Chương 3:

Tìm kiếm và sắp xếp (tt)

Nội dung

3.1 Tìm kiếm

3.2 Sắp xếp

- 3.2.1 Selection Sort
- 3.2.2 Bubble Sort
- 3.2.3 Insertion Sort
- 3.2.4 Quick Sort
- 3.2.5 Heap Sort

3.2 Sắp xếp

Sắp xếp là quá trình bố trí lại các phần tử của một tập đối tượng theo một thứ tự nhất định.

- Ví dụ:
  - {1, 2, 5, 7, 9, 12}, {14, 12, 7, 5, 2, 1}
  - {“An” “Binh” “Dương” “Hương”}
- Việc sắp xếp là một bài toán phổ biến trong tin học.
  - Do các yêu cầu tìm kiếm thuận lợi, sắp xếp kết xuất cho các bảng biểu...

3.2 Sắp xếp

- Mô tả bài toán: Để tiện cho việc minh họa các thuật toán sắp xếp ta mô tả bài toán như sau:
  - Cho mảng a gồm n phần tử a[0..n-1] là mảng các giá trị khóa. Khai báo là:

```
const   int   n = 10;
int   a[n];
```

  - Yêu cầu: sắp xếp các giá trị này sao cho mảng a có thứ tự tăng dần hoặc giảm dần.

3.2 Sắp xếp

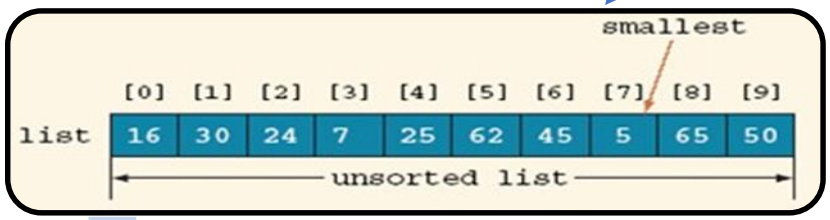
- 3.2.1 Sắp xếp kiểu lựa chọn (Selection Sort)
- 3.2.2 Sắp xếp kiểu đổi chỗ (Bubble Sort)
- 3.2.3 Sắp xếp kiểu thêm dần (Insertion Sort)
- 3.2.4 Sắp xếp nhanh (Quick Sort)
- 3.2.5 Sắp xếp kiểu vun đống (Heap Sort)

3.2.1 Selection Sort

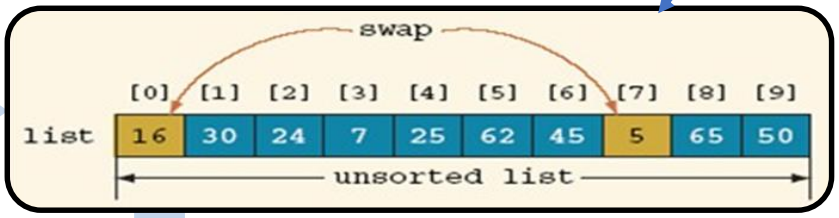
- **Ý tưởng chính**
  - Ở lượt thứ i, chọn khóa nhỏ nhất trong dãy từ a[i..n-1] ra và đổi chỗ nó với a[i], khi đó a[i] sẽ trở thành khoá nhỏ nhất trong dãy đang xét.
  - Thực hiện n-1 lượt với i = 0 -> n-2.

3.2.1 Selection Sort

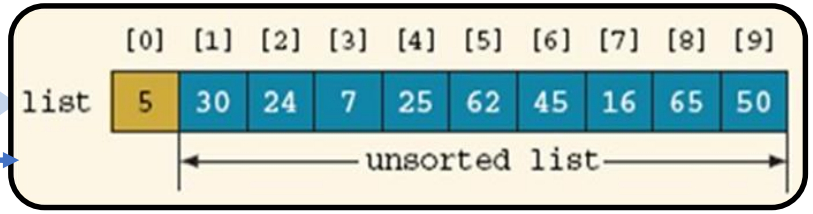
1. Tìm giá trị nhỏ nhất trong mảng



2. Đổi chỗ nó với phần tử ở vị trí đầu tiên trong mảng



3. Lặp lại các bước trên cho tất cả các phần tử còn lại của mảng bắt đầu từ vị trí thứ hai.



3.2.1 Selection Sort

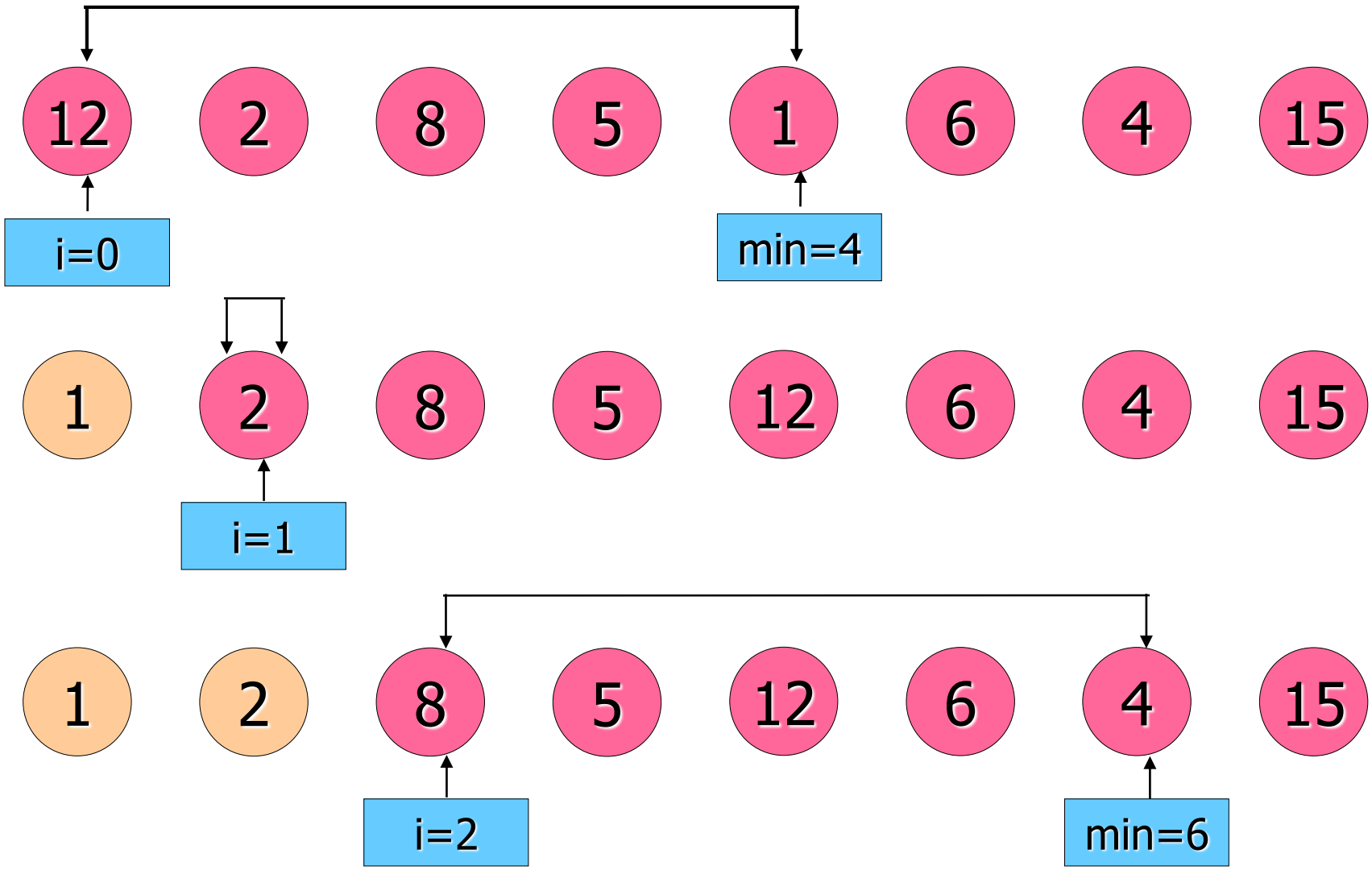
- Các bước thực hiện:
  - B1: i = 0;
  - B2: Tìm phần tử a[min] nhỏ nhất trong dãy hiện hành từ a[i] đến a[n-1]
  - B3: Đổi chỗ a[i] và a[min]
  - B4: Nếu i < n -1 thì i= i+1 $\Rightarrow$ Lặp lại B2

    Ngược lại $\Rightarrow$ Dừng
- Ví dụ: cho dãy số như sau:

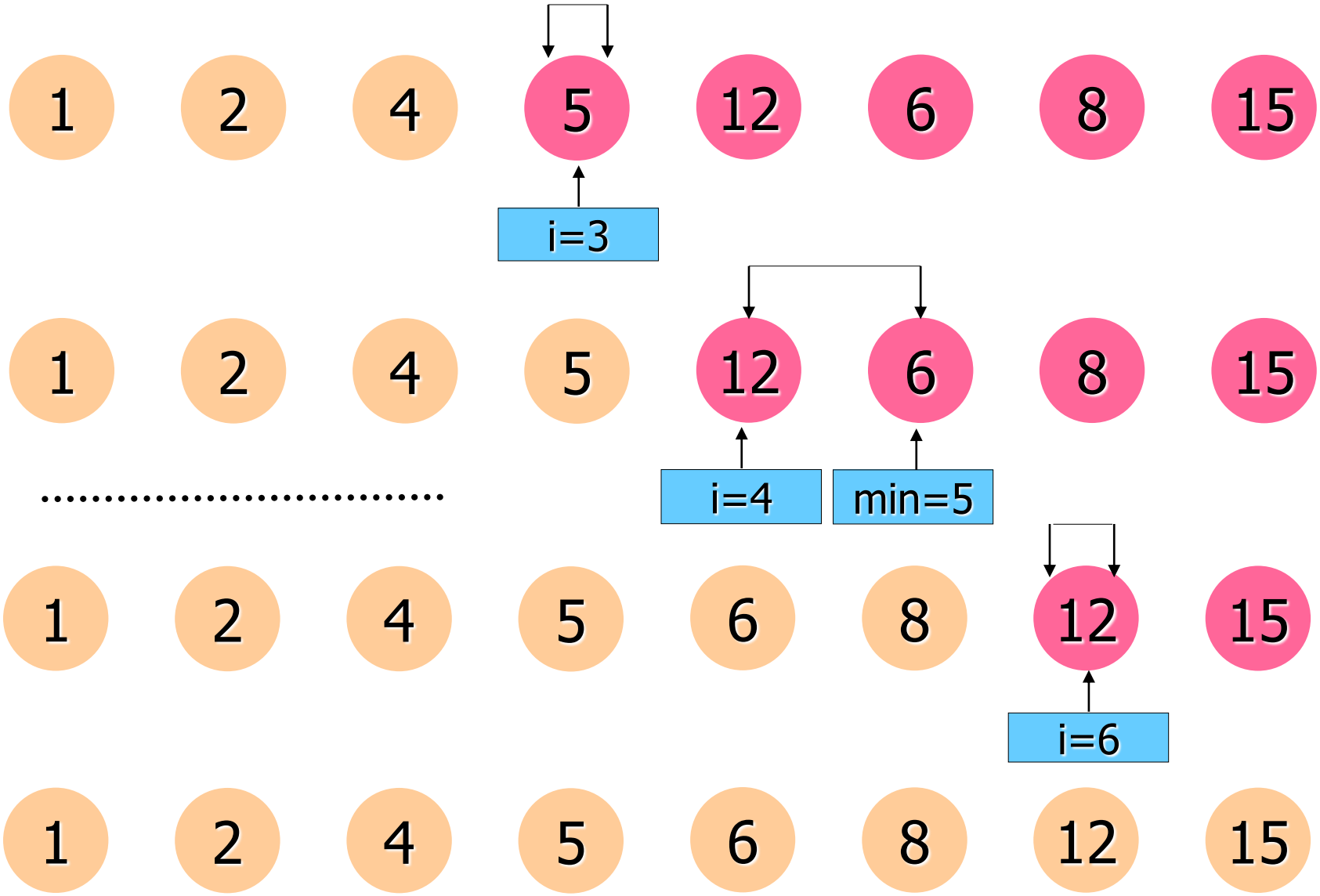
  12 2 8 5 1 6 4 15

  Minh họa phương pháp lựa chọn như sau

3.2.1 Selection Sort



3.2.1 Selection Sort



3.2.1 Selection Sort

- Cài đặt swap

```
void  swap(int  *x, int   *y)
{
      int  tg;          //phần tử trung gian
      tg  = *x;
      *x = *y;
      *y = tg;
}
```

3.2.1 Selection Sort

- Cài đặt Selection Sort

```
1. void SelectionSort(int a[], int n)
2. {
3.     int  min; //lưu chỉ số phần tử nhỏ nhất
4.     for(int i = 0; i < n-1; i++)
5.     {
6.           min = i;
7.           for(int j = i+1; j < n; j++)
8.                 if (a[j] < a[min])
9.                       min = j;
10.          if (i != min)   swap(&a[min],&a[i]);
11.    }
12.}
```

3.2.1 Selection Sort

- Đánh giá giải thuật

```
4.    for(int i = 0; i < n-1; i++)
5.    {
6.          min = i;
7.          for(int j = i+1; j < n; j++)
8.                if (a[j] < a[min])
9.                      min = j;
10.         if (i != min)    swap(a[min], a[i]);
11.   }
```

Độ phức tạp của giải thuật là:

$$T(n) = C\sum_{i=0}^{n-1}(n-i) = O(n^2)$$

3.2.2 Bubble Sort

- Ý tưởng chính
  - Xuất phát từ cuối dãy, đổi chỗ các cặp phần tử kế cận nghịch thế để đưa phần tử nhỏ hơn về đầu.
  - Sau đó ở bước tiếp theo không xét phần tử đó nữa. Do vậy 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 được xét.

