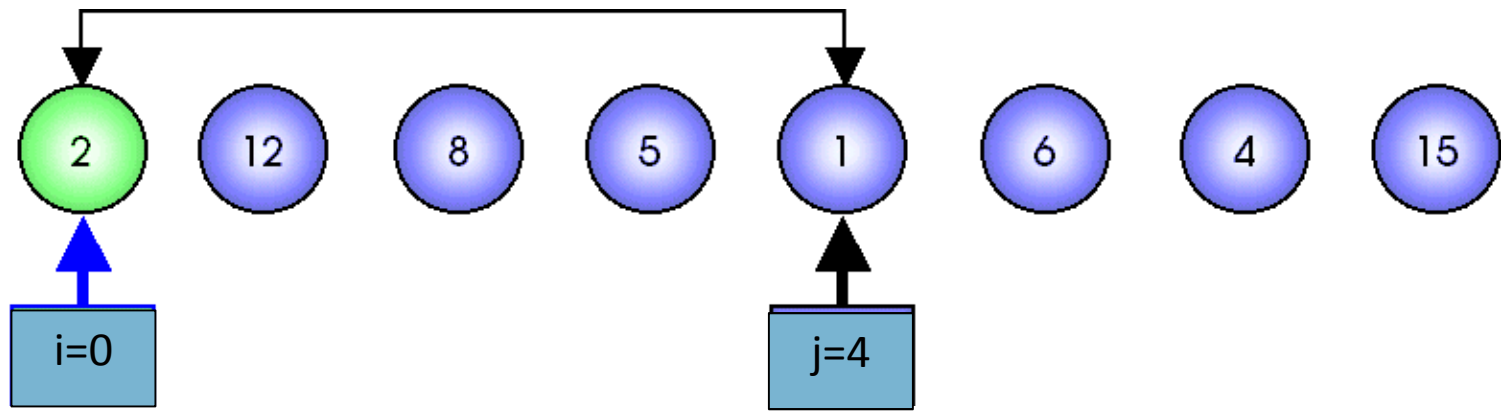
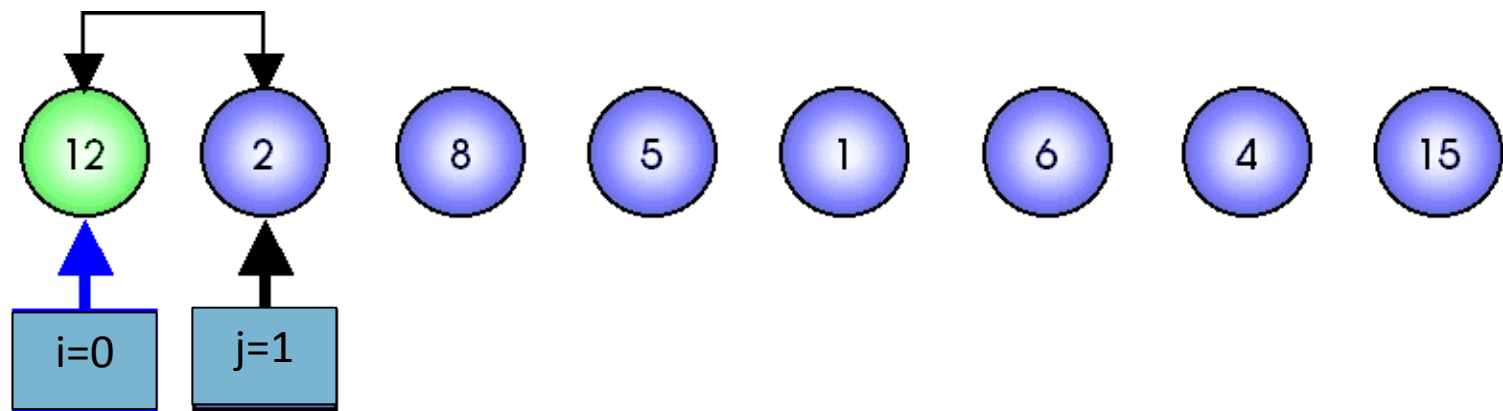
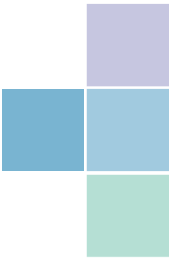


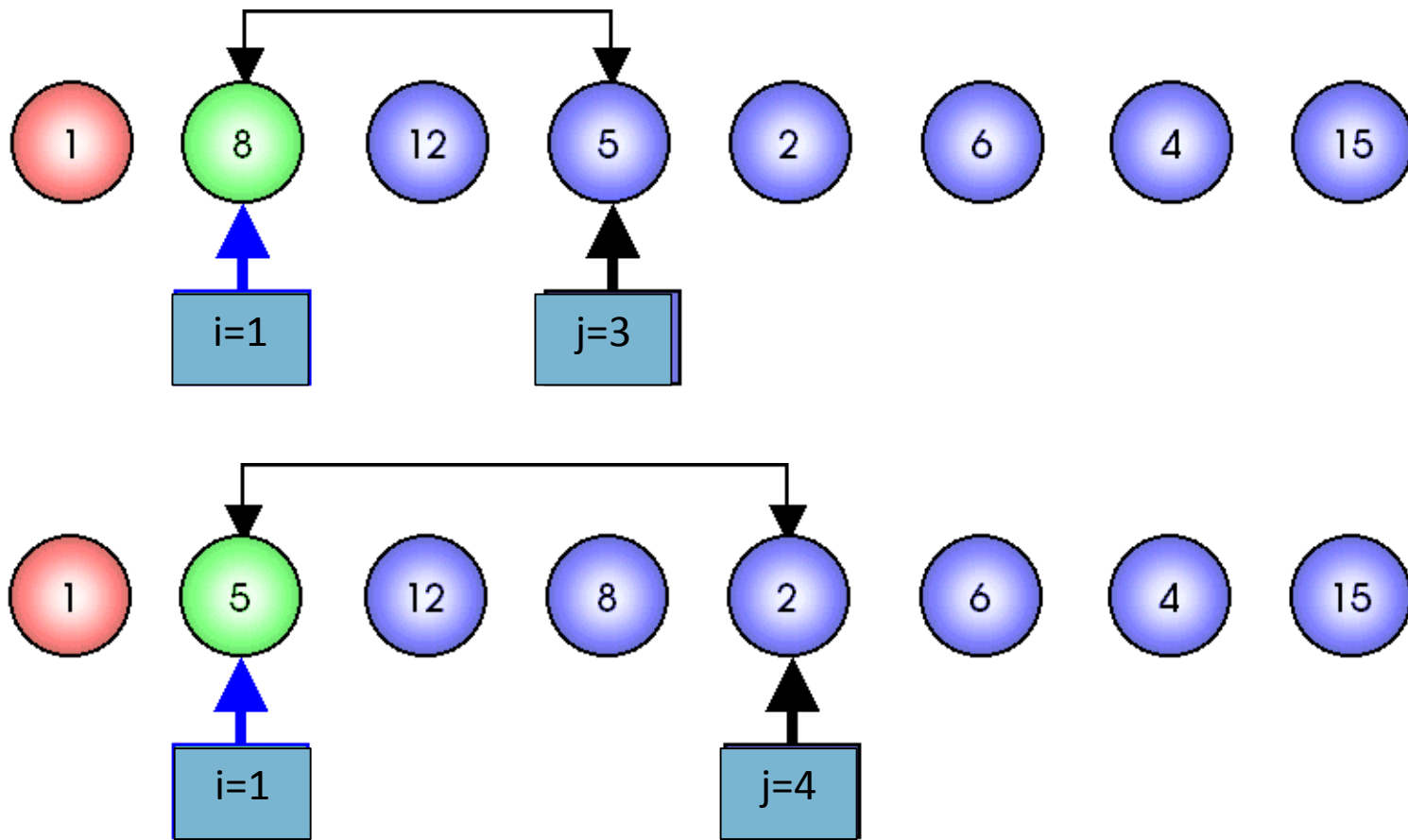
Sắp xếp đổi chỗ trực tiếp (Interchange Sort)

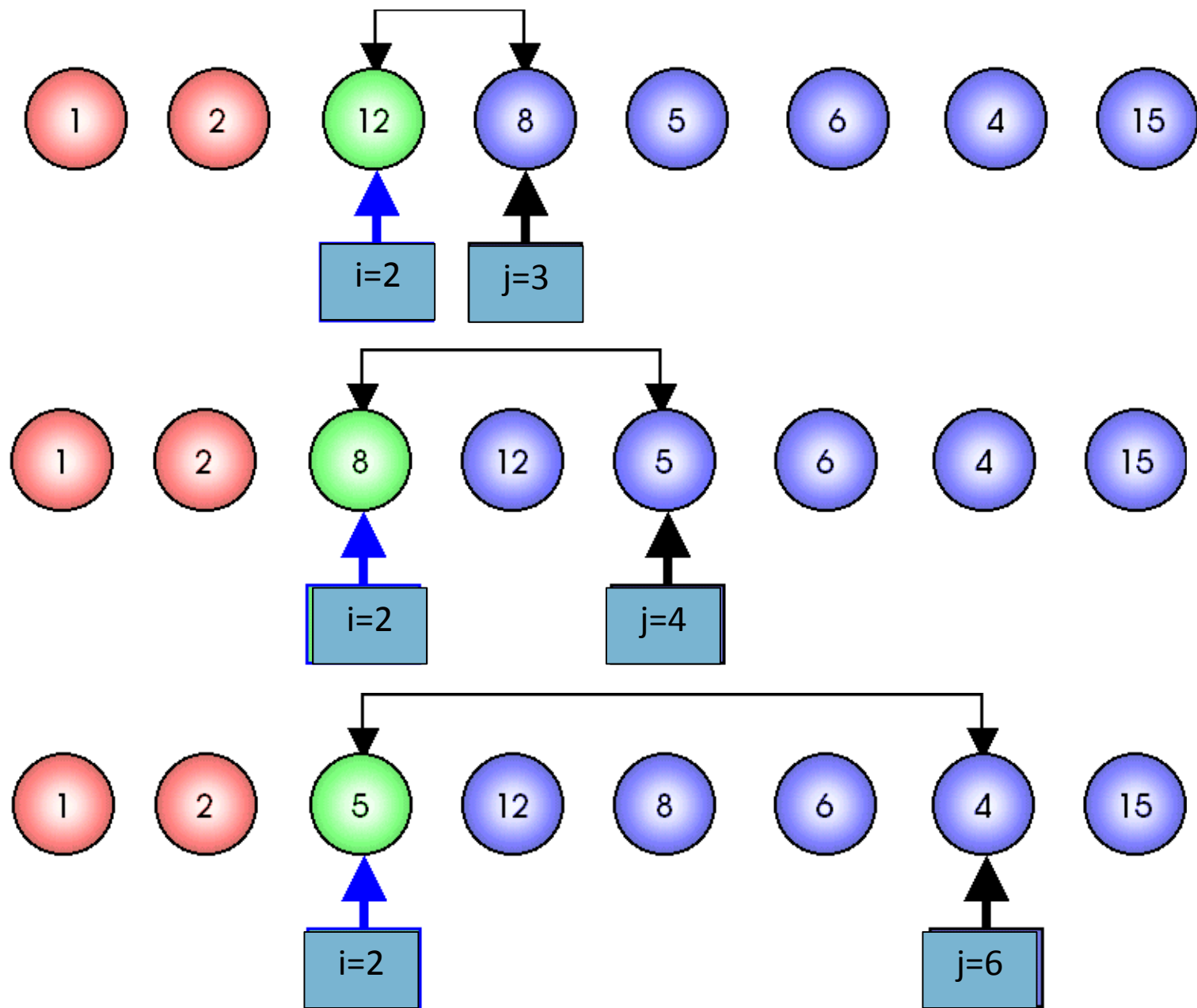
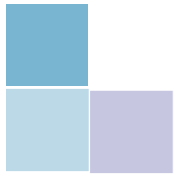
Cho mảng a:

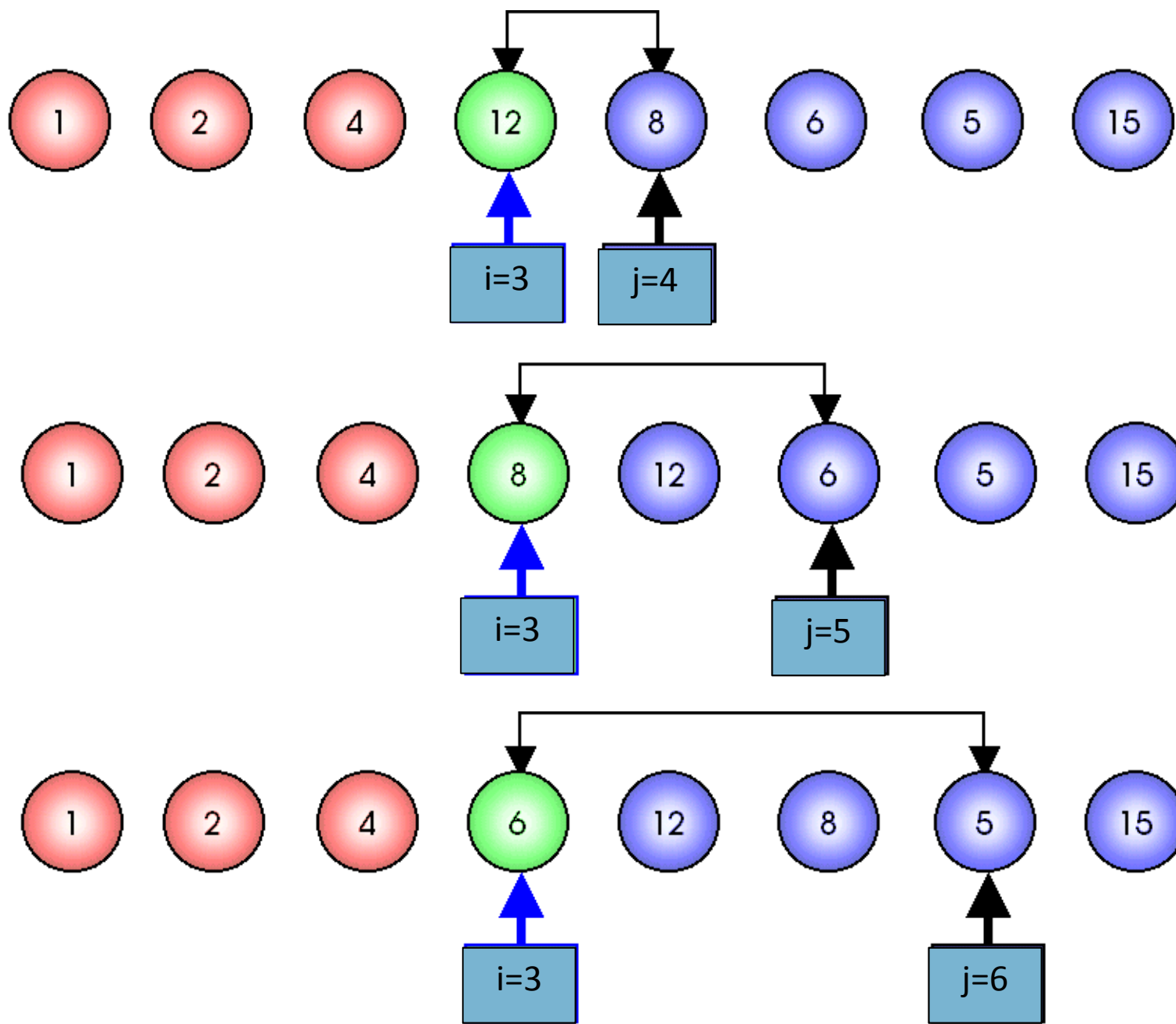
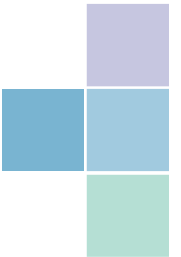
i:	0	1	2	3	4	5	6	7
a[i]:	12	2	8	5	1	6	4	15

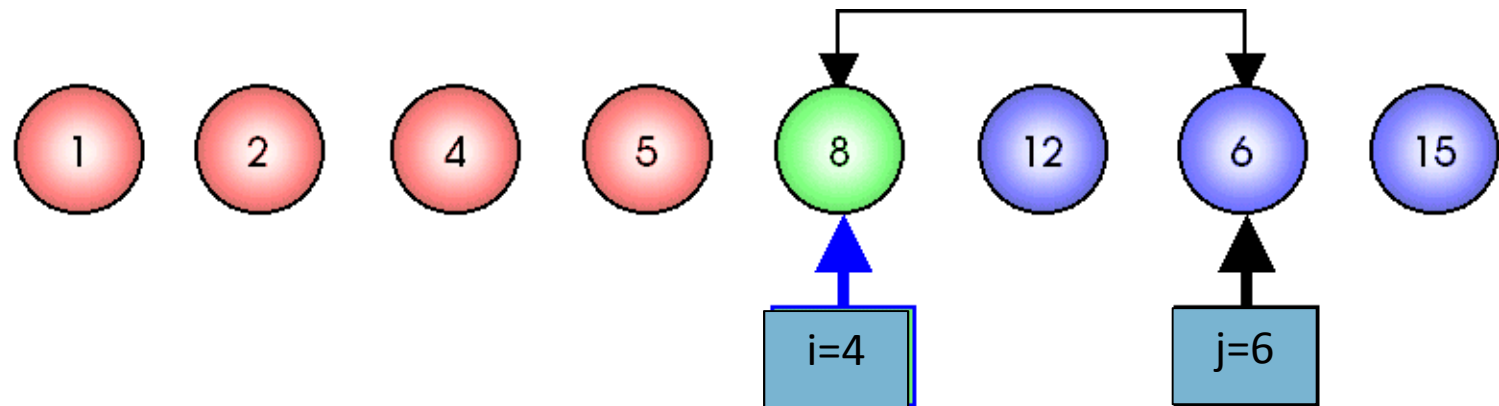
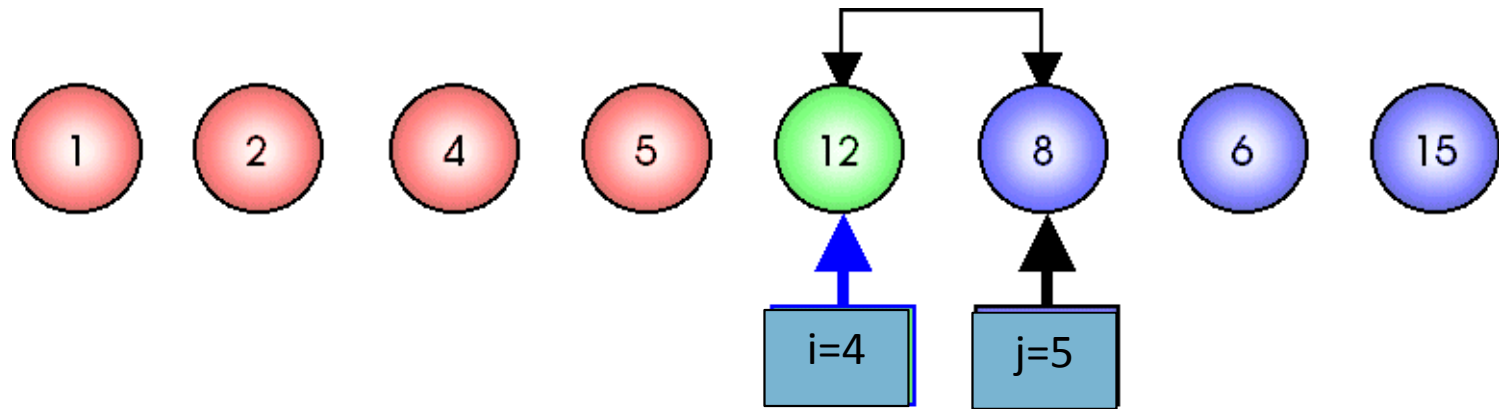
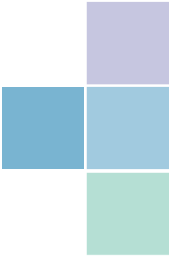
Sắp tăng a.

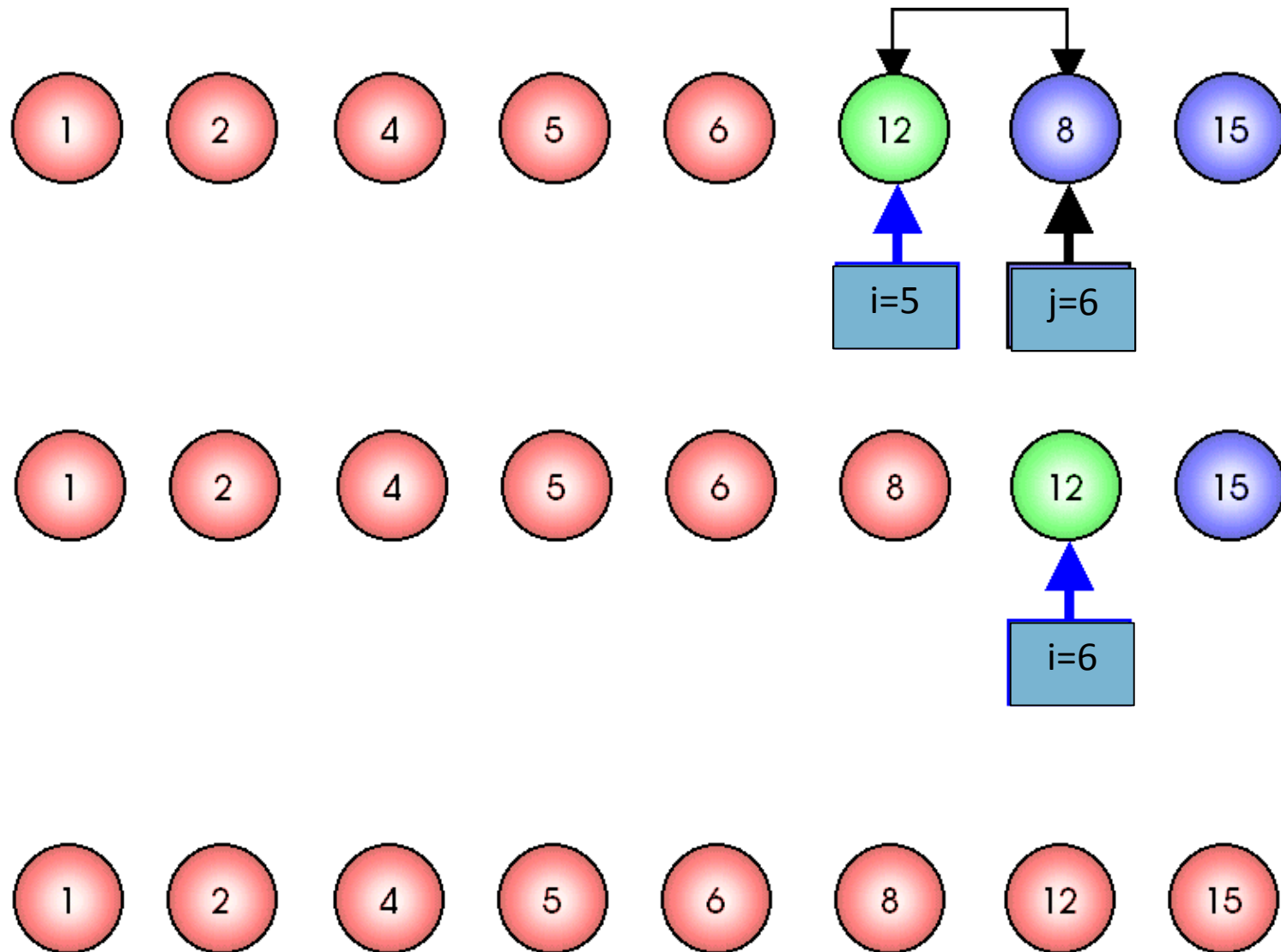
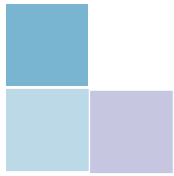












Sắp xếp đổi chỗ trực tiếp (Interchange Sort)

❖ Cài đặt:

```
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])
                HoanVi(a[i], a[j]);
}
```

❖ Đánh giá thuật toán

Trường hợp	Số lần so sánh	Số phép gán
Tốt nhất	$\sum_{i=1}^{N-1} (N-i) = \frac{N(N-1)}{2}$	0
Xấu nhất	$\frac{N(N-1)}{2}$	$\frac{N(N-1)}{2}$

Sắp xếp nổi bọt (Bubble Sort)

❖ **Ý tưởng** chính của thuật giải là xuất phát từ cuối (đầu) dãy, đổi chỗ các cặp phần tử kế cận để đưa phần tử nhỏ (lớn) hơn trong cặp phần tử đó về vị trí đúng đầu (cuối) 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.