3.2.2 Bubble Sort

- Các bước tiến hành
  - B1: i=0; // lần xử lý đầu tiên
  - B2: j=n-1; // duyệt từ cuối dãy ngược về vị trí i

    Trong khi (j>i) thực hiện:

    Nếu a[j] < a[j-1]: Đổi chỗ a[j] và a[j-1]

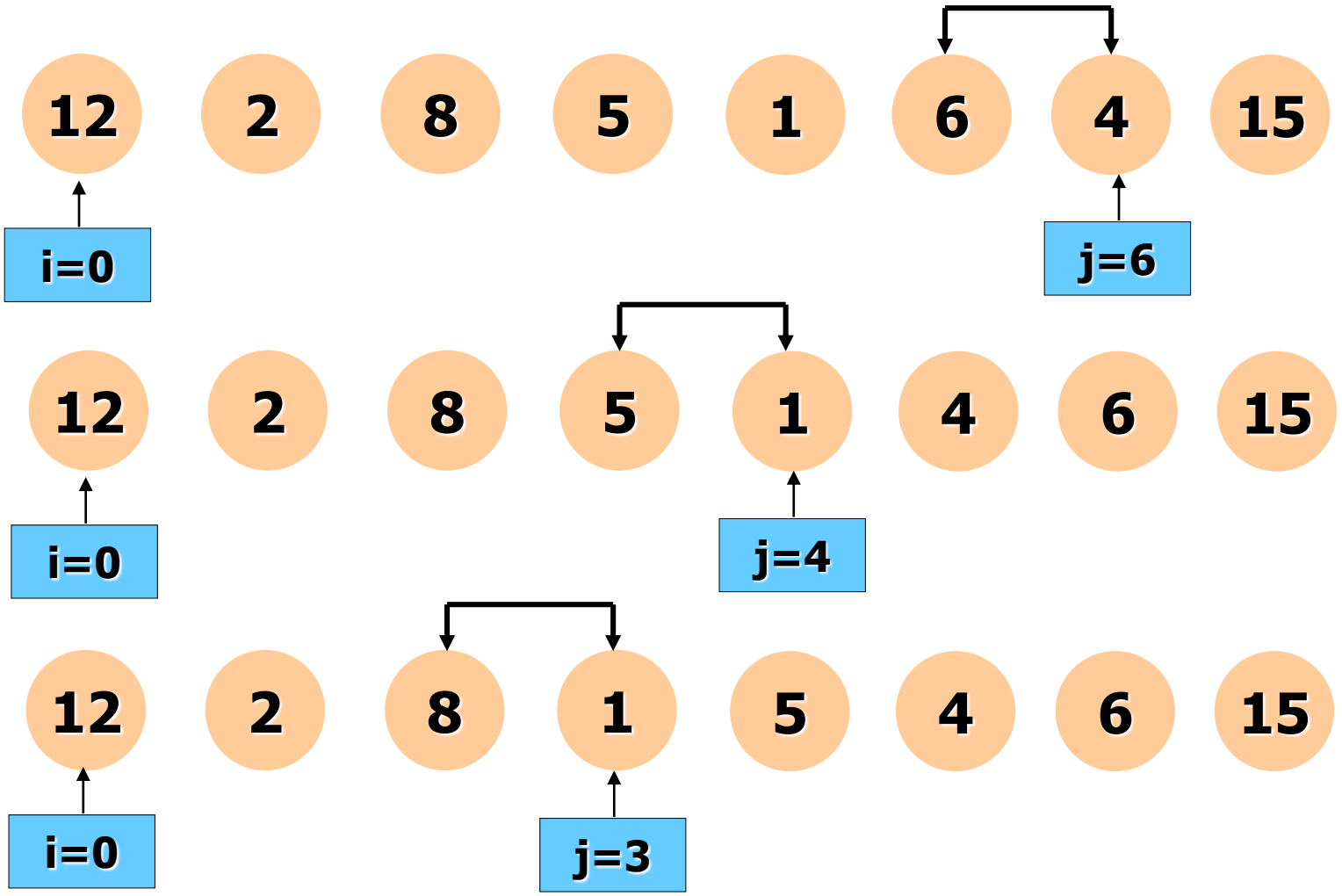
    j = j -1;
  - B3: i = i+1; // lần xử lý kế tiếp

    Nếu i > n-1: Hết dãy $\Rightarrow$ Dừng

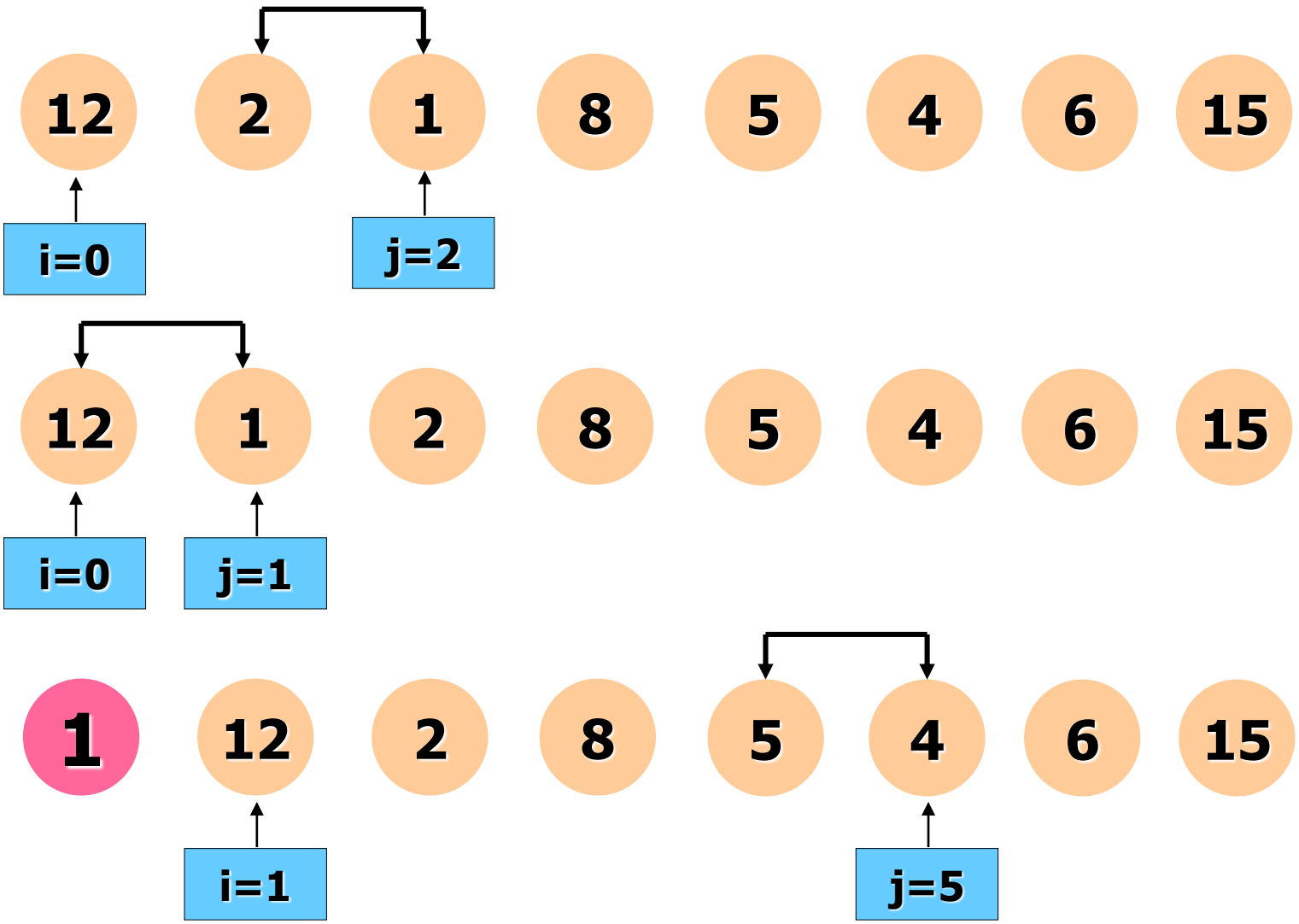
    Ngược lại: quay lại B2
- Ví dụ: Minh họa sắp xếp dãy số sau:

  12 2 8 5 1 6 4 15

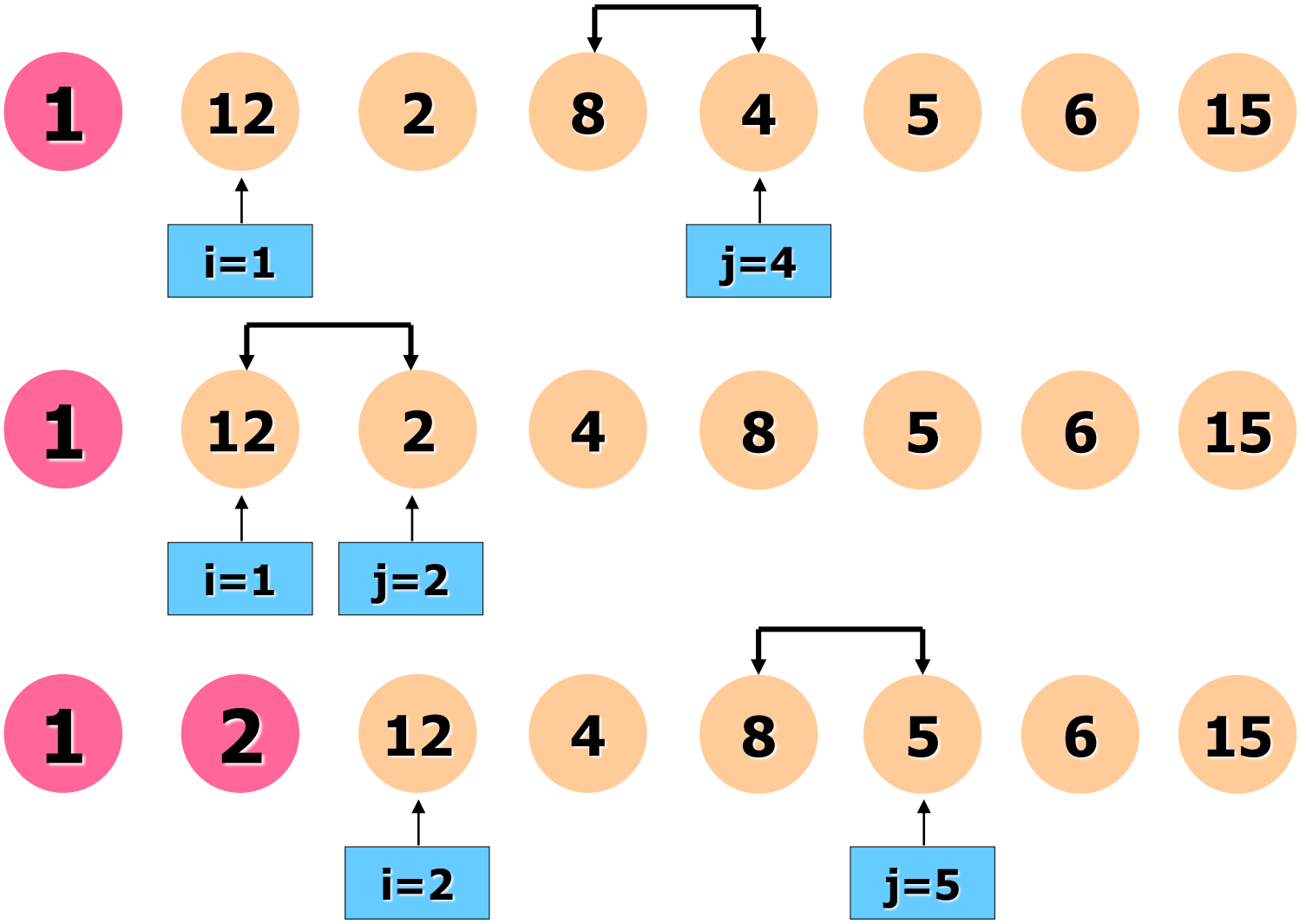
3.2.2 Bubble Sort



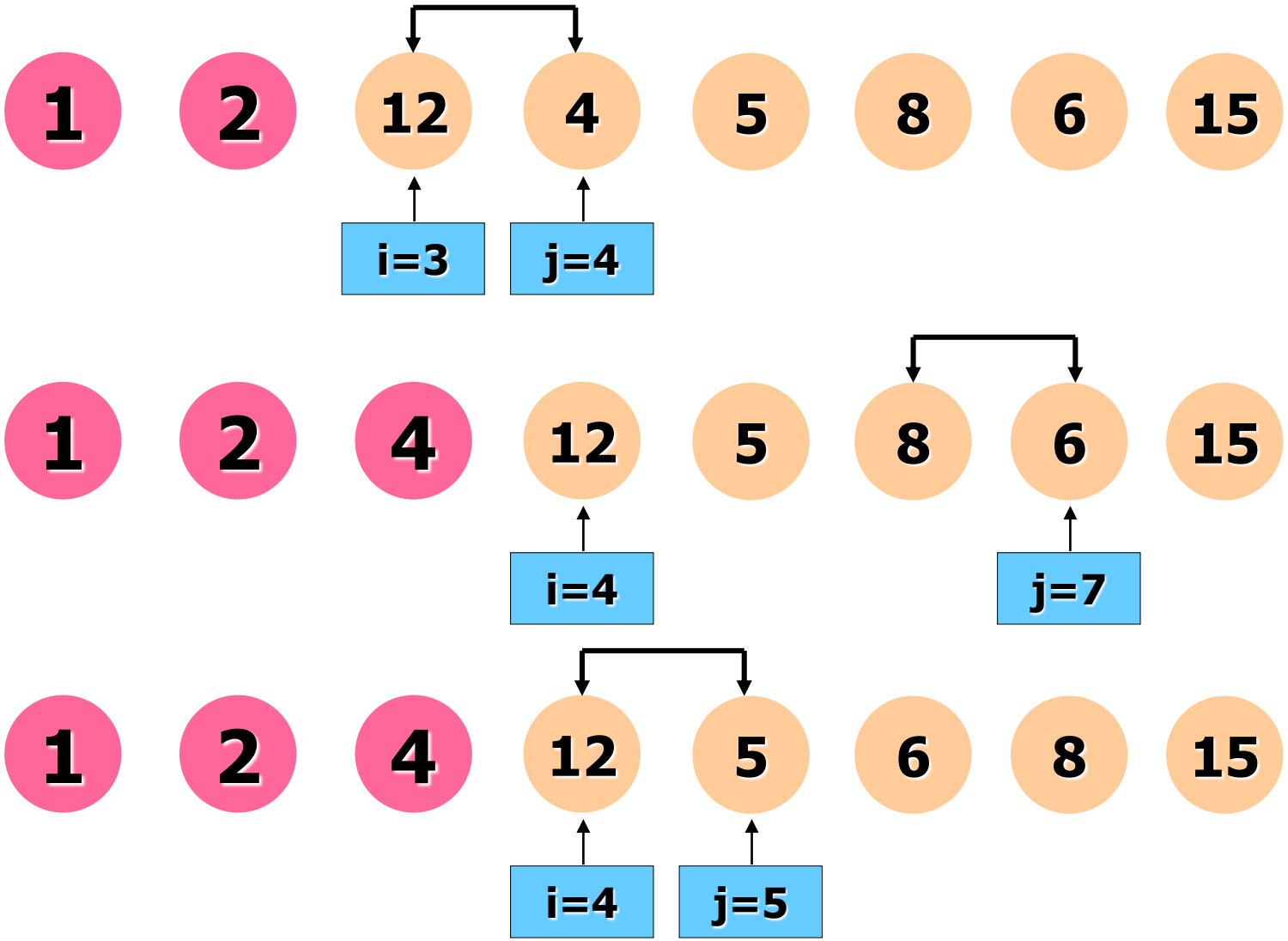
3.2.2 Bubble Sort



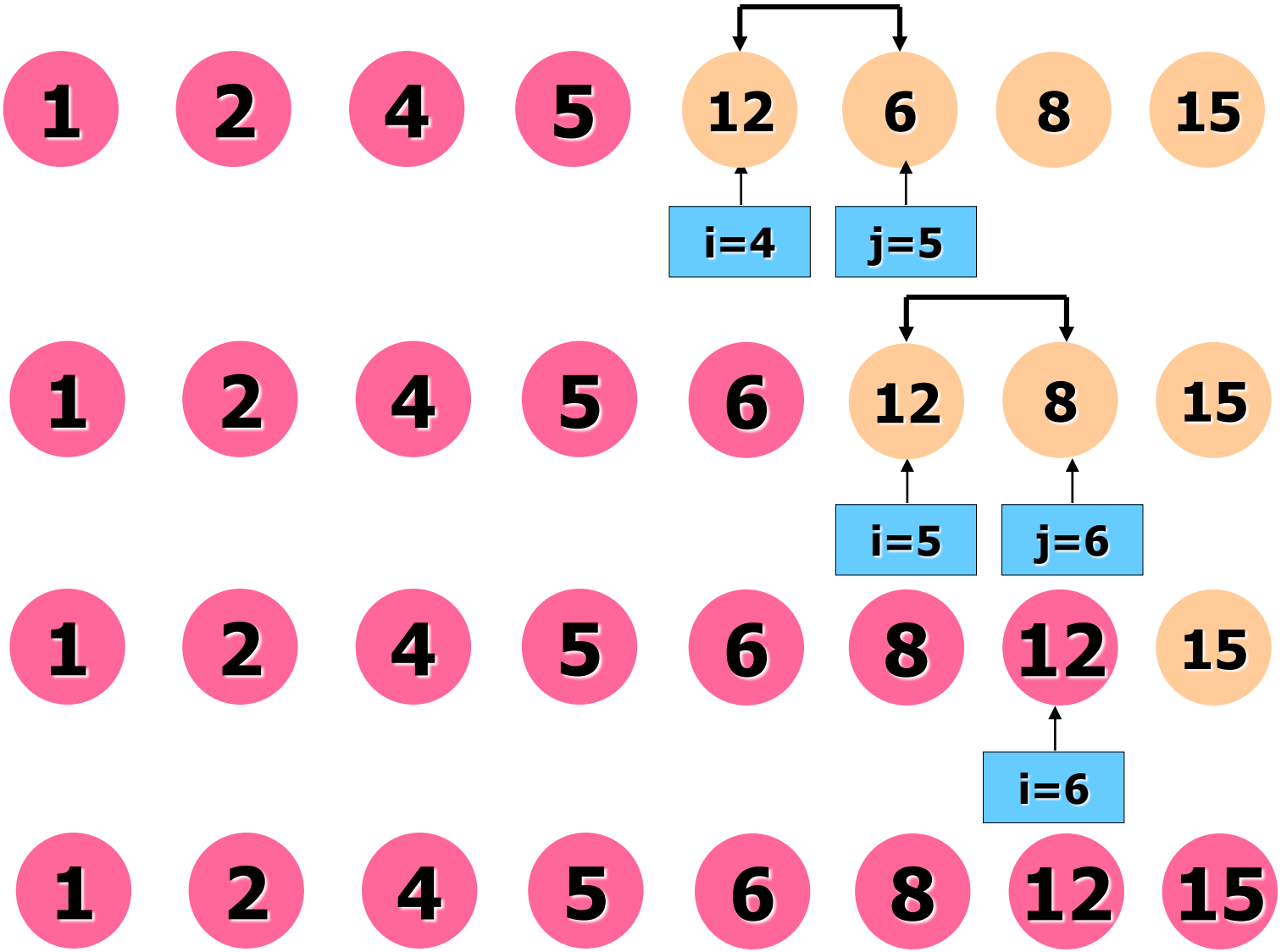
3.2.2 Bubble Sort



3.2.2 Bubble Sort



3.2.2 Bubble Sort



3.2.2 Bubble Sort

- Cài đặt Bubble sort

```
1. void BubbleSort(int a[], int n)
2. {
3.    int i, j;
4.    for(i = 0; i < n-1; i++)
5.          for(j = n-1; j >i; j--)
6.                if (a[j] < a[j-1])
7.                      swap(&a[j], &a[j-1]);
8. }
```

3.2.2 Bubble Sort

- Đánh giá giải thuật

```
4.   for(i = 0; i < n-1; i++)
5.        for(j = n-1; j >i; j--)
6.             if (a[j] < a[j-1])
7.                  swap(a[j], a[j-1]);
```

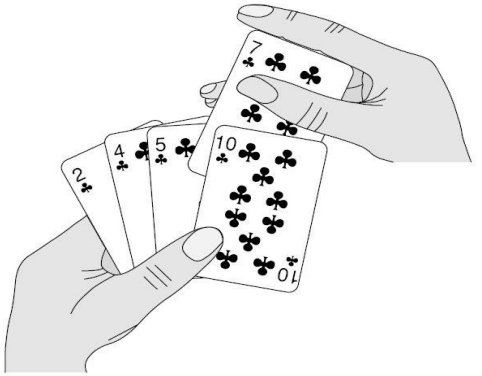
Độ phức tạp của giải thuật là:

$$T(n) = C\sum_{i=0}^{n-1}(n-i) = O(n^2)$$

3.2.3 Insertion Sort

- **Ý tưởng chính**
  - Cho dãy ban đầu a[0], a[1],.., a[n-1], ta có thể xem dãy con gồm một phần tử a[0] đã được sắp.
  - Sau đó thêm a[1] vào đoạn a[0] sao cho a[0] a[1] được sắp.
  - Tiếp tục thêm a[2] vào để có a[0] a[1] a[2] được sắp....
  - Cho đến khi thêm xong a[n-1] vào đoạn a[0] a[1]...a[n-2]

    $\Rightarrow$ đoạn a[0] a[1]...a[n-2] a[n-1] được sắp.



3.2.3 Insertion Sort

- **Các bước tiến hành**
  - B1: i = 1; //giả sử có đoạn a[0] đã được sắp
  - B2: x= a[i];

    Tìm được vị trí cần chèn x vào là **pos**
  - B3: Dời chỗ các phần tử từ a[pos] $\Rightarrow$ a[i-1] sang phải một vị trí để dành chỗ cho a[i].
  - B4: a[pos] = x; // có đoạn a[0]...a[i] được sắp.
  - B5: i = i +1;

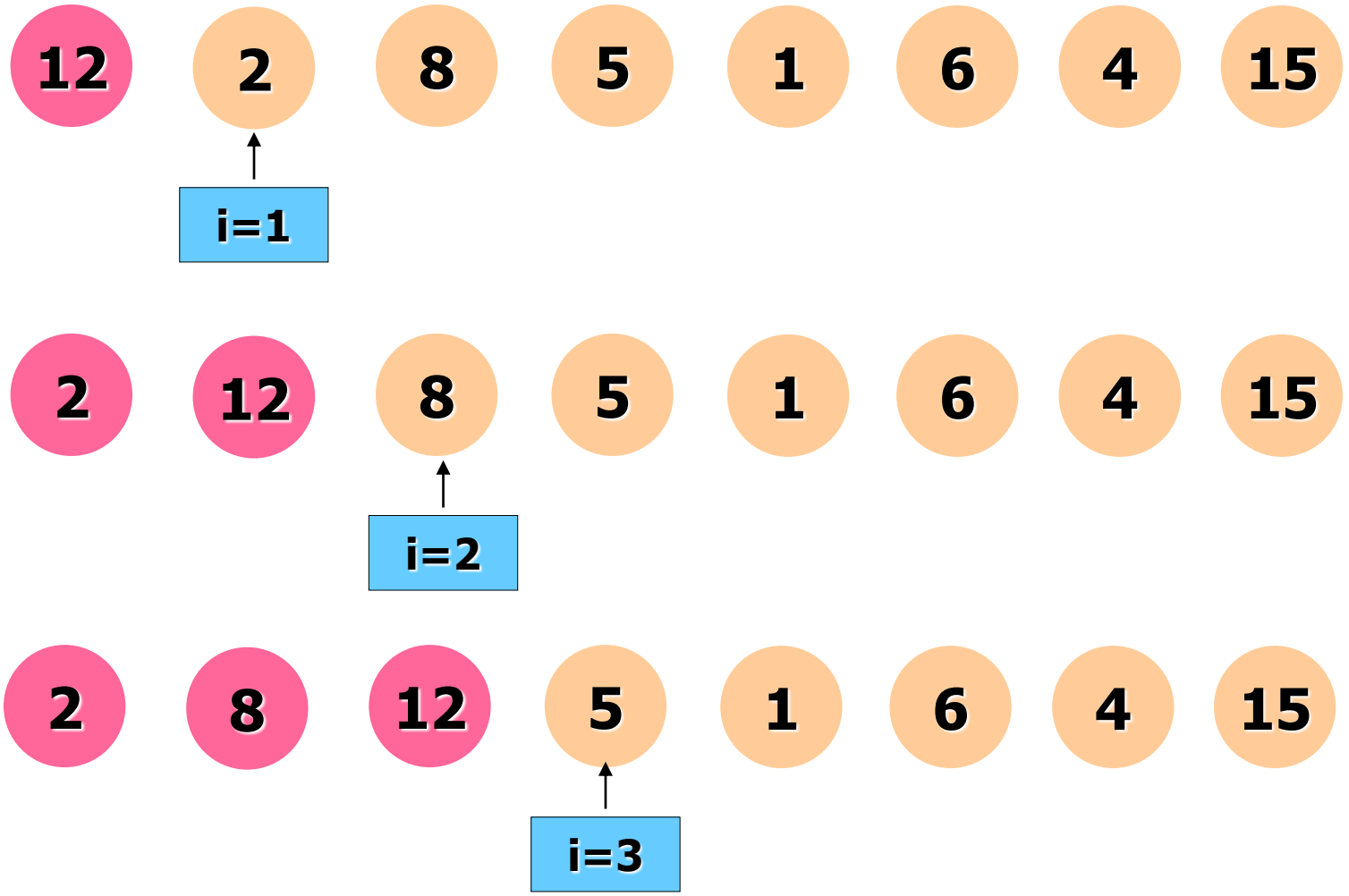
    Nếu i $\leq$ n-1: Lặp lại B2

    Ngược lại: Dừng $\Rightarrow$ Dãy đã được sắp

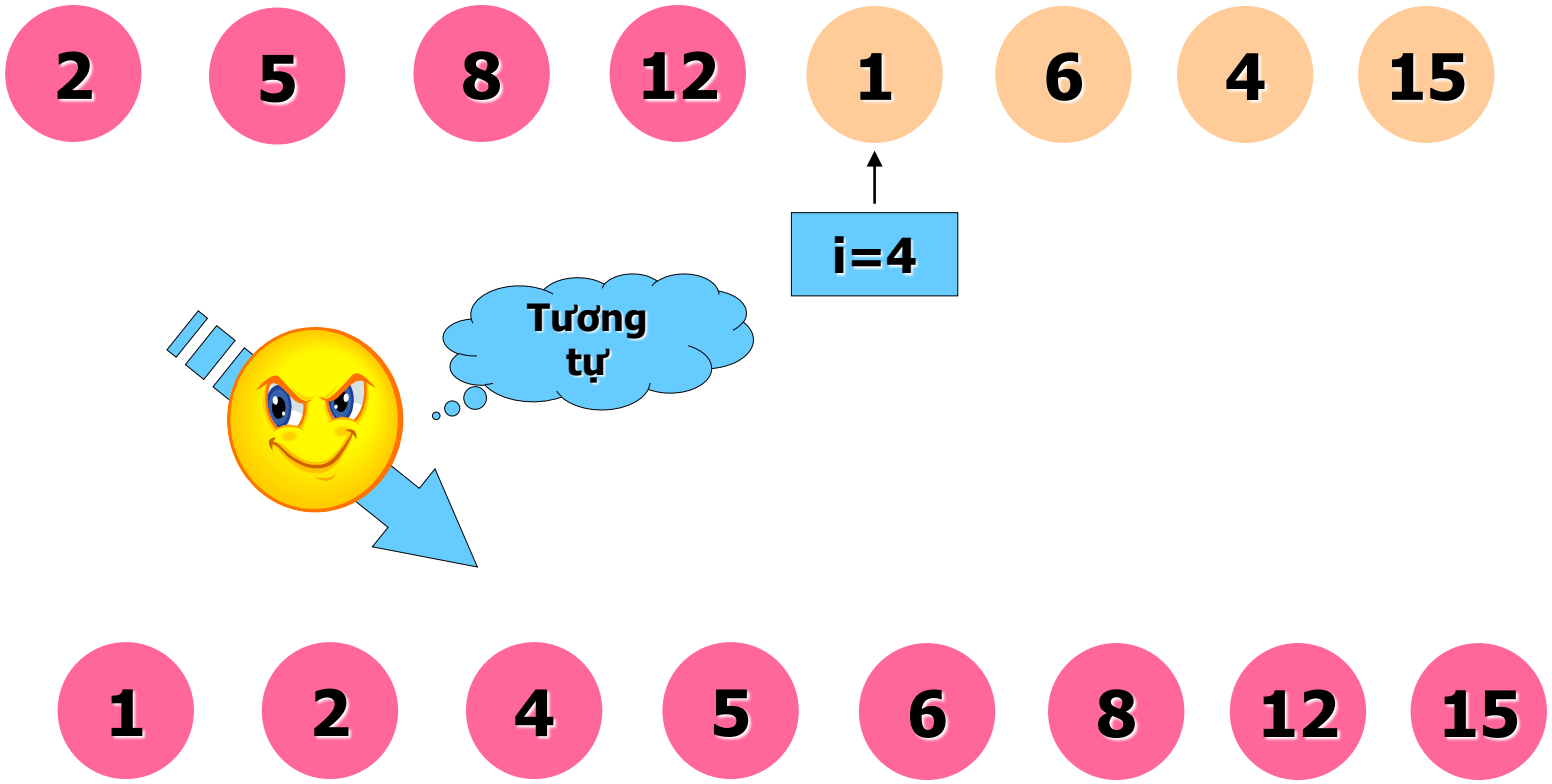
- Ví dụ: minh họa phương pháp chèn với dãy:

  12 2 8 5 1 6 4 15

3.2.3 Insertion Sort



3.2.3 Insertion Sort



3.2.3 Insertion Sort

```
1. void InsertionSort(int a[], int n)
2. {
3.    int pos, i, x; //x lưu phần tử a[i]
4.    for(i=1; i < n; i++)
5.    {     x = a[i];  pos = i-1;
6.          while ((pos ≥ 0) && (a[pos] > x))
7.          { //dời chỗ các phần tử đứng sau x
8.                a[pos+1] = a[pos];
9.                pos--;
10.         }
11.         a[pos+1] = x;  //chèn x vào dãy mới
12.   }
13.}
```

3.2.3 Insertion Sort

- **Đánh giá giải thuật**

```
4.   for(i=1; i < n; i++)
5.   {     x = a[i];  pos = i-1;
6.         while ((pos ≥ 0) && (a[pos] > x))
7.         { //dời chỗ các phần tử đứng sau x
8.               a[pos+1] = a[pos];
9.               pos--;
10.        }
11.        a[pos+1] = x;  //chèn x vào dãy mới
12.  }
```

Độ phức tạp của giải thuật phụ thuộc tình trạng dữ liệu vào, ta tính thời gian thực hiện trong 3 trường hợp: tốt nhất, xấu nhất và trung bình.

3.2.3 Insertion Sort

- **Đánh giá giải thuật**
  - Trường hợp tốt nhất khi dãy đầu vào đã sắp xếp **trùng** với thứ tự cần sắp xếp

$$T(n) = O(n)$$

  - Trường hợp xấu nhất khi dãy đầu vào đã sắp xếp **ngược** với thứ tự cần sắp xếp

$$T(n) = O(n^2)$$

  - Trường hợp trung bình khi dãy đầu vào bất kỳ

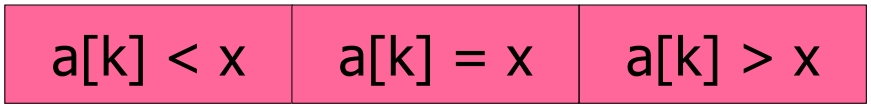
$$T(n) = O(n^2)$$

3.2.4 QuickSort

- Thuật toán do Hoare đề xuất
  - Tốc độ trung bình nhanh hơn thuật toán khác
  - Do đó Hoare dùng "*quick*" để đặt tên
- Ý tưởng chính
  - QS phân hoạch dãy ban đầu thành hai phần dựa vào một giá trị x
    - Dãy 1: gồm các phần tử a[i] ko lớn hơn x
    - Dãy 2: gồm các phần tử a[i] ko nhỏ hơn x

3.2.4 QuickSort

- Sau khi phân hoạch thì dãy ban đầu được phân thành ba phần:
- a[k] < x, với k = 1...i
- a[k] = x, với k = i..j
- a[k] > x, với k = j..n



3.2.4 QuickSort

- GT để sắp xếp dãy a[left], a[left+1],...,a[right]: được phát biểu theo cách đệ quy như sau:
- B1: Phân hoạch dãy a[left]...a[right] thành các dãy con:
  - Dãy con 1: a[left]...a[j] < x
  - Dãy con 2: a[j+1]...a[i-1] = x
  - Dãy con 3: a[i]...a[right] > x
- B2:
  - Nếu (left < j) // dãy con 1 có nhiều hơn 1 phần tử
    Phân hoạch dãy a[left]...a[j]
  - Nếu (i < right) // dãy con 3 có nhiều hơn 1 phần tử
    Phân hoạch dãy a[i]...a[right]

3.2.4 QuickSort

- Phân hoạch dãy a[left], a[left+1],...,a[right] thành hai dãy con:
- B1: Chọn tùy ý một phần tử a[k] trong dãy là giá trị mốc, left ≤ k ≤ right,
  - Cho x = a[k], i = left, j = right.
- B2: Tìm và hoán vị cặp phần tử a[i] và a[j] không đúng thứ tự đang sắp.
  - B2-1: Trong khi a[i] < x ⇒ i++;
  - B2-2: Trong khi a[j] > x ⇒ j--;
  - B2-3: Nếu i < j ⇒ swap(a[i], a[j]) // a[i], a[j] sai thứ tự
- B3:
  - Nếu i < j: ⇒ Bước 2;
  - Nếu i ≥ j: ⇒ Dừng.

3.2.4 QuickSort

```
1.  void QuickSort(int a[],int left,int right)
2.  {
3.      int   i, j, x;
4.      x = a[(left+right)/2];
5.      i = left;   j = right;
6.      do {
7.              while (a[i] < x)        i++;
8.             while (a[j] > x)  j--;
```

3.2.4 QuickSort

```
9.          if ( i <= j)
10.         {   swap(a[i], a[j]);
11.             i++;            //qua phần tử kế tiếp
12.             j--;            //qua phần tử đứng trước
13.         }
14. } while (i<j);
13.  if (left < j)              //ph đoạn bên trái
14.         QuickSort(a, left, j);
15.  if (right > i)             //ph đoạn bên phải
16.         QuickSort(a, i, right);
17. }
```

3.2.4 QuickSort

- **Đánh giá giải thuật**

  Hiệu quả thực hiện của giải thuật Quick Sort phụ thuộc vào việc chọn giá trị mốc.

  Trường hợp tốt nhất xảy ra nếu mỗi lần phân hoạch đều chọn được phần tử median (phần tử lớn hơn (hay bằng) nửa số phần tử, và nhỏ hơn (hay bằng) nửa số phần tử còn lại) làm mốc, khi đó dãy được phân chia thành 2 phần bằng nhau và cần $n*log_2(n)$ bước phân hoạch thì sắp xếp xong.

3.2.4 QuickSort

- **Đánh giá giải thuật**

  Nhưng nếu mỗi bước phân hoạch phần tử được chọn có giá trị cực đại (hay cực tiểu) là mốc, dãy sẽ bị phân chia thành 2 phần không đều: một phần chỉ có 1 phần tử, phần còn lại gồm (n-1) phần tử, do vậy cần thực hiện n bước phân hoạch mới sắp xếp xong. Vậy độ phức tạp trong trường hợp xấu nhất là $O(n^2)$.