❖ **Mô tả thuật giải:**

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:

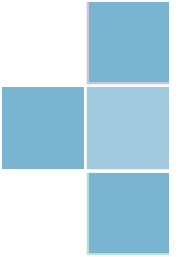
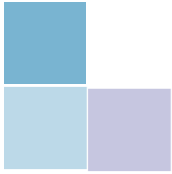
Nếu $a[j] < a[j-1]$: Hoán vị($a[j], a[j-1]$); //xét cặp phần tử kế cận

$j = j-1$;

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

Nếu $i = N-1$: Hết dãy. Dừng

Ngược lại: lặp lại bước 2

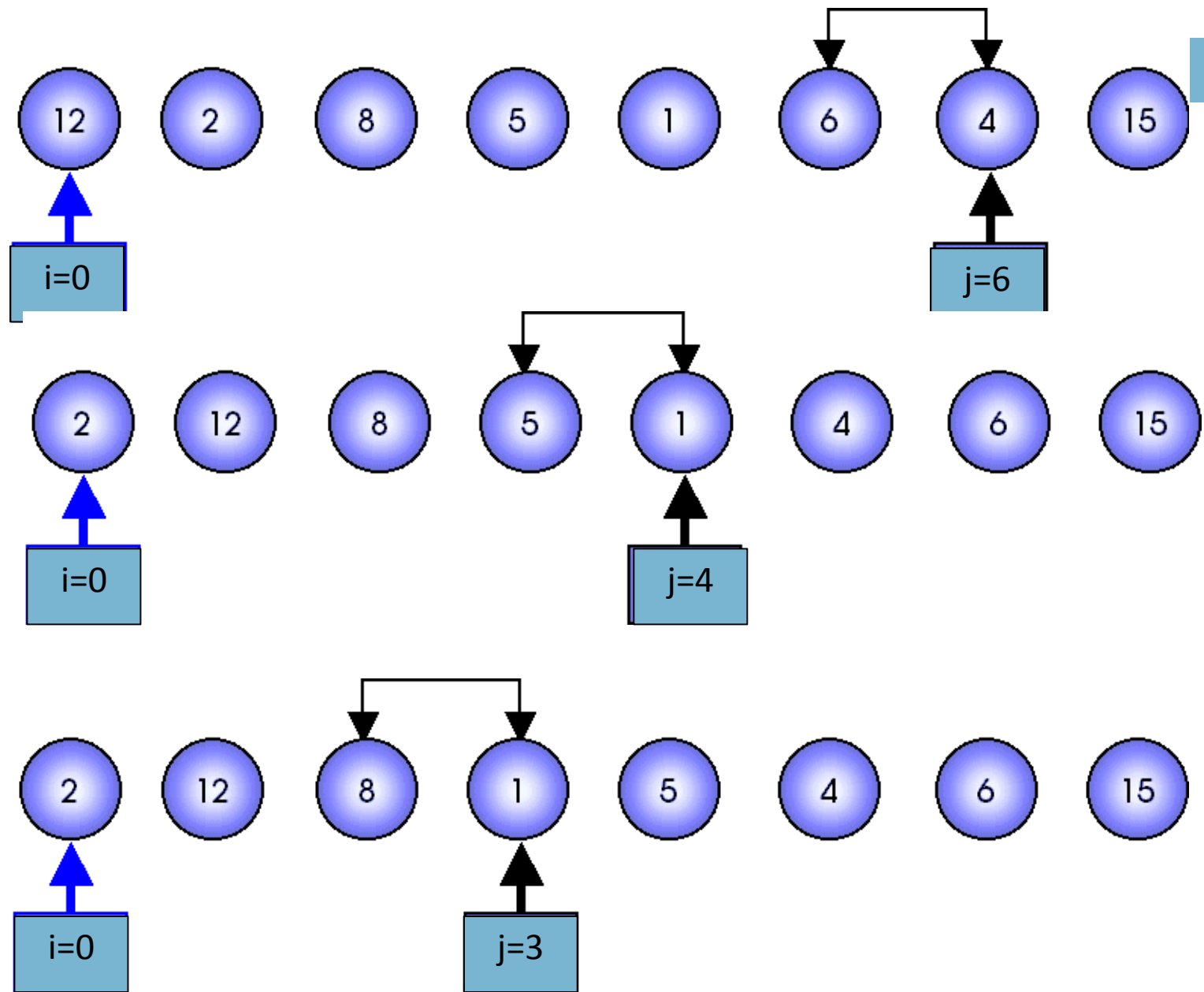


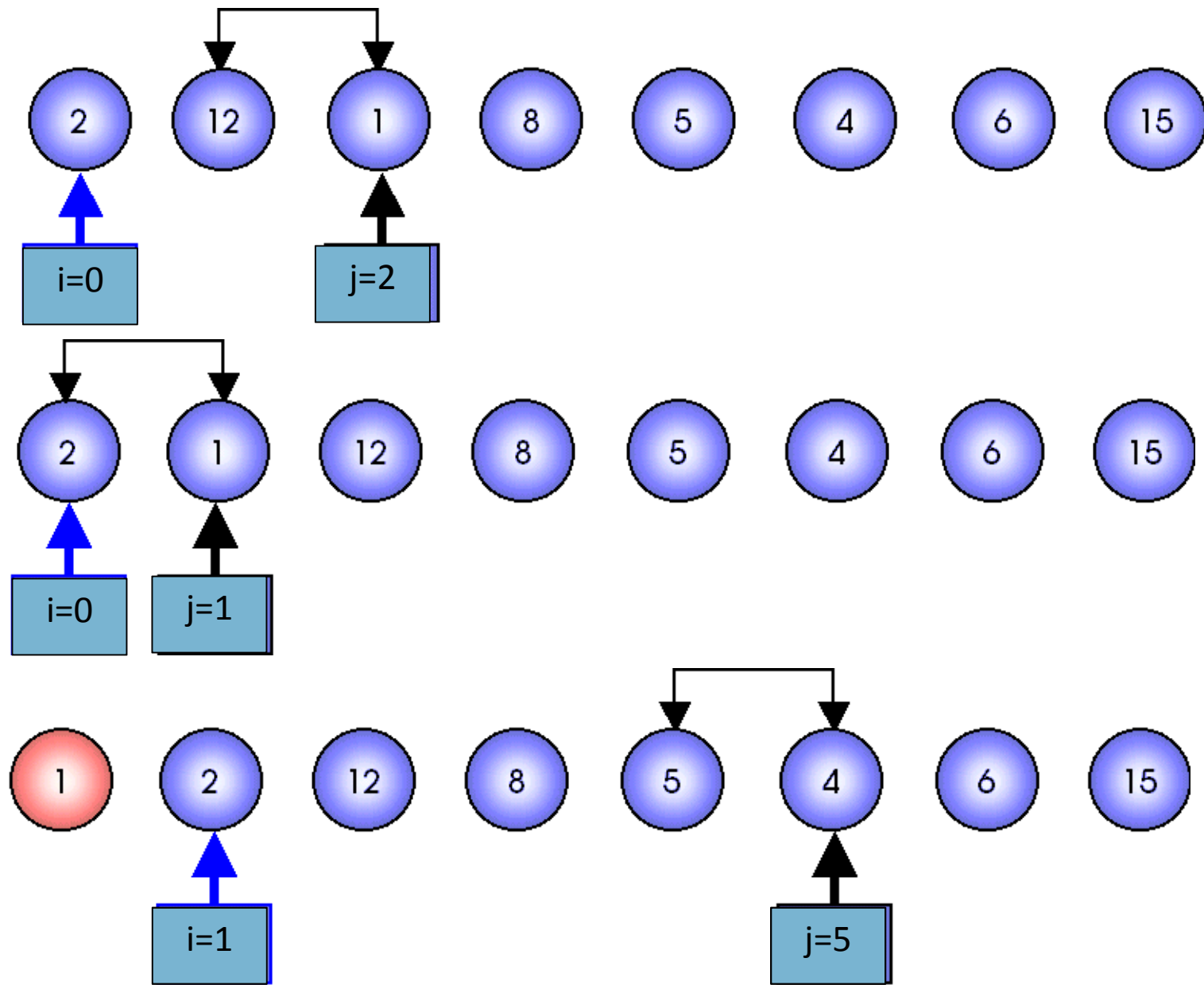
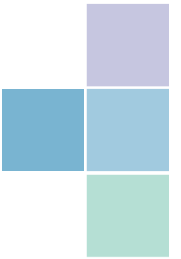
Sắp xếp nổi bọt (Bubble Sort)

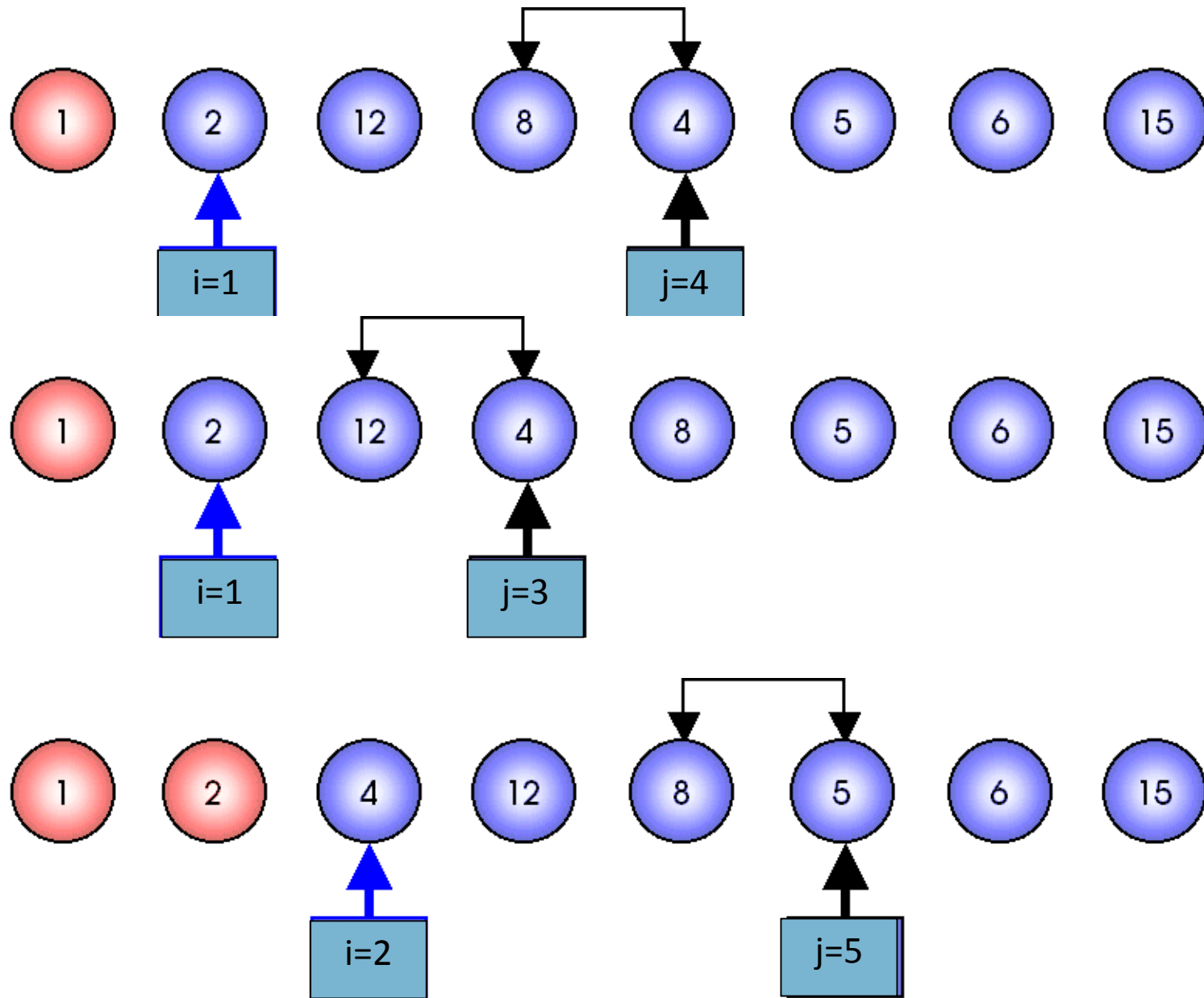
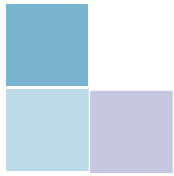
Cho mảng a:

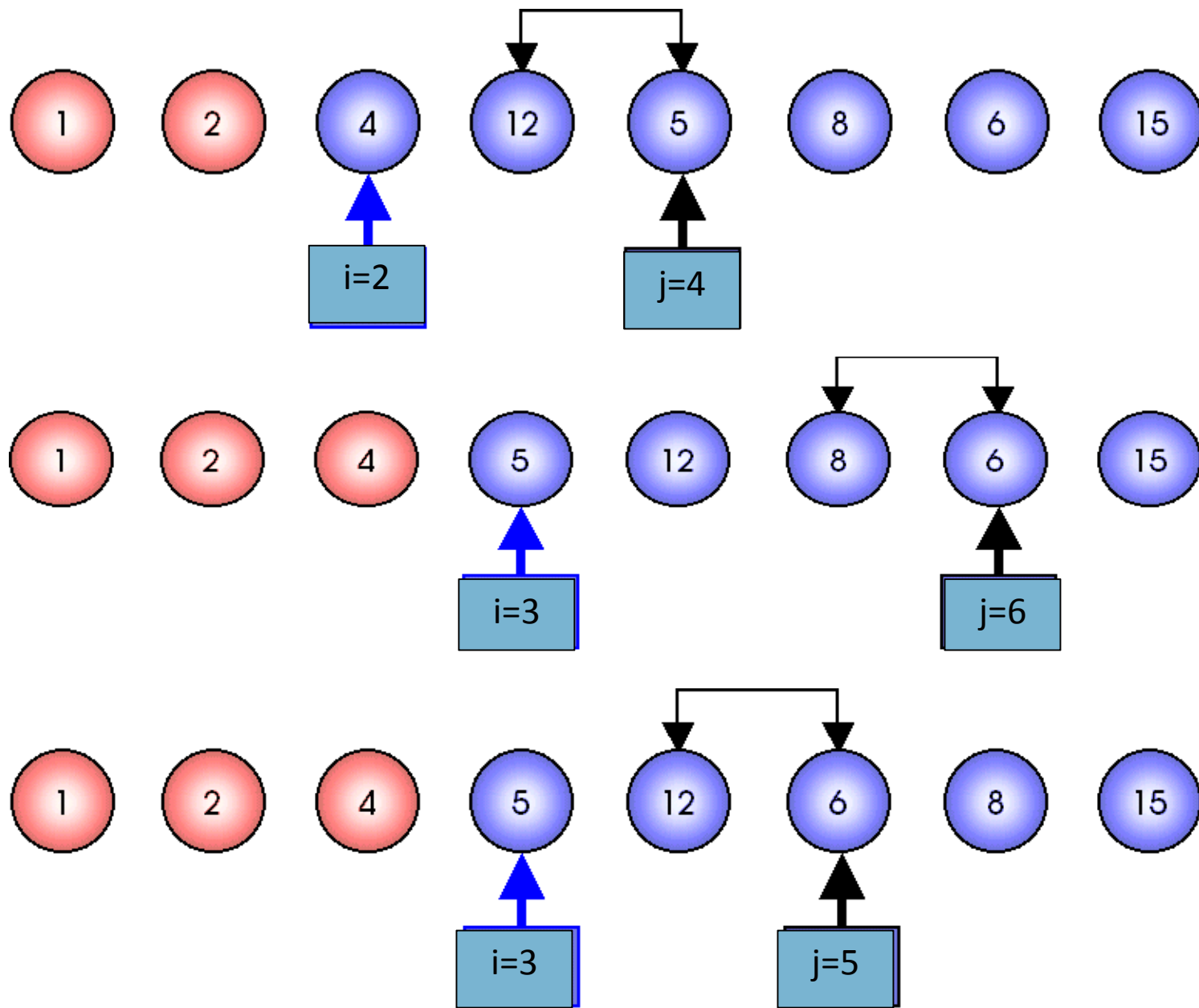
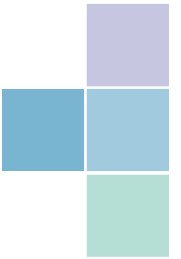
i:	0	1	2	3	4	5	6	7
a[i]:	12	2	8	5	1	6	4	15