  Trường hợp trung bình: $O(n\log_2 n)$

3.2.4 QuickSort

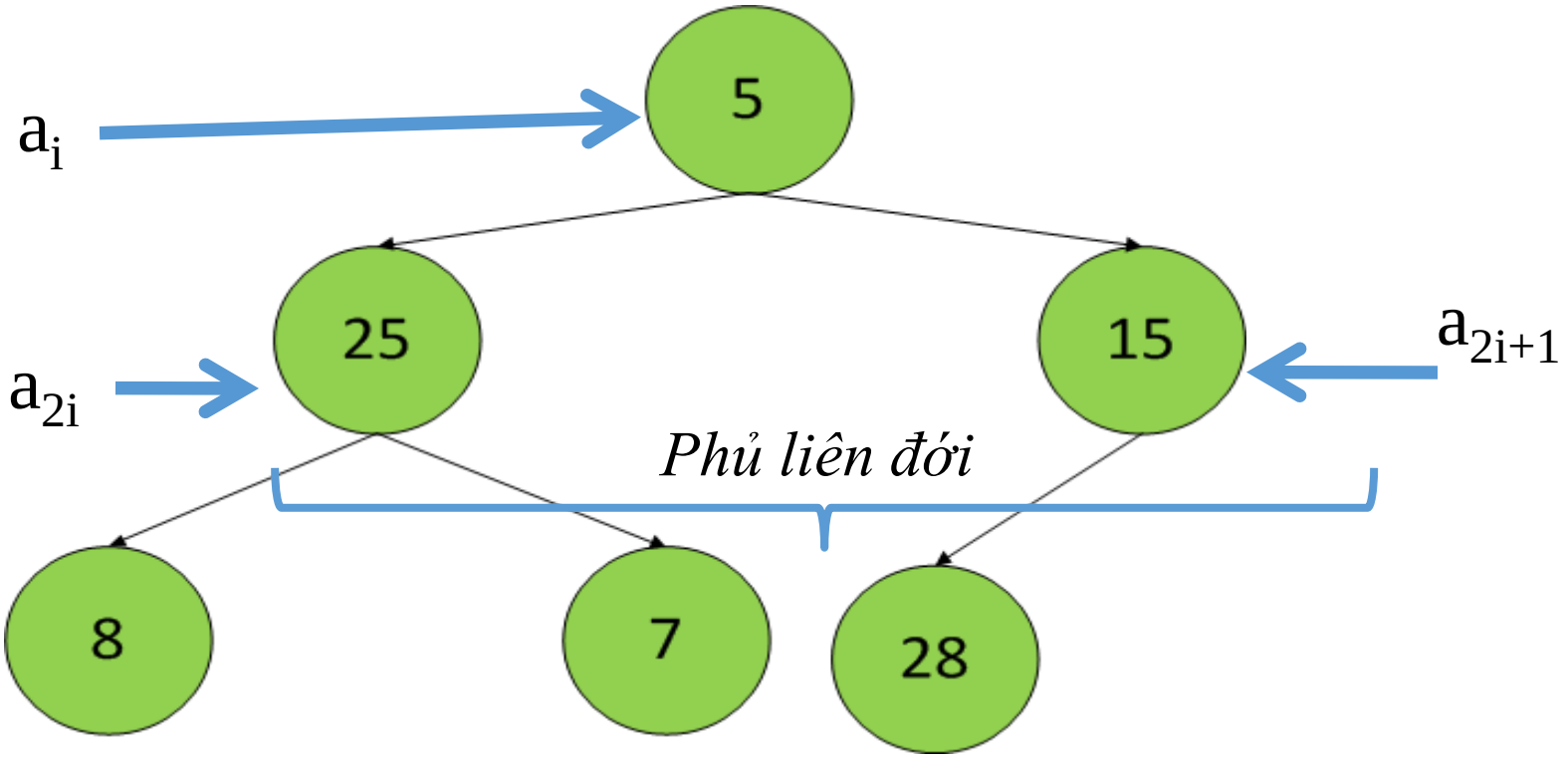
Ưu điểm	Khuyết điểm
Là thuật toán chạy nhanh nhất ở trường hợp trung bình.	Không ổn định
Không cần bộ nhớ phụ	Thời gian xấu nhất chiếm thời gian là $O(n^2)$
Dùng tốt cho DSLK	

3.2.5 HeapSort

- **Ý tưởng chính**
- Nhận xét: Khi tìm phần tử nhỏ nhất ở bước i, phương pháp sắp xếp chọn trực tiếp không tận dụng được các thông tin đã có được do các phép so sánh ở bước i-1.
- Vì lý do trên người ta tìm cách xây dựng một thuật toán sắp xếp có thể khắc phục nhược điểm này.
- Mấu chốt để giải quyết vấn đề vừa nêu là phải tìm ra được một cấu trúc dữ liệu cho phép tích lũy các thông tin về sự so sánh giá trị các phần tử trong qua trình sắp xếp.

3.2.5 HeapSort

- **Định nghĩa Heap:**
- Giả sử xét trường hợp sắp xếp tăng dần, khi đó Heap được định nghĩa là một dãy các phần tử $a_p, a_2, \ldots, a_q$ thoả các quan hệ sau với mọi i thuộc [p, q]:



3.2.5 HeapSort

- **Giải thuật Heapsort :**

Giải thuật Heapsort trải qua 2 giai đoạn :

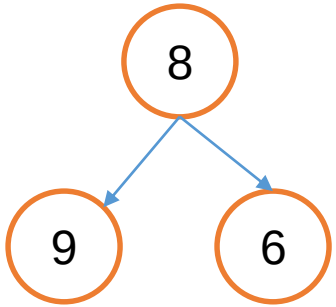
Giai đoạn 1: Hiệu chỉnh dãy số ban đầu thành heap;

So sánh giá trị a_i với giá trị lớn hơn trong hai giá trị a_{2i} và a_{2i+1}, nếu nhỏ hơn thì đổi chỗ chúng cho nhau.

Để vun mảng *a*[0..n-1] thành đống ta vun từ dưới lên, bắt đầu từ phần tử *a*[j]với *j* =[n/2]-1 ngược lên tới a[0].

3.2.5 HeapSort

- **Giải thuật Heapsort – Giai đoạn 1**



Vun thành đống với một nhánh cây:

- Tìm nút con lớn nhất trong 2 nút con
- So sánh nút con lớn nhất với nút gốc
  - Nếu giá trị nút gốc > giá trị nút con: nhánh cây đã là heap
  - Nếu giá trị nút gốc < giá trị nút con: đổi chỗ nút gốc và nút con

3.2.5 HeapSort

- **Giải thuật Heapsort**

```
1. void      Adjust(int a, int b)
2. {        int   i  =  a;
3.          int   j  =  2 * i;
4.          while (j <= b)
5.          {  int   k  =  j + 1;
6.             if (k <= b && A[k]< A[j])
7.                    j  =  k;
8.             if (A[i] < A[j])
9.                {
10.                   swap(A[i],A[j]);
11.                   i   =   j;
12.                   j   =   2 * i;
13.               }
14.            else      break;
15.         }
16. }
```

3.2.5 HeapSort

- **Giải thuật Heapsort – Giai đoạn 1**

Để vun mảng *A*[0..n-1] thành đống ta sử dụng hàm adjust vun từ dưới lên, bắt đầu từ phần tử *A*[j]với *j* =[n/2]-1 ngược lên tới a[0].

```
void      Creat_Heap(int   A[] , int  n)
{
     for(int  j = n / 2 - 1 ; j >= 0 ; j--)
          adjust(j, n-1);
}
```

3.2.5 HeapSort

- **Giải thuật Heapsort :**

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; hoánvị (a1, ar);

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

Dựa trên tính chất 3, ta có thể thực hiện giai đoạn 1 bằng cách bắt đầu từ heap mặc nhiên an/2+1 , an/2+2 ... an, sau đó thêm dần các phần tử an/2, an/2-1, ., a1 ta sẽ nhận được heap theo mong muốn.

3.2.5 HeapSort

- **Giải thuật Heapsort – Giai đoạn 2**

Sắp xếp dãy số dựa trên heap:

```
1. void HeapSort(int A[],int n)
2. {
3.      Creat_Heap(A, n);
4.      for (i = n-1; i >= 1; i--)
5.      {
6.          swap(&A[0], &A[i]);
7.          adjust(1, i-1);
8.      }
9. }
```

3.2.5 HeapSort

- **Đánh giá giải thuật**

Trong giai đoạn sắp xếp ta cần thực hiện n-1 bước mỗi bước cần nhiều nhất là $\log_2(n-1)$,$\log_2(n-2)$, … 1 phép đổi chỗ.

Như vậy độ phức tạp thuật toán Heap sort $O(n*\log_2 n)$

3.2.5 HeapSort

- **Ví dụ minh hoạ:**

Cho dãy số: 12 2 8 5 1 6 4 15

Sắp xếp dãy số sau theo giá trị tăng dần bằng Heap Sort

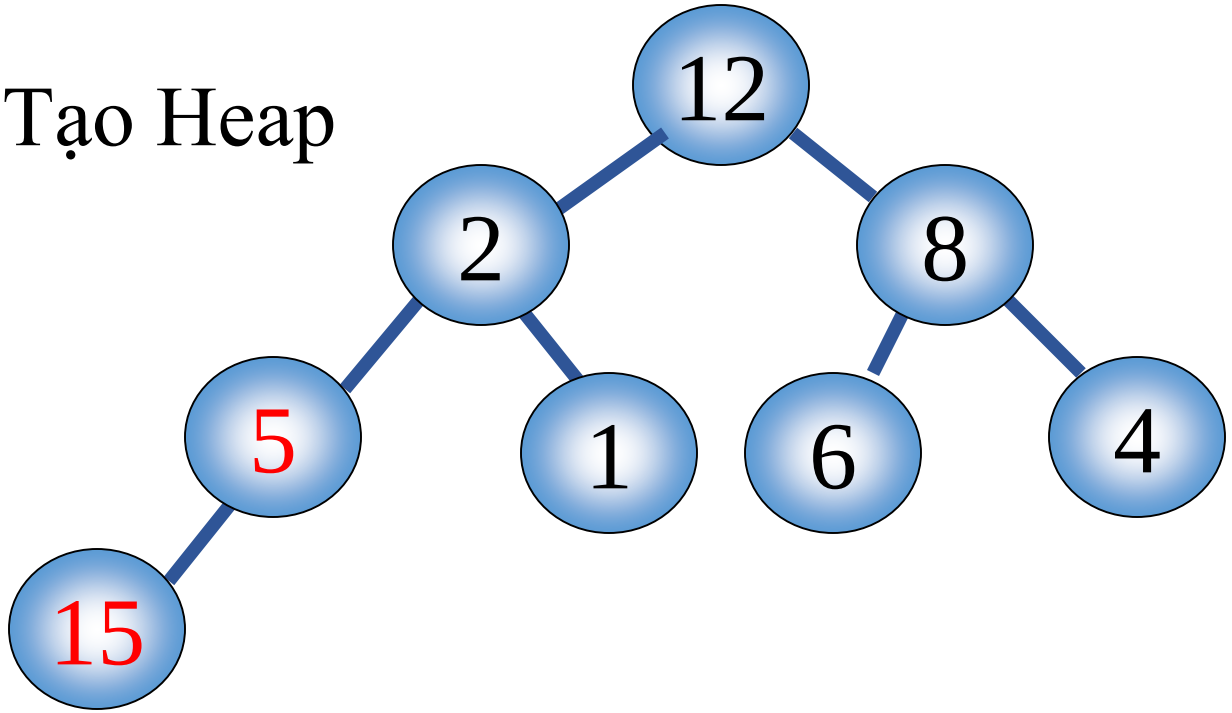
Các phần từ trên mảng có chỉ số từ [n/2] đến [n-1] đều là lá.

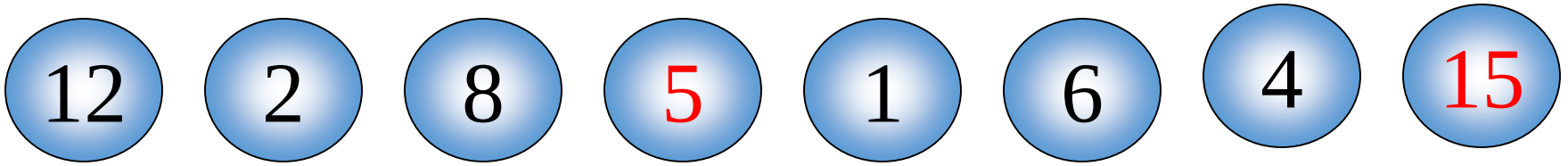
Trên mảng có 8 phần tử, nên các phần tử ở vị trí từ 4-7 là lá.