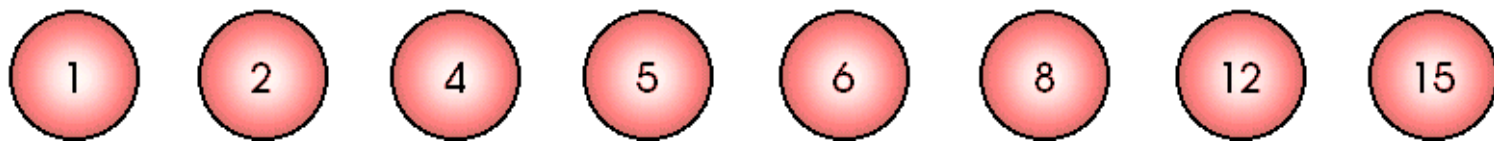
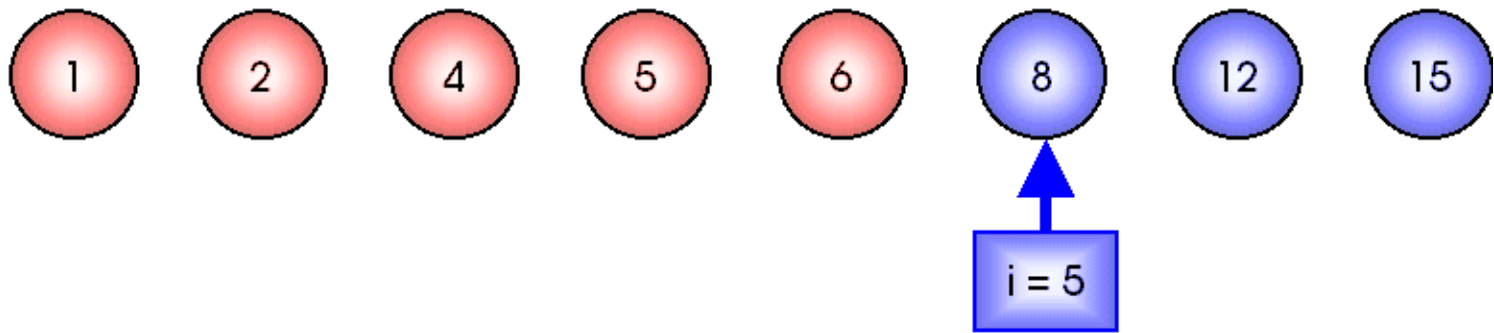
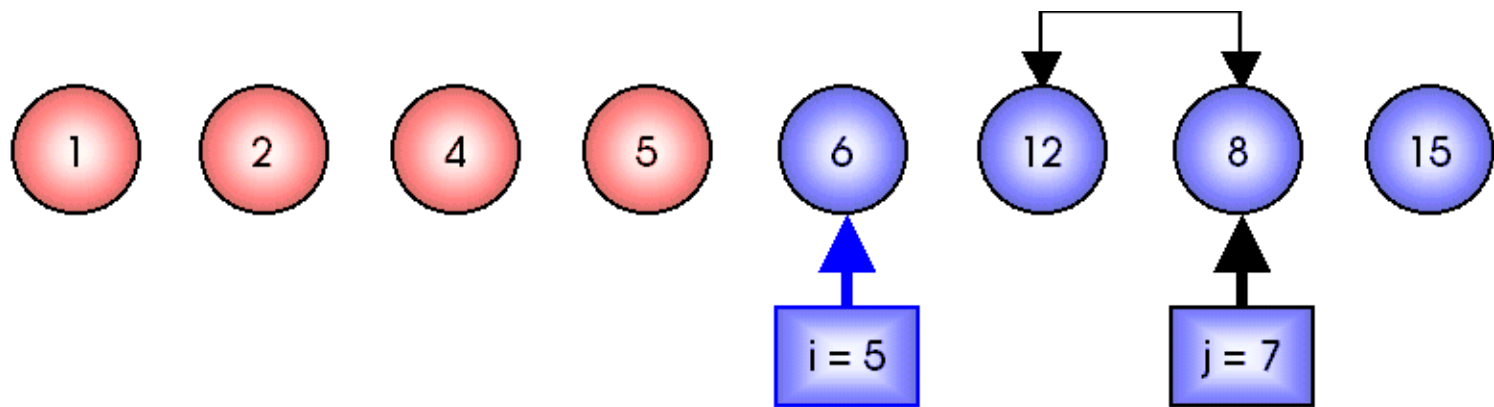
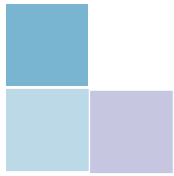
Sắp tăng a.











Sắp xếp nổi bọt (Bubble Sort)

❖ Cài đặt:

```
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])
                HoanVi(a[j], a[j - 1]);
}
```

❖ Đánh giá thuật giải:

Trường hợp	Số lần so sánh	Số phép gán
Tốt nhất	$\sum_{i=1}^{N-1} (N-i) = \frac{N(N-1)}{2}$	0
Xấu nhất	$\frac{N(N-1)}{2}$	$\frac{N(N-1)}{2}$



Thuật giải vun đống (Heap Sort)



❖ **Ý tưởng:** cải tiến thuật giải Chọn trực tiếp

- Khi tìm phần tử nhỏ nhất ở bước i , phương pháp chọn trực tiếp không tận dụng được các thông tin đã có trong các phép so sánh ở bước $i-1$. Thuật giải Heap Sort khắc phục nhược điểm này.
- Khái niệm heap và thuật giải Heapsort do J. Williams đề xuất.

Thuật giải vun đống (Heap Sort)

❖ Định nghĩa heap:

- Giả sử xét trường hợp sắp xếp tăng dần, khi đó Heap được định nghĩa là một dãy các phần tử a_1, a_2, \dots, a_r thoả các quan hệ sau với mọi $i \in [1, r]$.
 - $a_i \geq a_{2i+1}$
 - $a_i \geq a_{2i+2}$ $\{(a_i, a_{2i+1}), (a_i, a_{2i+2})$ là các cặp phần tử liên đới }

❖ Heap có các tính chất sau:

- Tính chất 1: Nếu a_1, a_2, \dots, a_r là một heap thì khi cắt bỏ một số phần tử ở hai đầu của heap, dãy còn lại vẫn là một heap.
- Tính chất 2: Nếu a_0, a_2, \dots, a_{N-1} là một heap thì phần tử a_0 (đầu heap) luôn là phần tử lớn nhất trong heap.
- Tính chất 3: Mọi dãy a_1, a_2, \dots, a_r với $2l > r$ là một heap.
- Tính chất 4: Nếu dãy a_0, \dots, a_{N-1} là một heap thì ta có thể mô tả "liên tiếp" những phần tử của dãy này trên một cây nhị phân có tính chất:
 - Con trái (nếu có) của a_i là $a_{2i+1} \leq a_i$
 - Và con phải (nếu có) của a_i là $a_{2i+2} \leq a_i$



Thuật giải vun đống (Heap Sort)



❖ **Mô tả thuật giải:** Thuật giải Heapsort trải qua 2 giai đoạn:

- 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 nhỏ nhất về vị trí đúng ở cuối dãy: $r = N-1$;

Hoán vị (a_0, a_r);

Bước 2:

Loại bỏ phần tử nhỏ nhất ra khỏi heap: $r = r-1$;

Hiệu chỉnh phần còn lại của dãy từ a_0, a_2, \dots, a_r thành một heap.

Bước 3:

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

Ngược lại: Dừng



Thuật giải vun đống (Heap Sort)



❖ **Cài đặt:** để cài đặt thuật giải Heapsort cần xây dựng các thủ tục phụ trợ:

- Thủ tục hiệu chỉnh dãy a_1, a_{1+1}, \dots, a_r thành heap:
void Shift (int a[], int l, int r)
- Thủ tục hiệu chỉnh dãy a_0, a_2, \dots, a_{N-1} thành heap:
void CreateHeap(int a[], int N)



Thủ tục hiệu chỉnh dãy a_1, a_{l+1}, \dots, a_r thành heap



- ❖ Giả sử có dãy $a_1, a_{l+1} \dots a_r$, trong đó đoạn $a_{l+1} \dots a_r$, đã là một heap, ta cần xây dựng hàm hiệu chỉnh $a_1, a_{l+1} \dots a_r$ thành heap.
- ❖ Để làm điều này, ta lần lượt xét quan hệ của một phần tử a_i nào đó với các phần tử liên đới của nó trong dãy là a_{2i+1} và a_{2i+2} , nếu vi phạm điều kiện quan hệ của heap, thì đổi chỗ a_i với phần tử liên đới thích hợp của nó.
- ❖ Lưu ý việc đổi chỗ này có thể gây phản ứng dây chuyền.

Thủ tục hiệu chỉnh dãy a_l, a_{l+1}, \dots, a_r thành heap

```
void Shift(int a[], int l, int r)
{
    int x, i, j;
    i = l; j = 2 * i + 1; //(ai, aj), (ai, aj+1) là các phần tử liên đới
    x = a[i];
    while (j <= r)
    {
        if (j < r) //nếu có hai phần tử liên đới
            if (a[j] < a[j + 1])// xác định phần tử liên đới lớn nhất
                j = j + 1;
        if (a[j] <= x)
            return;//thỏa quan hệ liên đới, dừng
        else
        {
            a[i] = a[j];
            i = j; //xét tiếp khả năng hiệu chỉnh lan truyền
            j = 2 * i + 1;
            a[i] = x;
        }
    }
}
```

Hiệu chỉnh dãy $a_0, a_1 \dots a_{N-1}$ thành heap:

- ❖ Cho một dãy bất kỳ a_0, a_1, \dots, a_{N-1} , theo tính chất 3, ta có dãy $a_{(N-1)/2+1}, \dots, a_{N-1}$ đã là một heap. Ghép thêm phần tử $a_{(N-1)/2}$ vào bên trái heap hiện hành và hiệu chỉnh lại dãy $a_{(N-1)/2}, a_{(N-1)/2+1}, \dots, a_{N-1}$ thành heap.