Ta sẽ bắt đầu vun đống từ phần thử thứ i=[n/2]-1=3

3.2.5 HeapSort

Bước 1: Tạo Heap

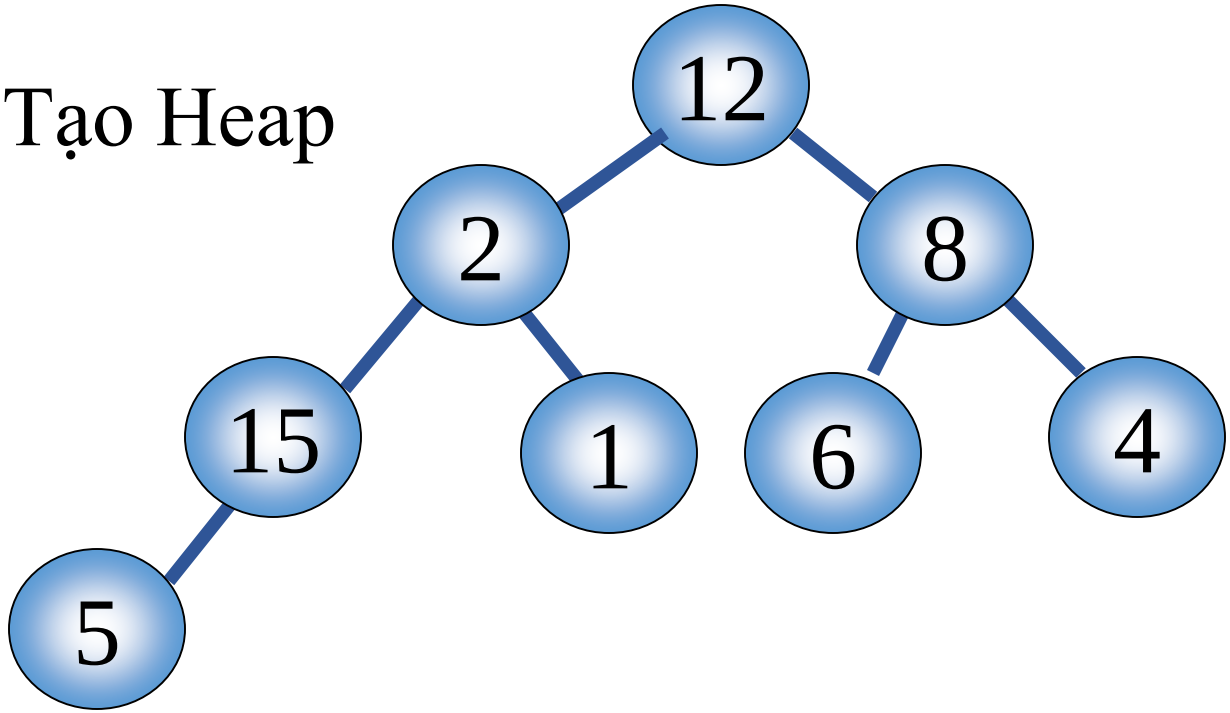


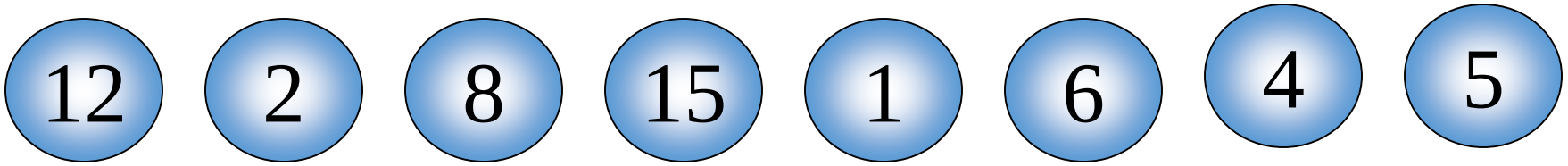


i=3

3.2.5 HeapSort

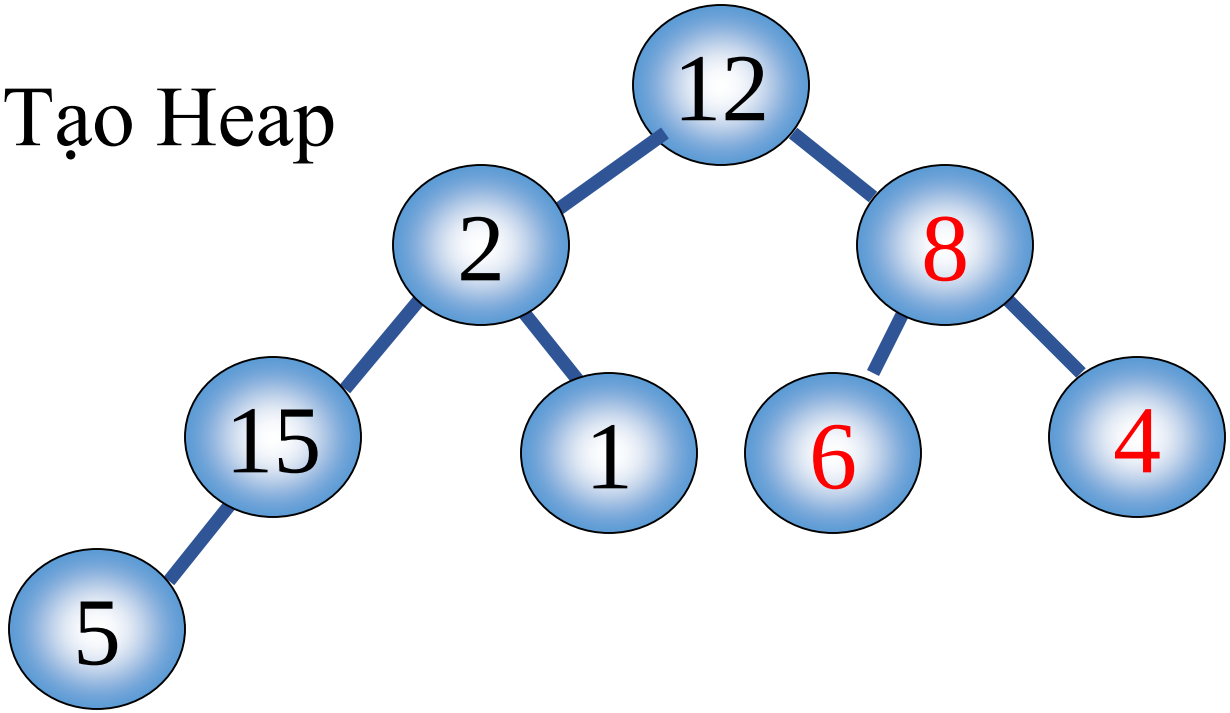
Bước 1: Tạo Heap

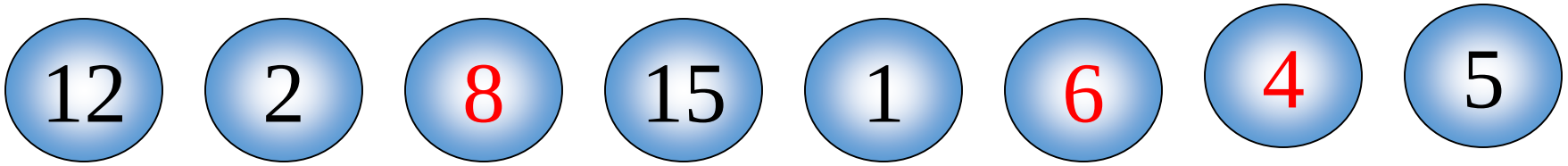




3.2.5 HeapSort

Bước 1: Tạo Heap

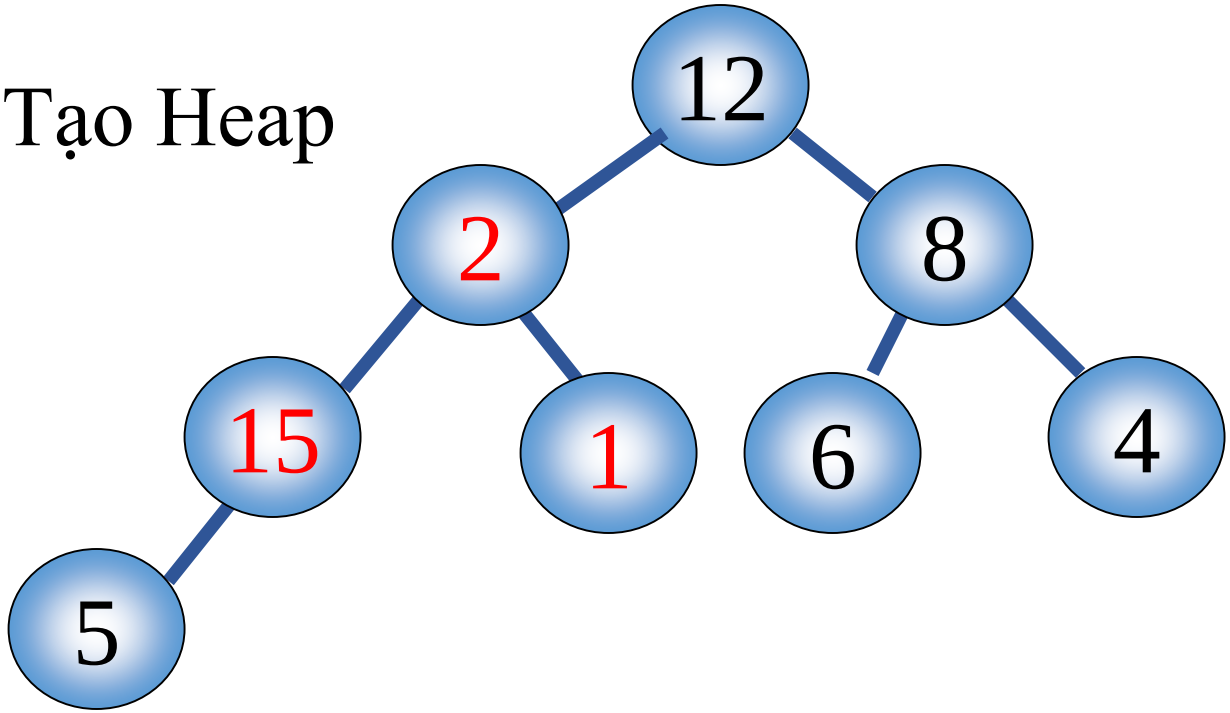


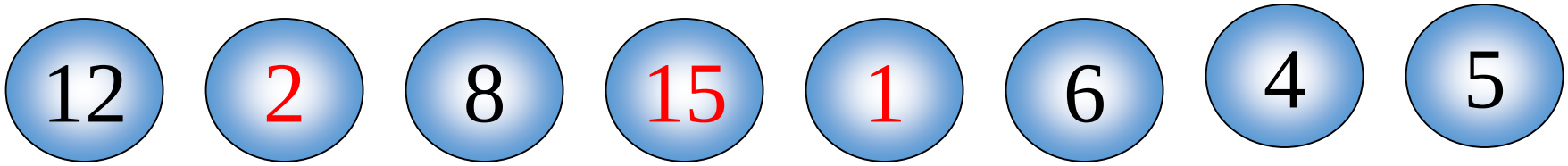


i=2

3.2.5 HeapSort

Bước 1: Tạo Heap