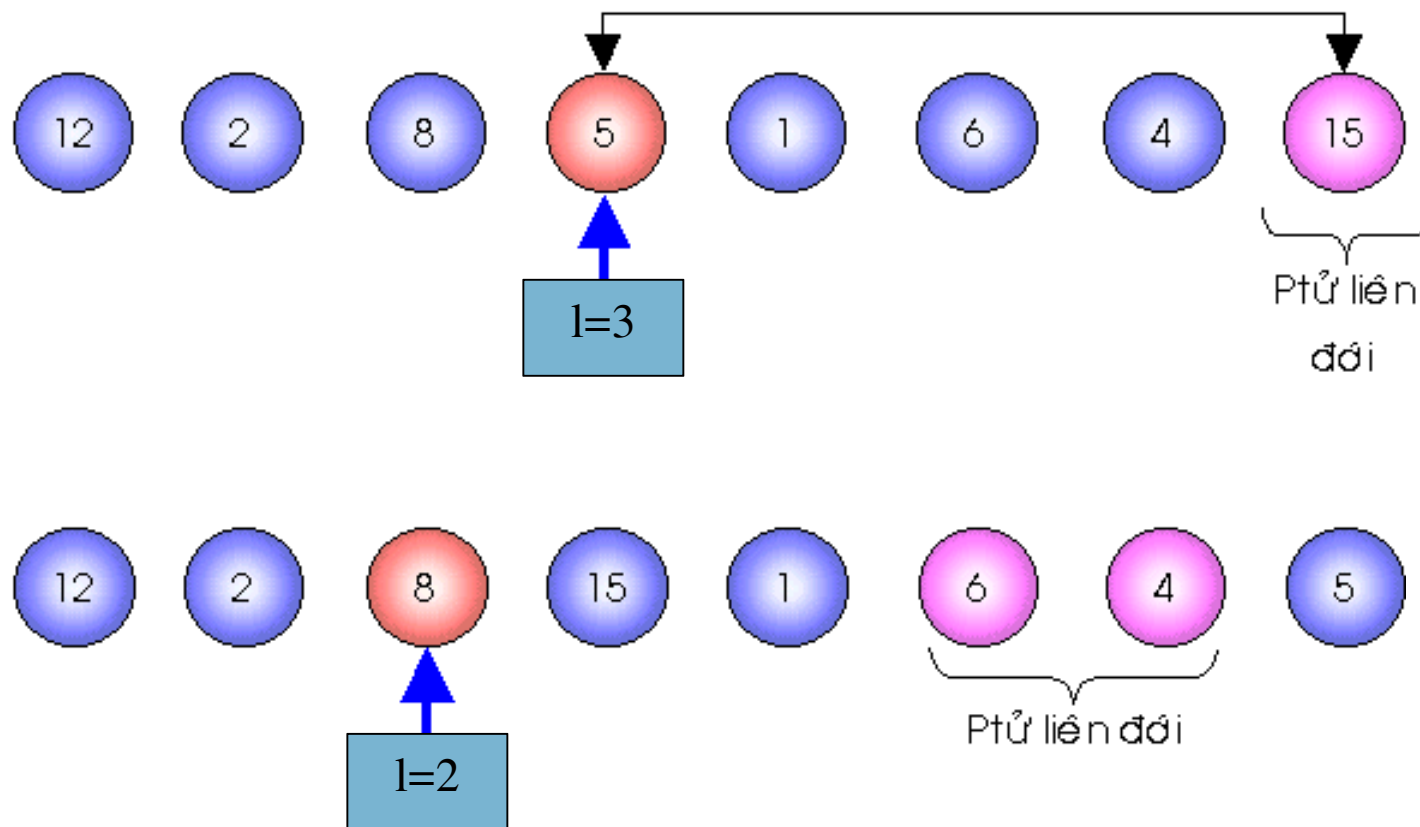
```
void CreateHeap(int a[], int n)
{
    int l;
    l = (n - 1) / 2;    // a[l] là phần tử ghép thêm
    while (l >= 0)
    {
        Shift(a, l, n - 1);
        l = l - 1;
    }
}
```

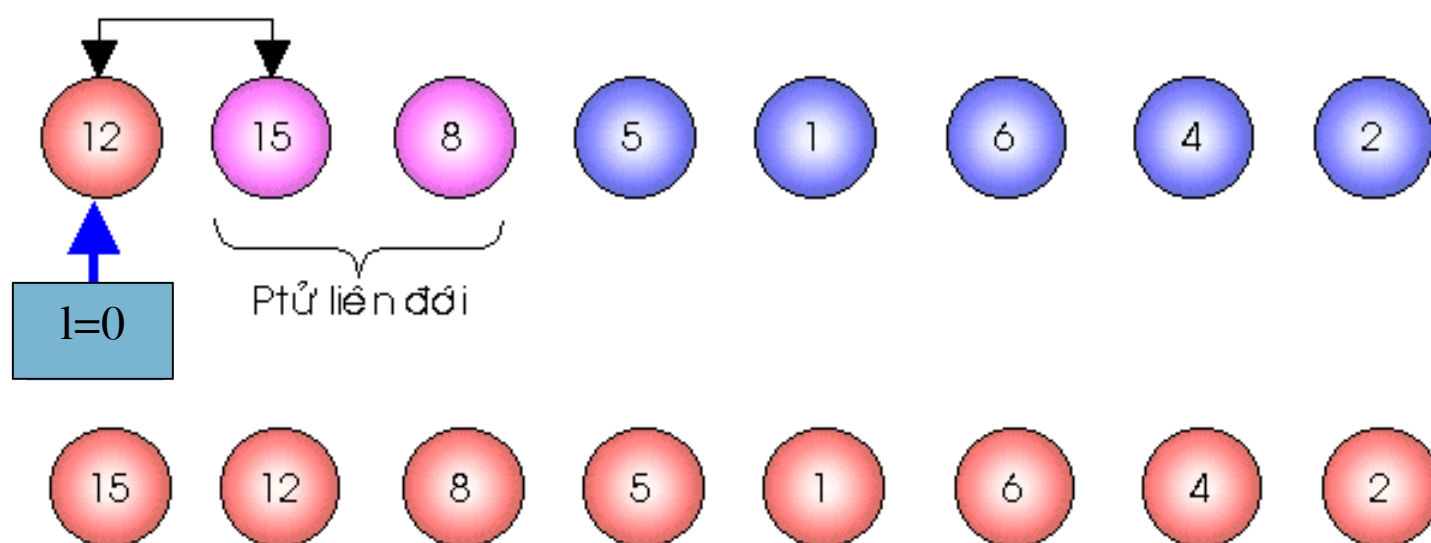
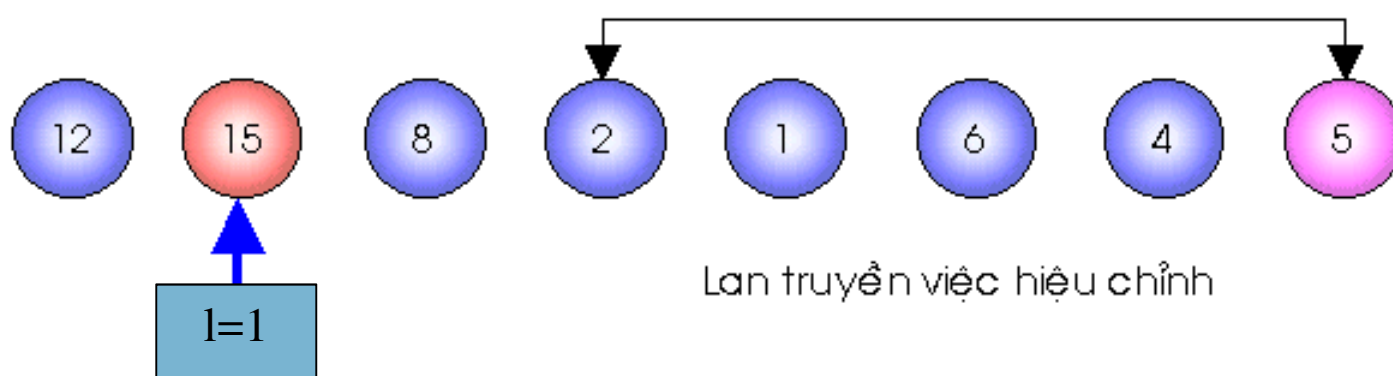
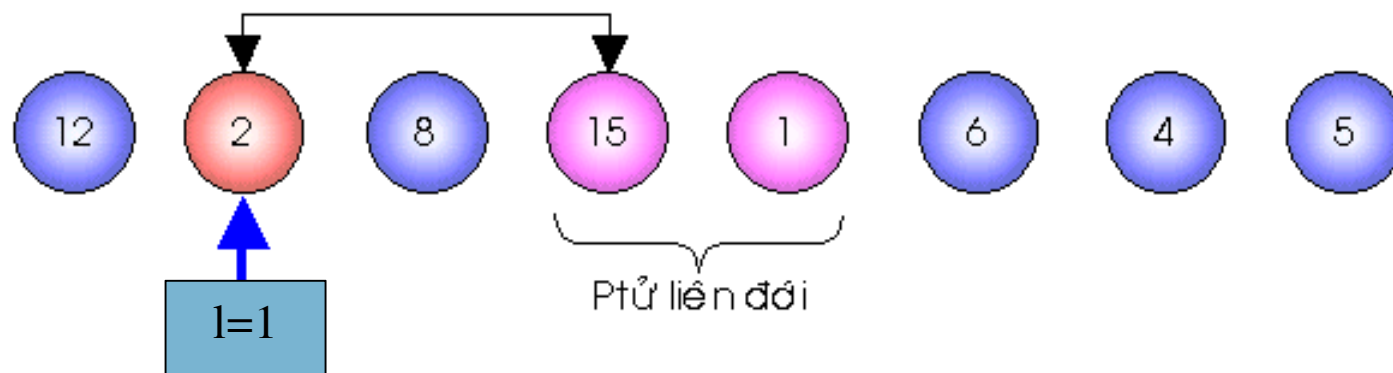
Thủ tục sắp xếp HeapSort

```
void HeapSort(int a[], int n)
{
    int r;
    CreateHeap(a, n);
    r = n - 1; // r là vị trí đúng cho phần tử nhỏ nhất
    while (r > 0)
    {
        HoanVi(a[0], a[r]);
        r = r - 1;
        Shift(a, 0, r);
    }
}
```

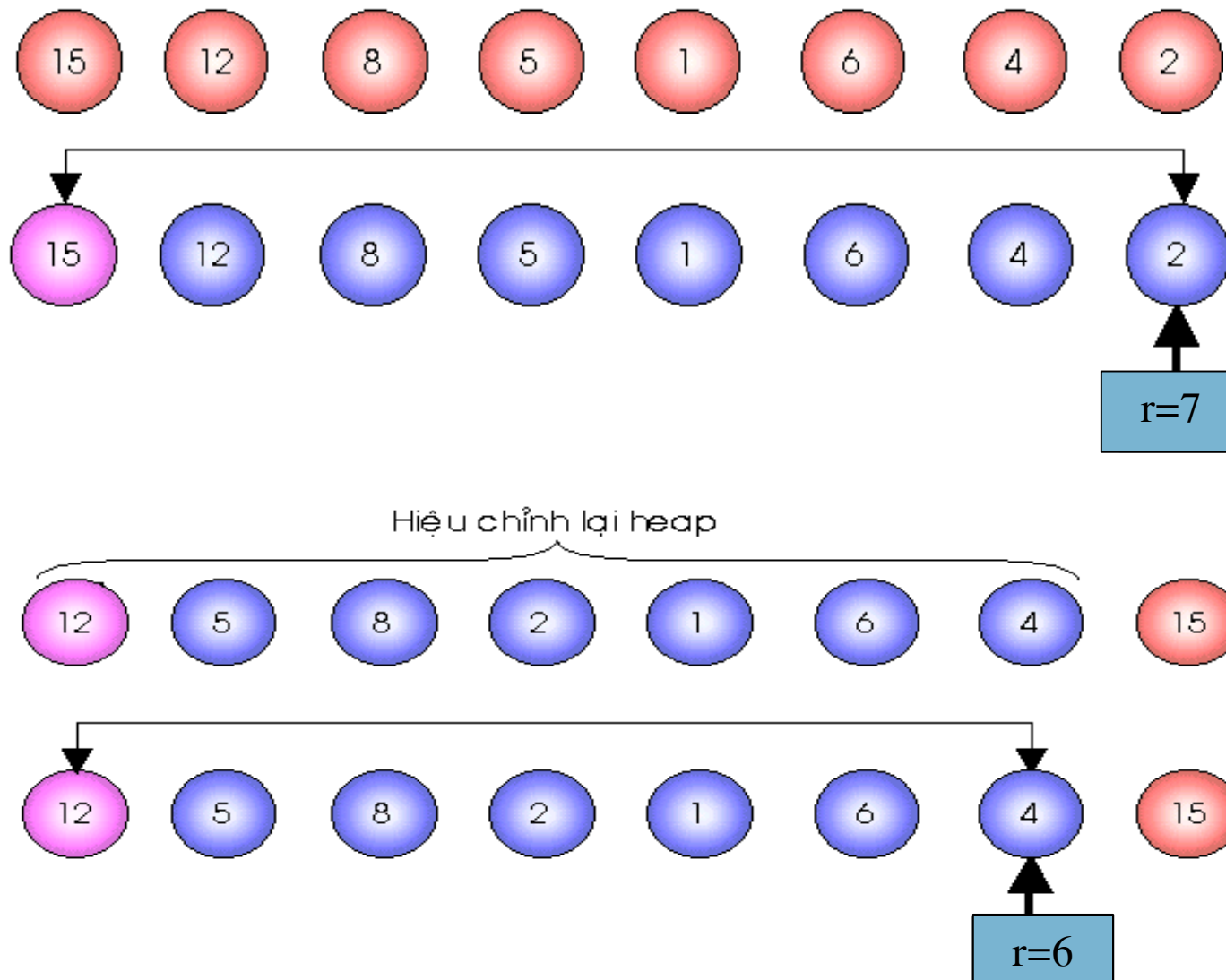
- ❖ Việc đánh giá thuật giải Heapsort rất phức tạp, nhưng đã chứng minh được trong trường hợp xấu nhất độ phức tạp là **$O(n \log_2 n)$** .