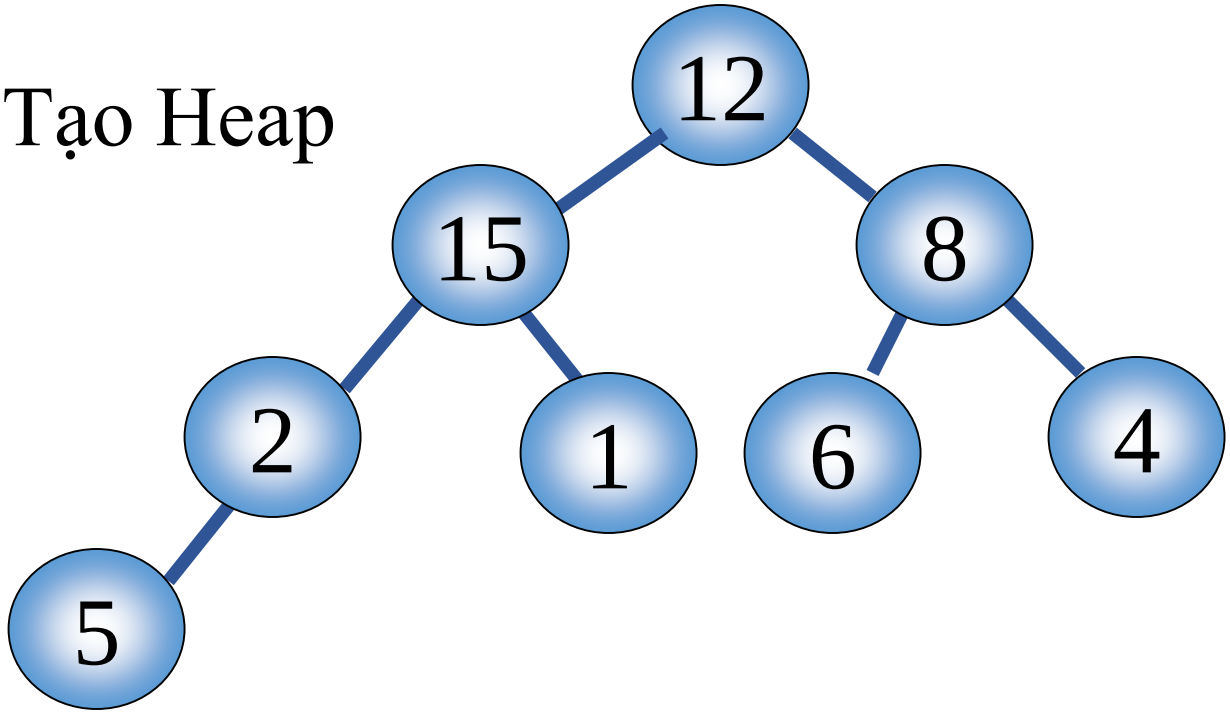


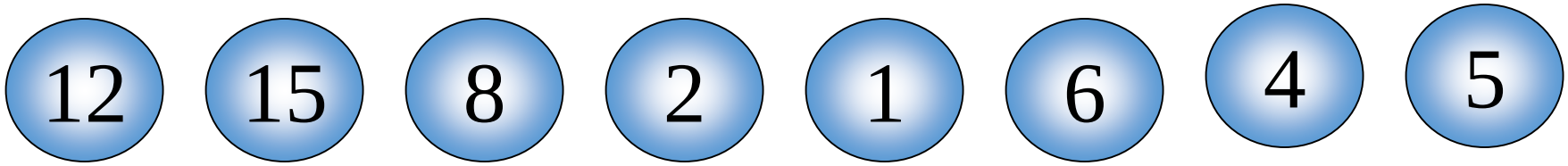


i=1

3.2.5 HeapSort

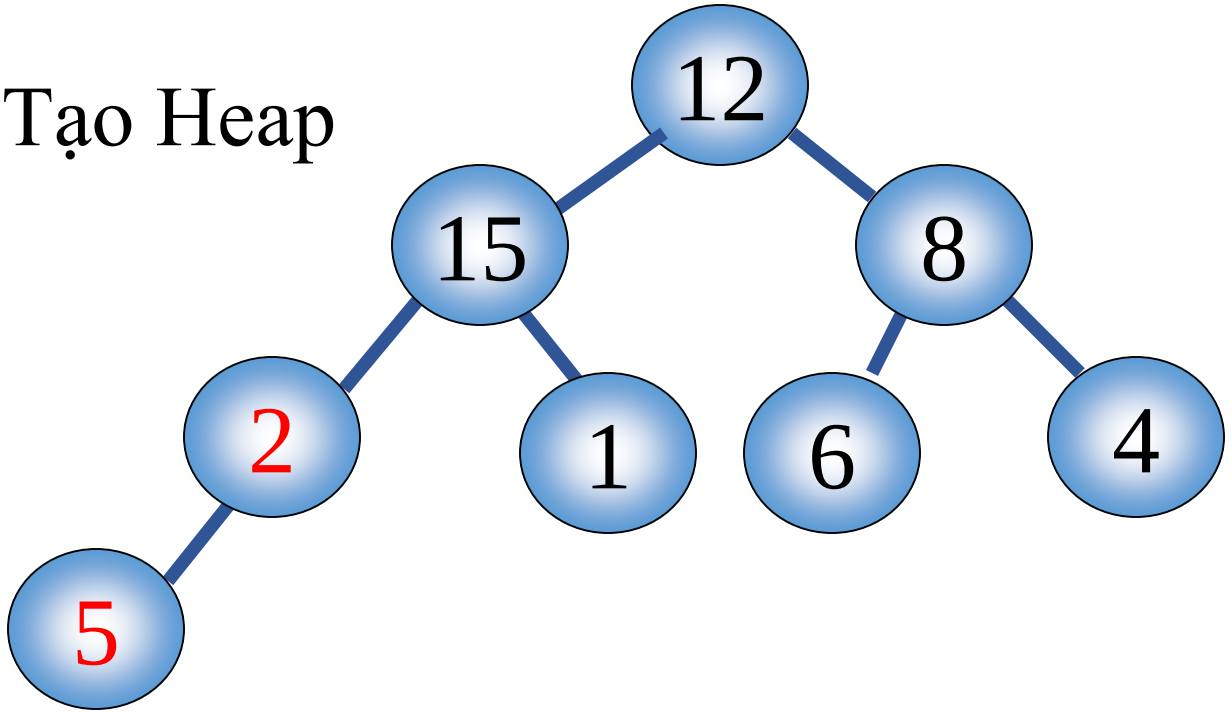
Bước 1: Tạo Heap



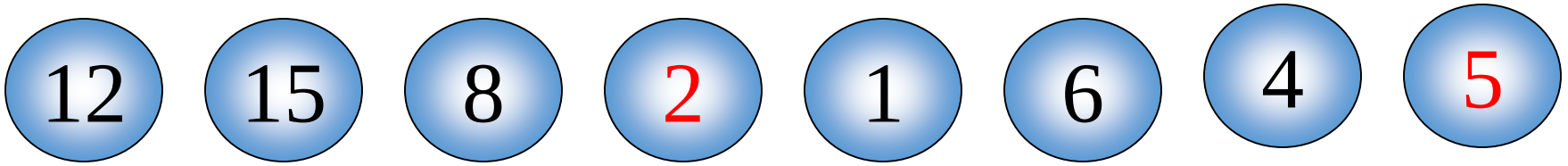


3.2.5 HeapSort

Bước 1: Tạo Heap

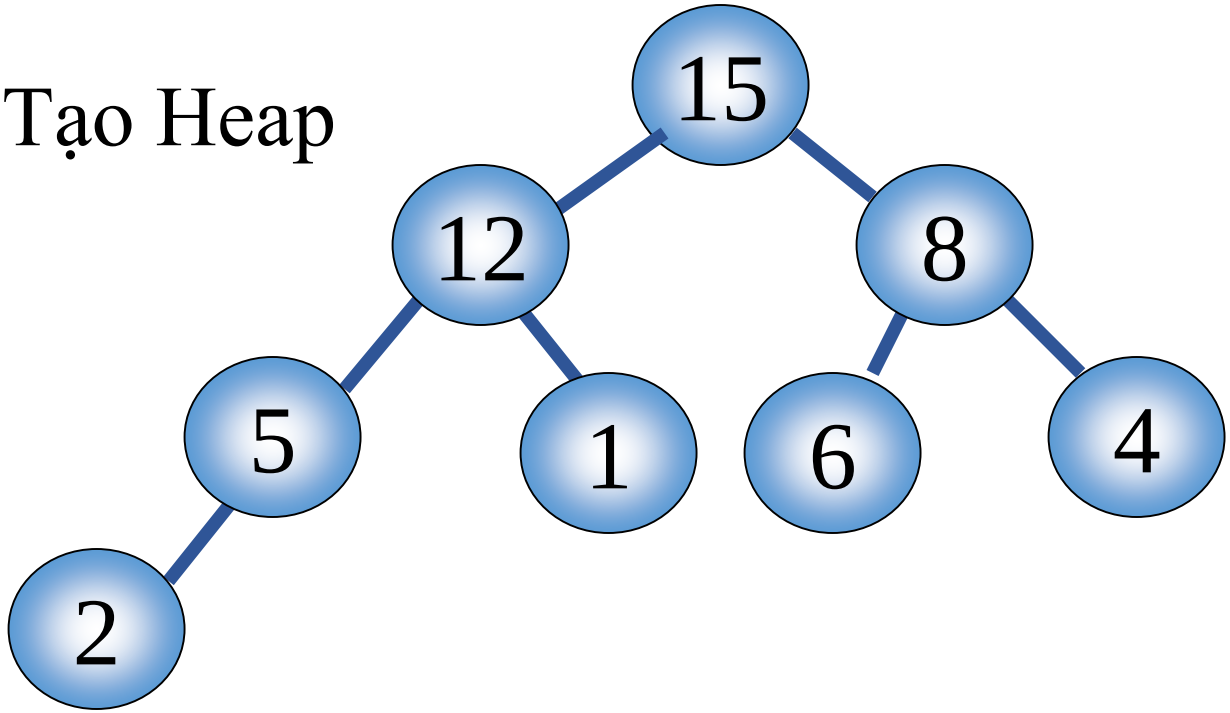


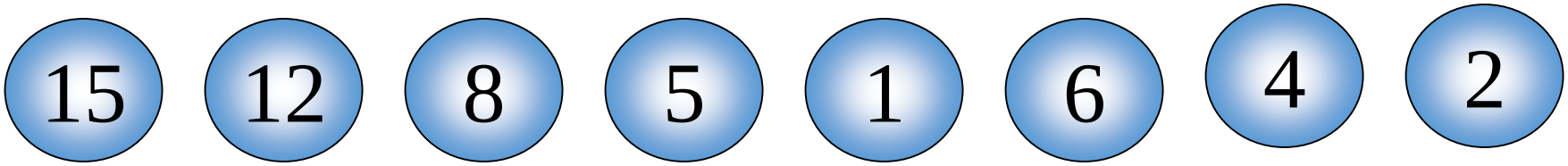
Lan truyền việc hiệu chỉnh



3.2.5 HeapSort

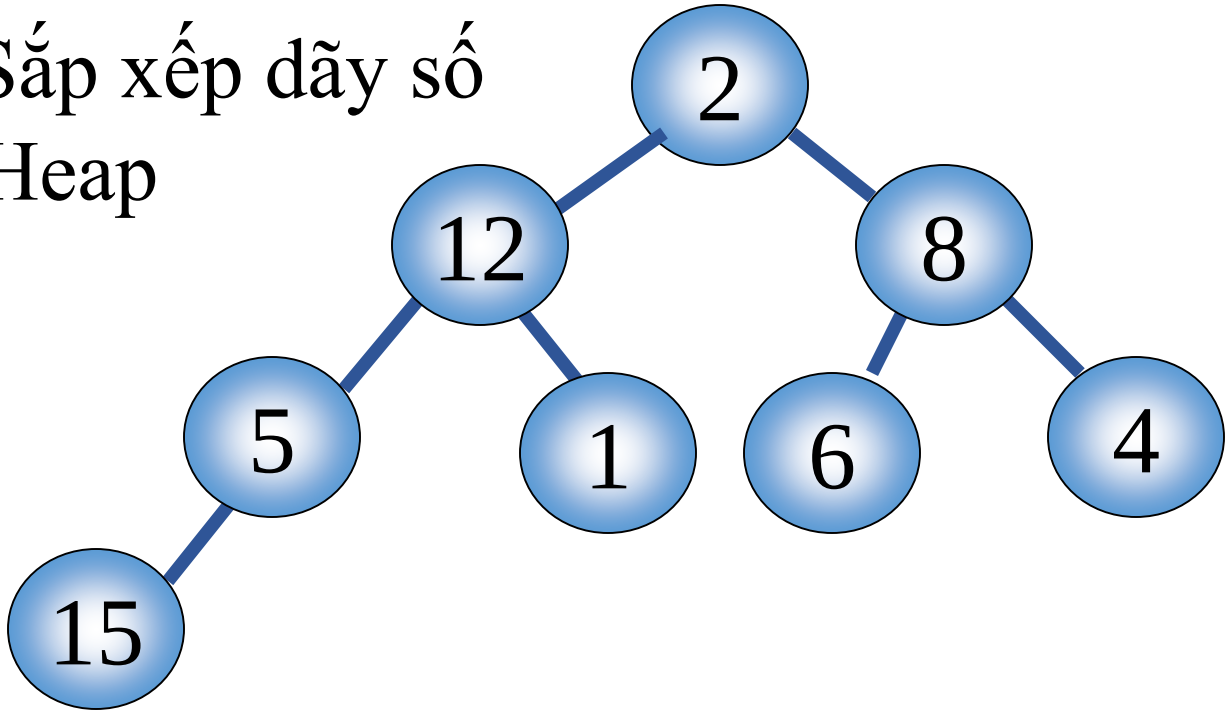
Bước 1: Tạo Heap

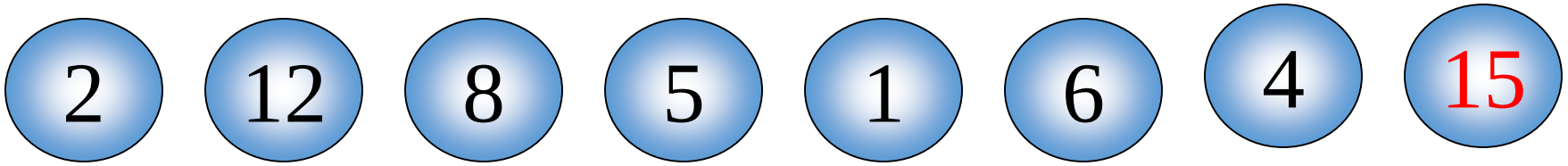