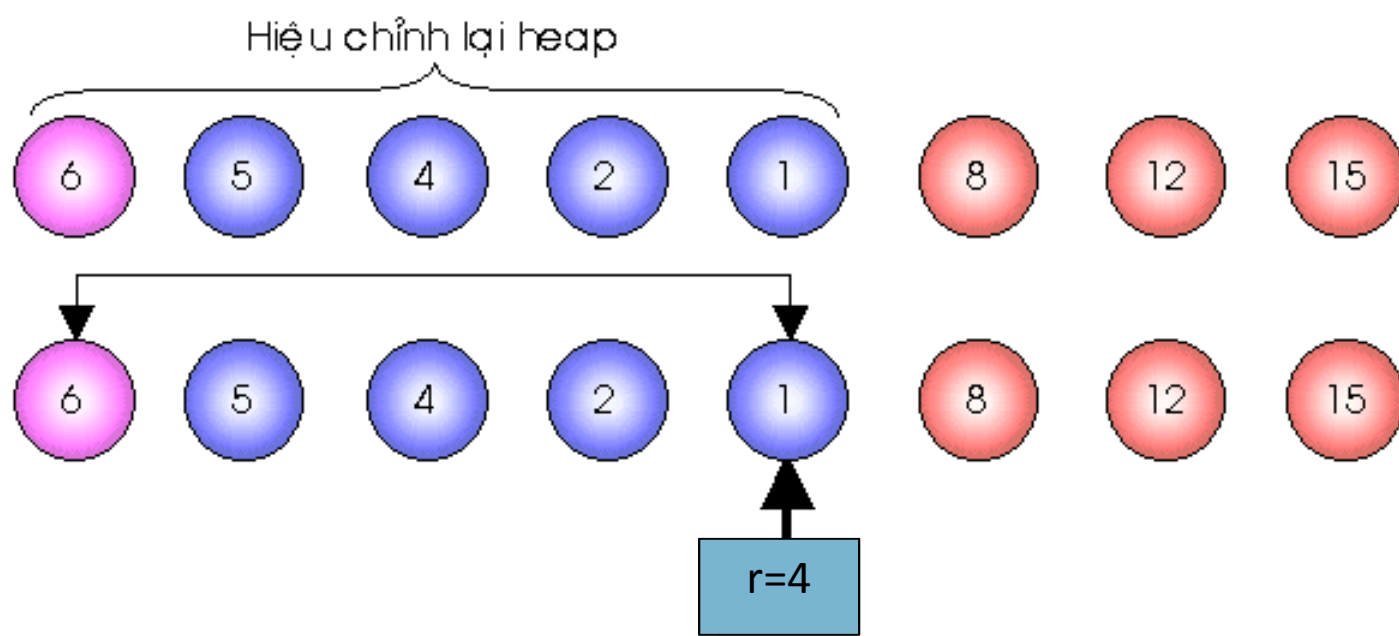
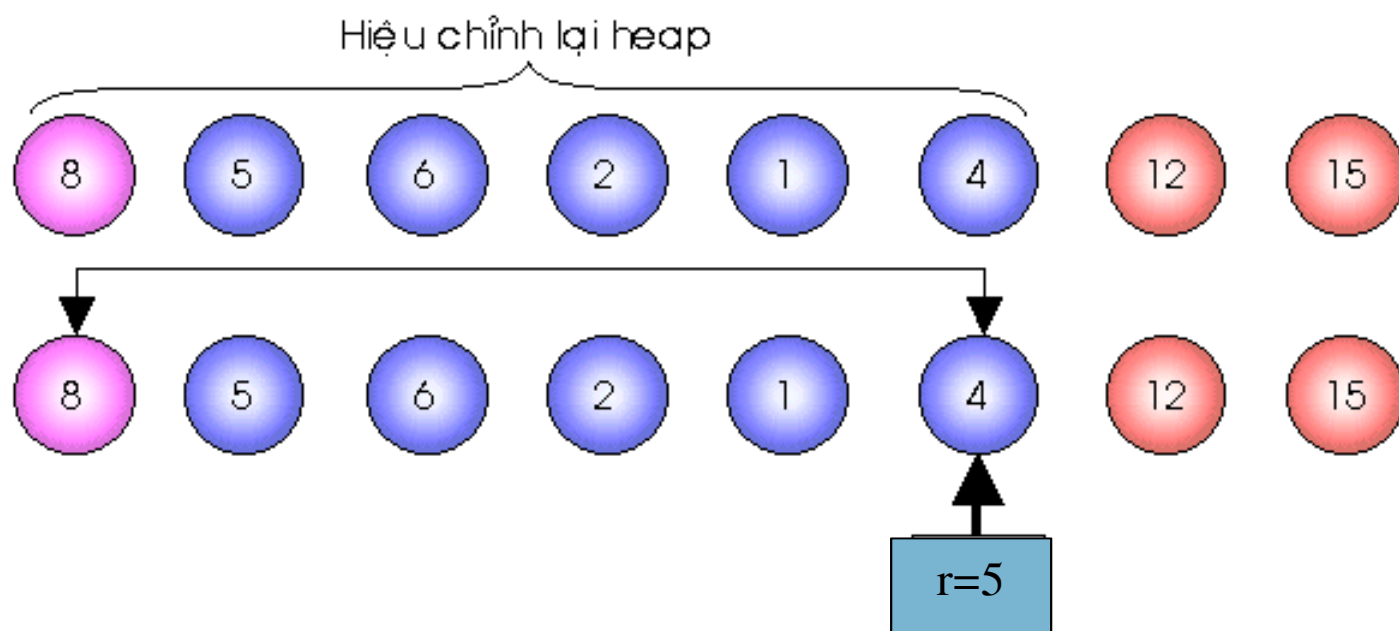
Thủ tục hiệu chỉnh dãy a_l, a_{l+1}, \dots, a_r thành heap

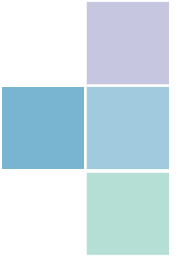




Sắp xếp dãy số dựa trên heap

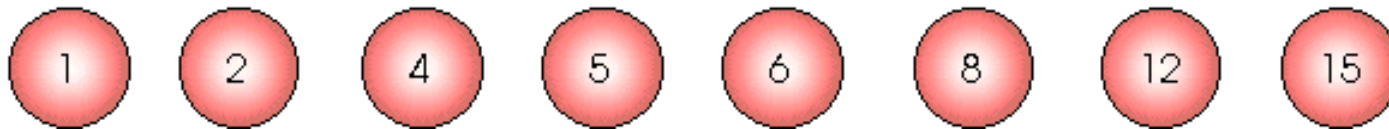






Sắp xếp dãy số dựa trên heap

❖ Thực hiện tương tự cho $r=5,4,3,2$ ta được:





Sắp xếp nhanh (Quick Sort)



❖ **Ý tưởng:** để sắp xếp dãy a_0, a_1, \dots, a_{N-1} thuật giải QuickSort dựa trên việc phân hoạch dãy ban đầu thành hai phần:

- Dãy con 1: Gồm các phần tử $a_0.. a_j$ có giá trị không lớn hơn x
- Dãy con 2: Gồm các phần tử $a_i.. a_{N-1}$ có giá trị không nhỏ hơn x với x là giá trị của một phần tử tùy ý trong dãy ban đầu.
- Sau khi thực hiện phân hoạch, dãy ban đầu được phân thành 3 phần:
 - $a_k < x$, với $k = 0..j$
 - $a_k = x$, với $k = j+1..i-1$
 - $a_k > x$, với $k = i..N-1$

trong đó dãy con thứ 2 đã có thứ tự, nếu các dãy con 1 và 3 chỉ có 1 phần tử thì chúng cũng đã có thứ tự, khi đó dãy ban đầu đã được sắp. Ngược lại, nếu các dãy con 1 và 3 có nhiều hơn 1 phần tử thì dãy ban đầu chỉ có thứ tự khi các dãy con 1, 3 được sắp. Để sắp xếp dãy con 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.

Sắp xếp nhanh (Quick Sort)

❖ Thuật giải phân hoạch dãy a_l, a_{l+1}, \dots, a_r thành 2 dãy con

Bước 1:

Chọn tùy ý một phần tử $a[k]$ trong dãy là giá trị mốc, $l \leq k \leq r$:

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

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 $i < j // a[i] \geq x \geq a[j]$ mà $a[j]$ đứng sau $a[i]$

Hoán vị $(a[i], a[j]);$

Bước 3:

Nếu $i < j$: Lặp lại Bước 2.//chưa xét hết mảng

Nếu $i \geq j$: Dừng



Sắp xếp nhanh (Quick Sort)



❖ Thuật giải sắp xếp phân hoạch

Sắp xếp dãy a_1, a_{1+1}, \dots, a_r

Có thể phát biểu thuật giải sắp xếp QuickSort một cách đệ qui như sau:

- Bước 1: Phân hoạch dãy $a_1 \dots a_r$ thành các dãy con:

- Dãy con 1: $a_0 \dots a_j \leq x$
- Dãy con 2: $a_{j+1} \dots a_{i-1} = x$
- Dãy con 3: $a_i \dots a_r \geq x$

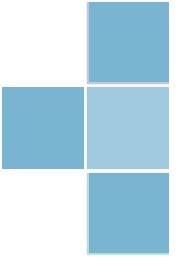
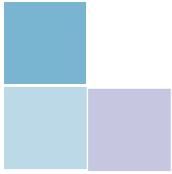
- Bước 2:

Nếu $(1 < j)$ // dãy con 1 có nhiều hơn 1 phần tử

Phân hoạch dãy $a_0 \dots a_j$

Nếu $(i < r)$ // dãy con 3 có nhiều hơn 1 phần tử

Phân hoạch dãy $a_i \dots a_r$



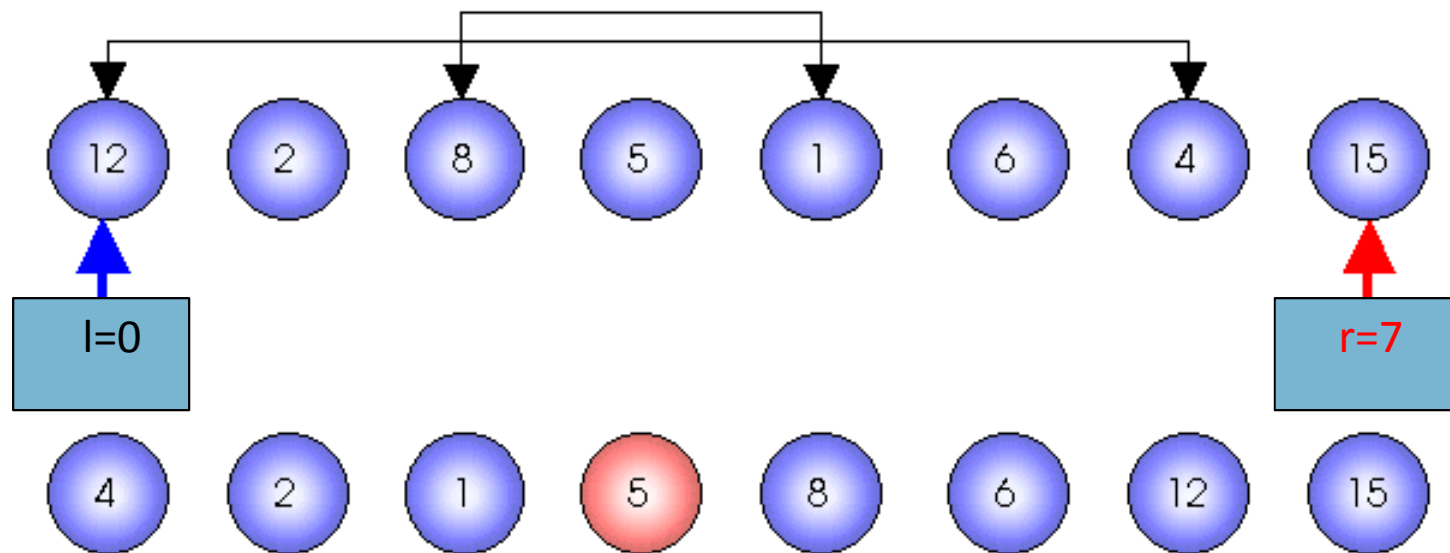
Sắp xếp nhanh (Quick Sort)

Cho mảng a:

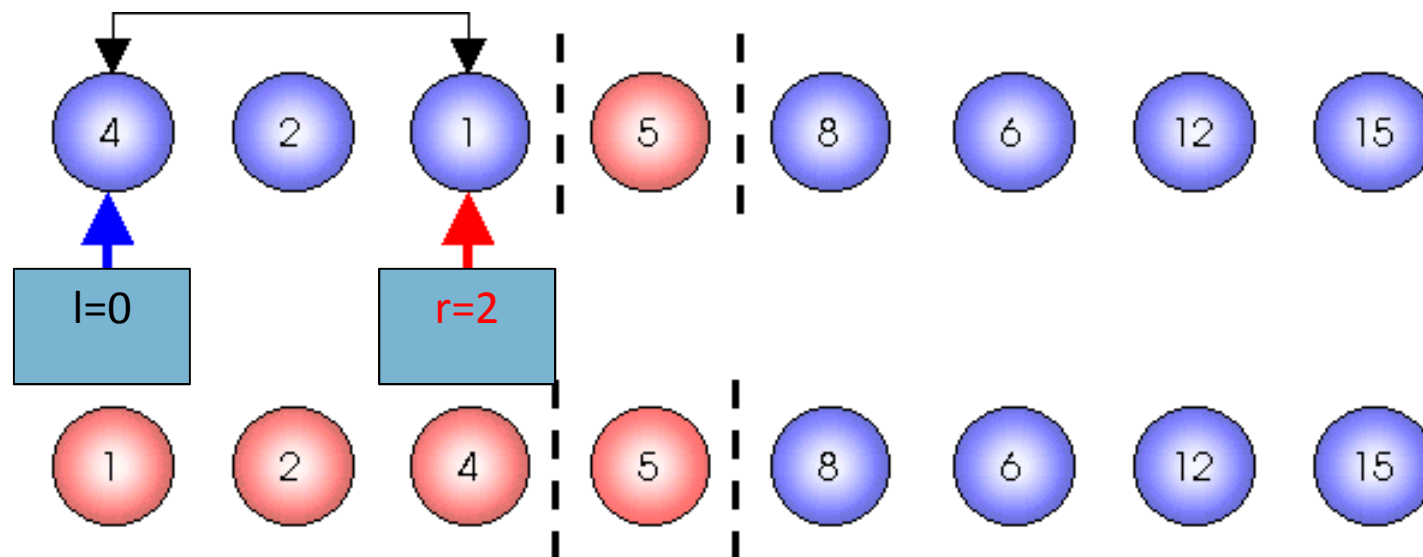
i:	0	1	2	3	4	5	6	7
a[i]:	12	2	8	5	1	6	4	15

Sắp tăng a.

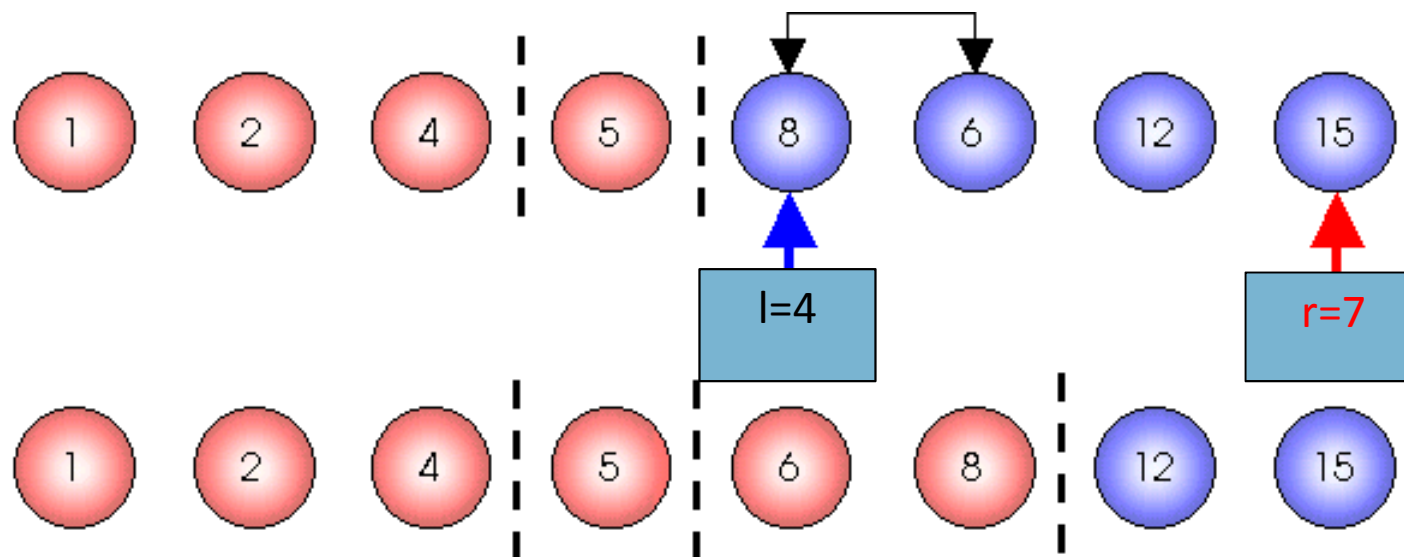
Sắp xếp nhanh (Quick Sort)



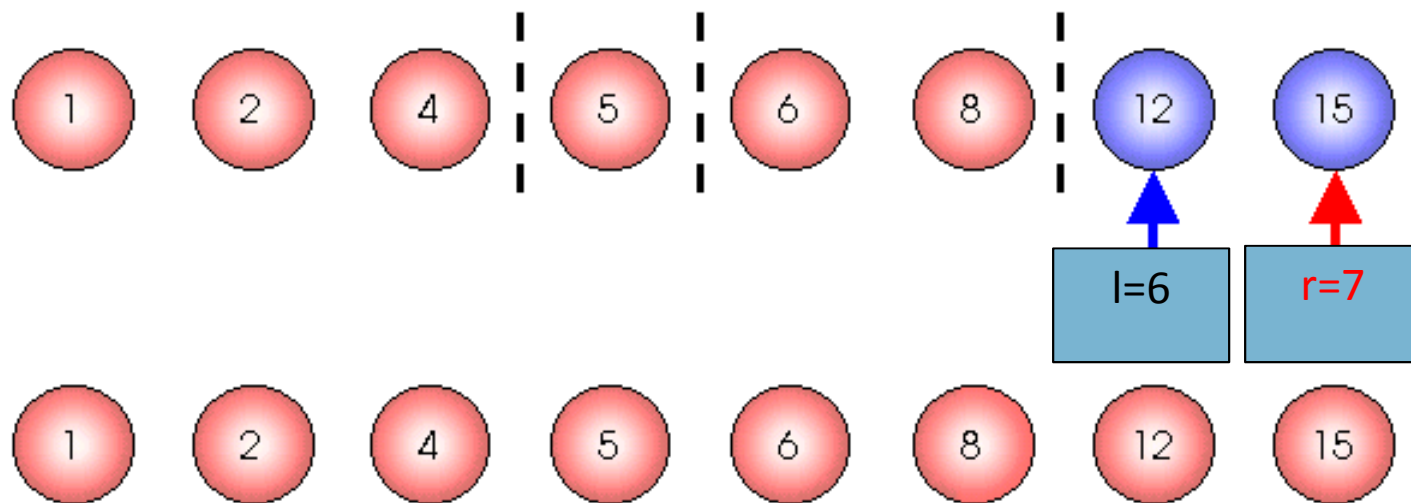
Phần hoạch đoạn $l=0, r=7$: $x = a[3] = 5$



Phần hoạch đoạn $l=0, r=2$: $x = A[1] = 2$



Phân hoạch đoạn $l = 4, r = 7$: $x = a[5] = 6$



Phần hoạch đoạn $l = 6, r = 7$: $x = a[6] = 6$

Sắp xếp nhanh (Quick Sort)

❖ Cài đặt

```
void QuickSort(int a[], int l, int r)
{
    int i, j, x;
    x = a[(l + r) / 2]; //chọn phần tử giữa làm mốc
    i = l; j = r;
    do
    {
        while (a[i] < x) i++;
        while (a[j] > x) j--;
        if (i <= j)
        {
            HoanVi(a[i], a[j]);
            i++; j--;
        }
    } while (i <= j);
    if (l < j) QuickSort(a, l, j);
    if (i < r) QuickSort(a, i, r);
}
```



Sắp xếp trộn (Merge Sort)



❖ Ý tưởng

- Để sắp xếp dãy a_0, a_1, \dots, a_{N-1} , thuật giải Merge Sort dựa trên nhận xét sau:
 - Mỗi dãy a_0, a_1, \dots, a_{N-1} bất kỳ đều có thể coi như 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.



Sắp xếp trộn (Merge Sort)



❖ Thuật giải:

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, \dots, a_{N-1} thành 2 dãy b, c theo nguyên tắc luân phiên từng nhóm k phần tử:

$$b = a_0, \dots, a_{k-1}, a_{2k}, \dots, a_{3k-1}, \dots$$

$$c = a_k, \dots, a_{2k-1}, a_{3k}, \dots, a_{4k-1}, \dots$$

Bước 3: Trộn từng cặp dãy con gồm k phần tử của 2 dãy b, c vào a .

Bước 4:

$$k = k * 2;$$

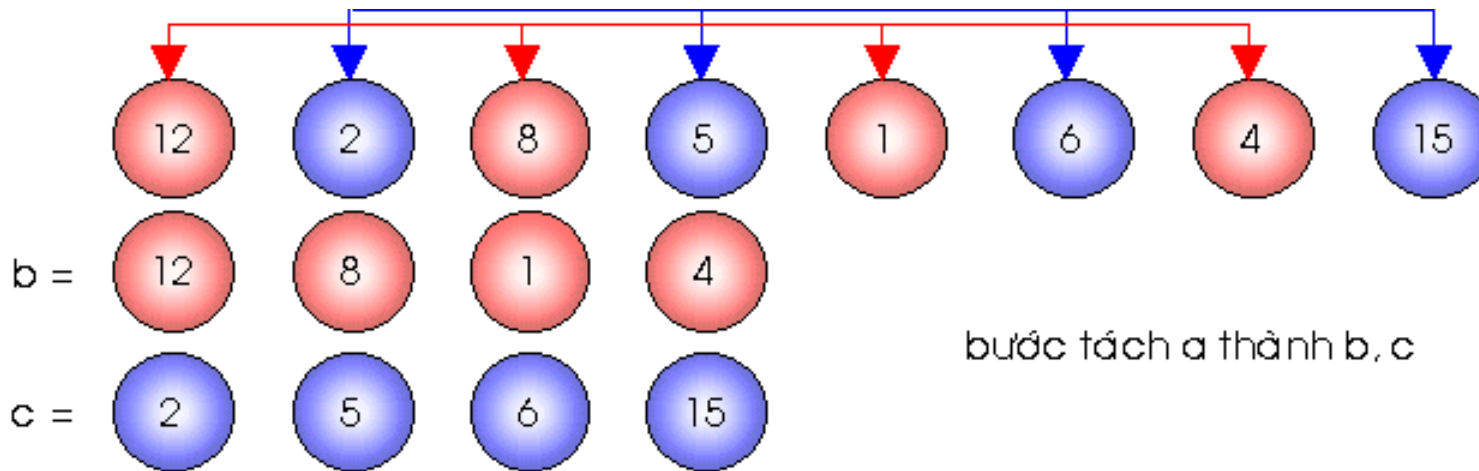
Nếu $k < n$ thì trở lại bước 2.

Ngược lại: Dừng

Sắp xếp trộn (Merge Sort)

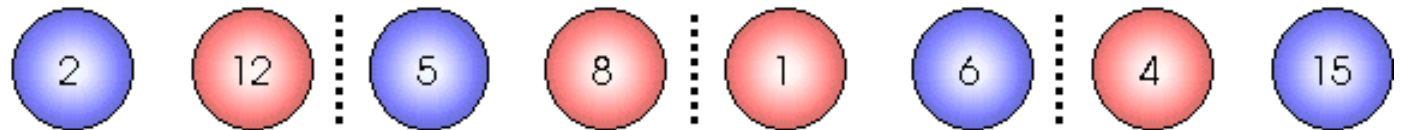
k = 1:

Trộn từng phần tử
của b,c vào a



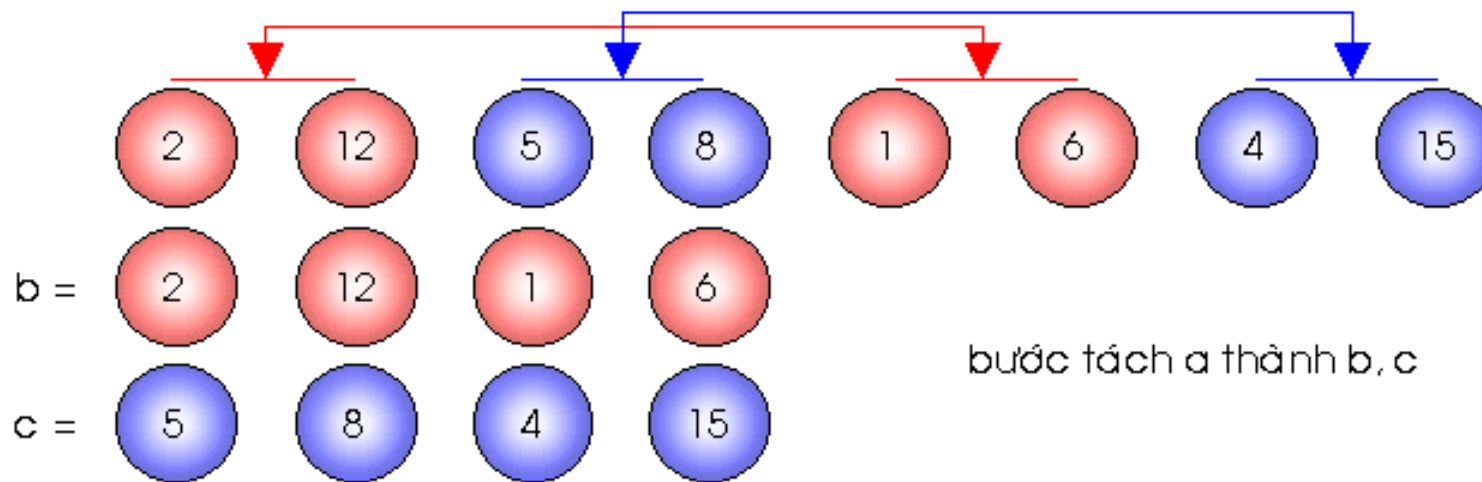
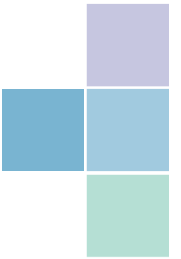
bước tách a thành b, c

Trộn từng phần tử
của b,c vào a



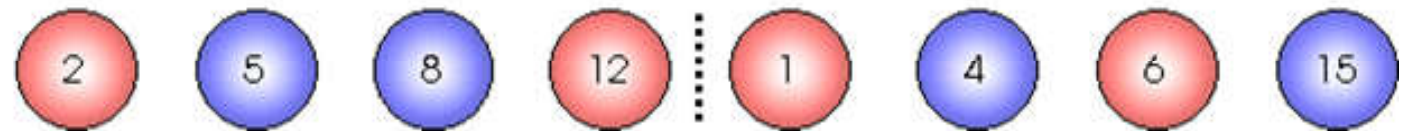


k = 2:



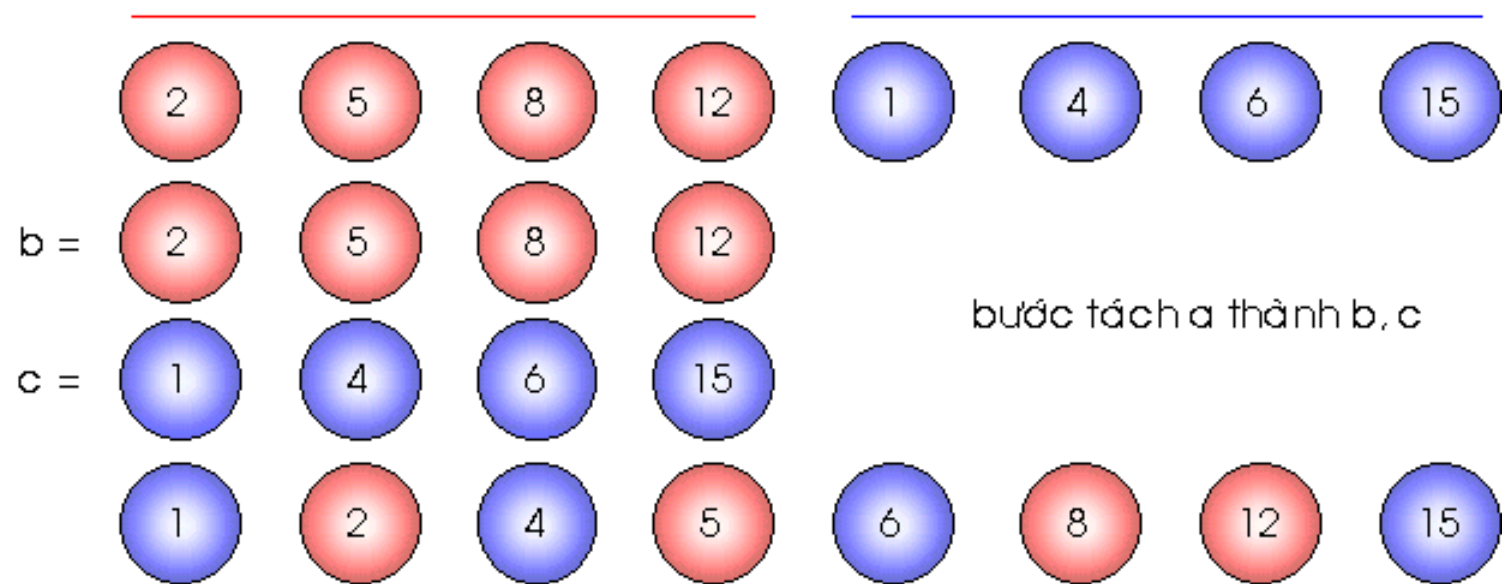
bước tách a thành b, c

Trộn từng cặp
phần tử của b, c
vào a





$k = 4$:



Sắp xếp trộn (Merge Sort)

```
void MergeSort(int a[], int n)
{
    int p, pb, pc; //các chỉ số trên các mảng a, b, c
    int i, k = 1; //độ dài của dãy con khi phân hoạch
    int b[MAX], c[MAX];
    do // tách a thành b và c
    {
        p = pb = pc = 0;
        while (p < n)
        {
            for (i = 0; (p < n) && (i < k); i++)
                b[pb++] = a[p++];
            for (i = 0; (p < n) && (i < k); i++)
                c[pc++] = a[p++];
        }
        Merge(a, pb, pc, k); //trộn b và c thành a
        k *= 2;
    } while (k < n);
}
```

Sắp xếp trộn (Merge Sort)

```
void Merge(int a[], int nb, int nc, int k)
{
    int p, pb, pc, ib, ic, kb, kc;
    int b[MAX], c[MAX];
    p = pb = pc = 0; ib = ic = 0;
    while ((0 < nb) && (0 < nc))
    {
        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;
            }
        }
        else // (ib != kb)
        {
            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;
            }
        }
    }
}
```

- ❖ Đánh giá thuật giải: chi phí thực hiện của thuật giải MergeSort sẽ là $O(N \log_2 N)$.



Sắp xếp theo cơ số (Radix Sort)



❖ Ý tưởng

- Radix Sort là một thuật giải tiếp cận theo một hướng hoàn toàn khác. Nếu như trong các thuật giải 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 nó còn có tên khác là Postman's sort
- Mô phỏng lại qui trình trên, để sắp xếp dãy a_0, a_1, \dots, a_{N-1} , thuật giải Radix Sort thực hiện như sau:
 - Trước tiên, ta có thể giả sử mỗi phần tử a_i trong dãy a_0, a_1, \dots, a_{N-1} 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, huyện, phường xã,...



Sắp xếp theo cơ số (Radix Sort)



❖ Mô tả thuật giải:

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, \dots, B_9 rỗng.

Bước 3:

For($i=0$; $i < N$; $i++$)

Đặt a_i vào lô B_t với $t =$ chữ số thứ k của a_i .

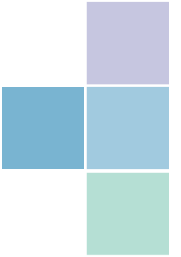
Bước 4: Nối B_0, B_1, \dots, B_9 lại theo đúng trình tự thành a .

Bước 5:

$k = k+1$;

Nếu $k < m$ trở lại bước 2

Ngược lại: dừng.



Sắp xếp theo cơ số (Radix Sort)

i	0	1	2	3	4	5	6	7	8	9	10	11
a[i]	7013	8425	1239	0428	1424	7009	4518	3252	9170	0999	1725	0701

i	0	1	2	3	4	5	6	7	8	9	10	11
a[i]	70 13	84 25	12 39	04 28	14 24	70 09	45 18	32 52	91 70	09 99	17 25	07 01

12	070 <u>1</u>											
11	172 <u>5</u>											
10	099 <u>9</u>											
9	917 <u>0</u>											
8	325 <u>2</u>											
7	451 <u>8</u>											
6	700 <u>9</u>											
5	142 <u>4</u>											
4	042 <u>8</u>											
3	123 <u>9</u>											099 <u>9</u>
2	842 <u>5</u>							172 <u>5</u>			451 <u>8</u>	700 <u>9</u>
1	701 <u>3</u>	917 <u>0</u>	070 <u>1</u>	325 <u>2</u>	701 <u>3</u>	142 <u>4</u>	842 <u>5</u>				042 <u>8</u>	123 <u>9</u>
CS	A	0	1	2	3	4	5	6	7	8	9	

12	09 <u>9</u> 9										
11	70 <u>0</u> 9										
10	12 <u>3</u> 9										
9	45 <u>1</u> 8										
8	04 <u>2</u> 8										
7	17 <u>2</u> 5										
6	84 <u>2</u> 5										
5	14 <u>2</u> 4										
4	70 <u>1</u> 3			04 <u>2</u> 8							
3	32 <u>5</u> 2			17 <u>2</u> 5							
2	07 <u>0</u> 1	70 <u>0</u> 9	45 <u>1</u> 8	84 <u>2</u> 5							
1	91 <u>7</u> 0	07 <u>0</u> 1	70 <u>1</u> 3	14 <u>2</u> 4	12 <u>3</u> 9		32 <u>5</u> 2		91 <u>7</u> 0		09 <u>9</u> 9
CS	A	0	1	2	3	4	5	6	7	8	9

12	0 <u>9</u> 99										
11	9 <u>1</u> 70										
10	3 <u>2</u> 52										
9	1 <u>2</u> 39										
8	0 <u>4</u> 28										
7	1 <u>7</u> 25										
6	8 <u>4</u> 25										
5	1 <u>4</u> 24										
4	4 <u>5</u> 18										
3	7 <u>0</u> 13					0 <u>4</u> 28					
2	7 <u>0</u> 09	7 <u>0</u> 13		3 <u>2</u> 52		8 <u>4</u> 25			1 <u>7</u> 25		
1	0 <u>7</u> 01	7 <u>0</u> 09	9 <u>1</u> 70	1 <u>2</u> 39		1 <u>4</u> 24	4 <u>5</u> 18		0 <u>7</u> 01		0 <u>9</u> 99
CS	A	0	1	2	3	4	5	6	7	8	9

12	<u>0</u> 999										
11	<u>1</u> 725										
10	<u>0</u> 701										
9	<u>4</u> 518										
8	<u>0</u> 428										
7	<u>8</u> 425										
6	<u>1</u> 424										
5	<u>3</u> 252										
4	<u>1</u> 239										
3	<u>9</u> 170	<u>0</u> 999	<u>1</u> 725								
2	<u>7</u> 013	<u>0</u> 701	<u>1</u> 424						<u>7</u> 013		
1	<u>7</u> 009	<u>0</u> 428	<u>1</u> 239		<u>3</u> 252	<u>4</u> 518			<u>7</u> 009	<u>8</u> 425	<u>9</u> 170
CS	A	0	1	2	3	4	5	6	7	8	9

12	<u>9</u> 170										
11	<u>8</u> 425										
10	<u>7</u> 013										
9	<u>7</u> 009										
8	<u>4</u> 518										
7	<u>3</u> 252										
6	<u>1</u> 725										
5	<u>1</u> 424										
4	<u>1</u> 239										
3	<u>0</u> 999										
2	<u>0</u> 701										
1	<u>0</u> 428										
CS	A	0	1	2	3	4	5	6	7	8	9



Sắp xếp theo cơ số (Radix Sort)



❖ Đánh giá thuật giải:

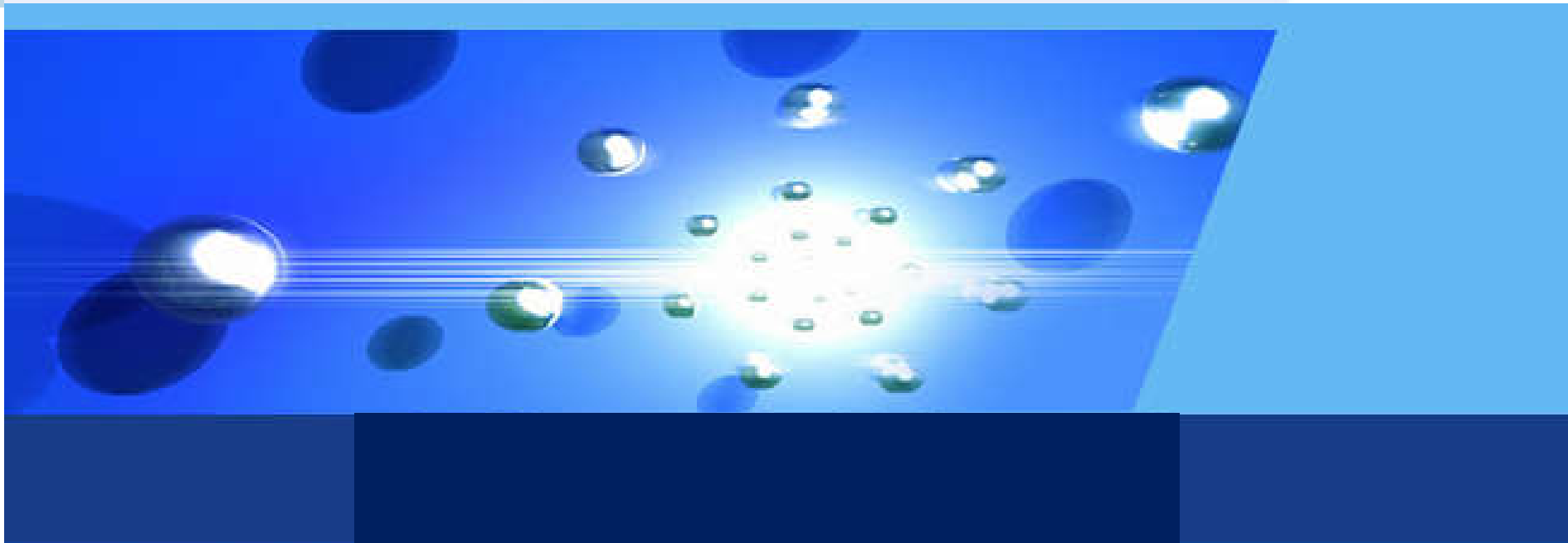
- Với dãy N số, mỗi con số có tối đa m chữ số, thuật giải thực hiện m lần các thao tác phân lô và ghép lô.
- Trong thao tác phân lô, mỗi phần tử chỉ được xét đúng một lần, khi ghép cũng vậy. Như vậy, chi phí cho việc thực hiện thuật giải hiển nhiên là $O(2mn) = O(n)$.



Bài tập



❖ Bài 1. Cài đặt thuật toán Radix sort.



Hết chương 2