3.2.5 HeapSort

Bước 2: Sắp xếp dãy số dựa trên Heap

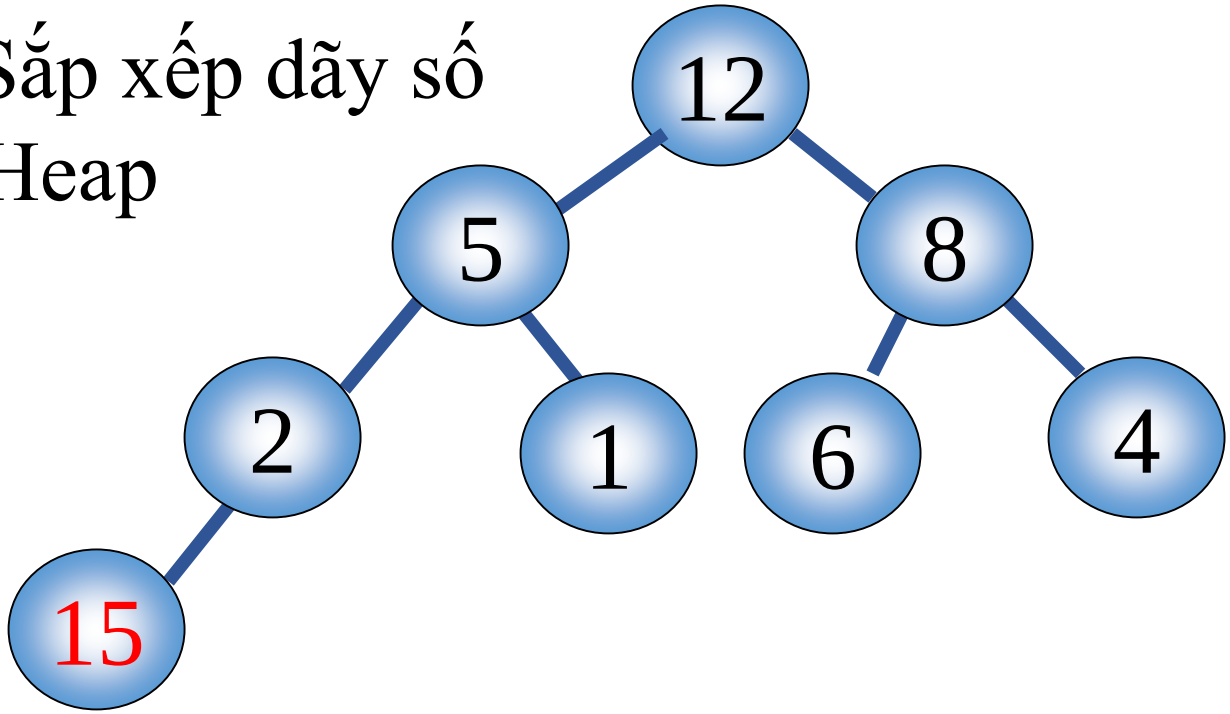


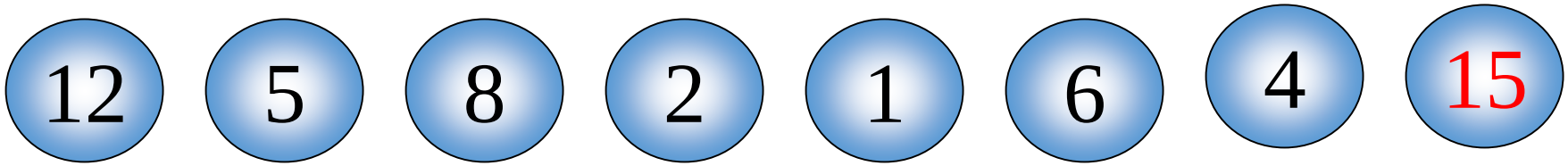


r=7

3.2.5 HeapSort

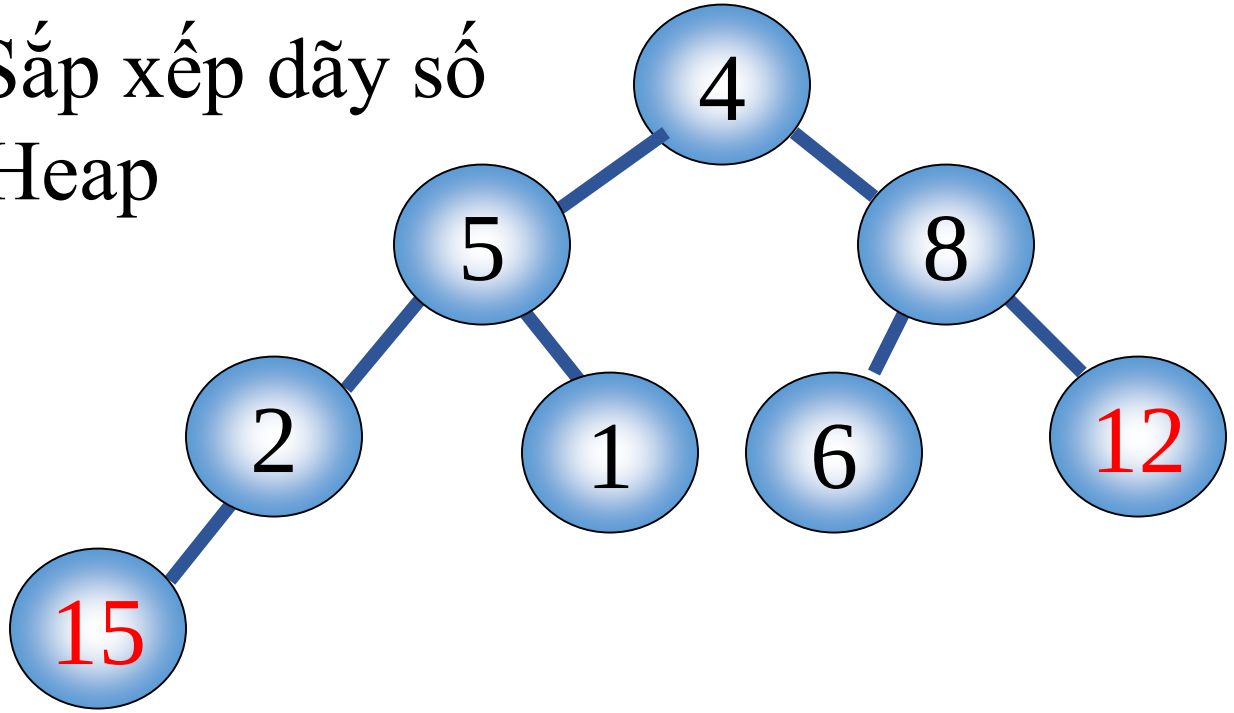
Bước 2: Sắp xếp dãy số dựa trên Heap

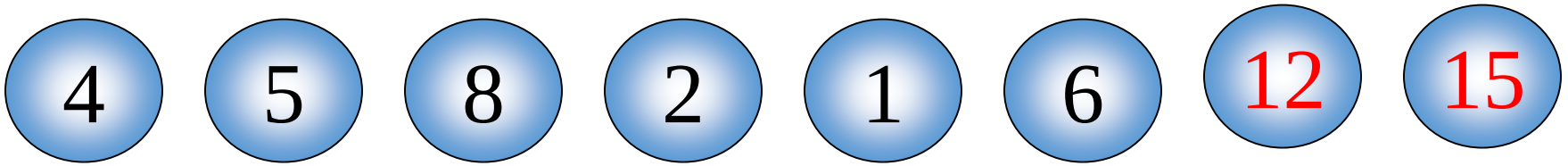




3.2.5 HeapSort

Bước 2: Sắp xếp dãy số dựa trên Heap

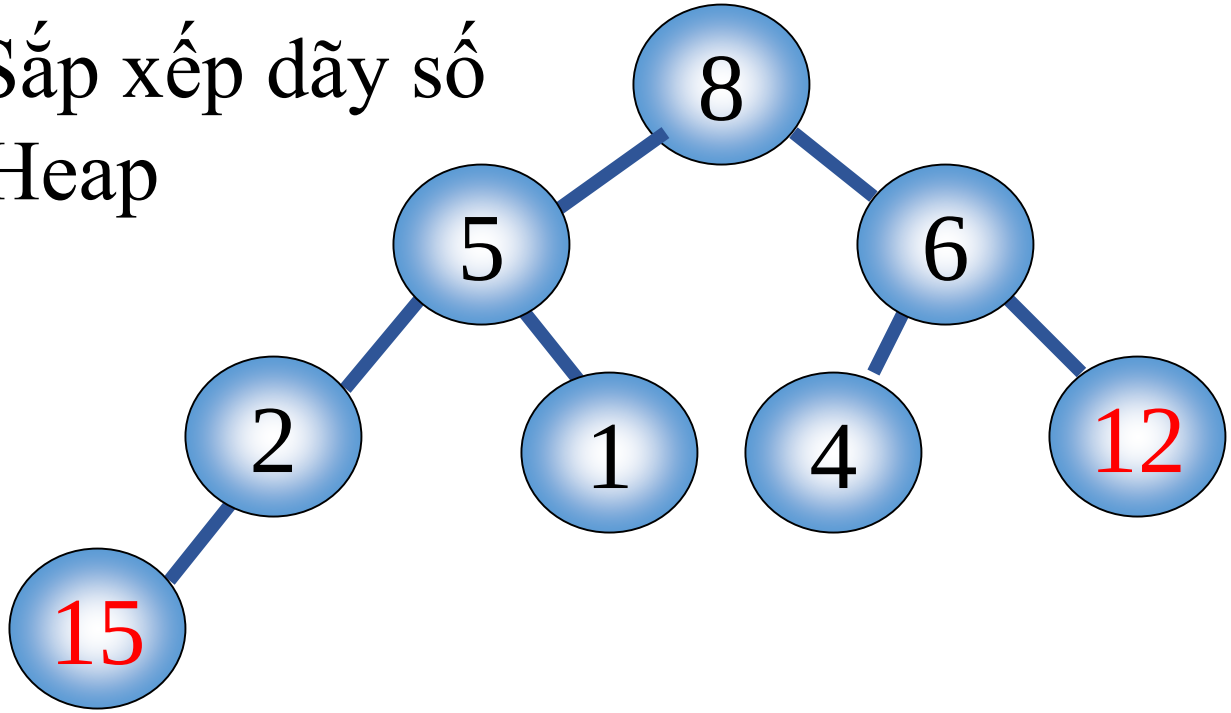


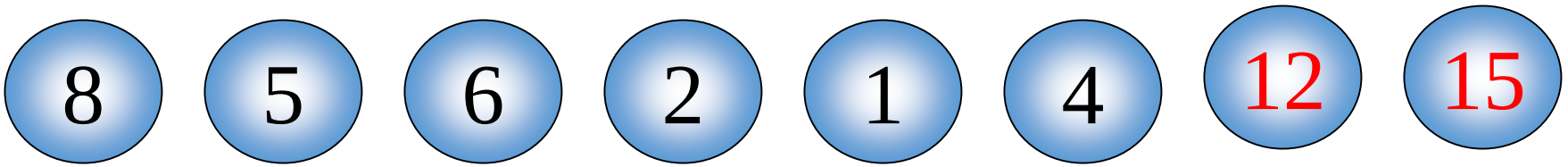


r=6

3.2.5 HeapSort

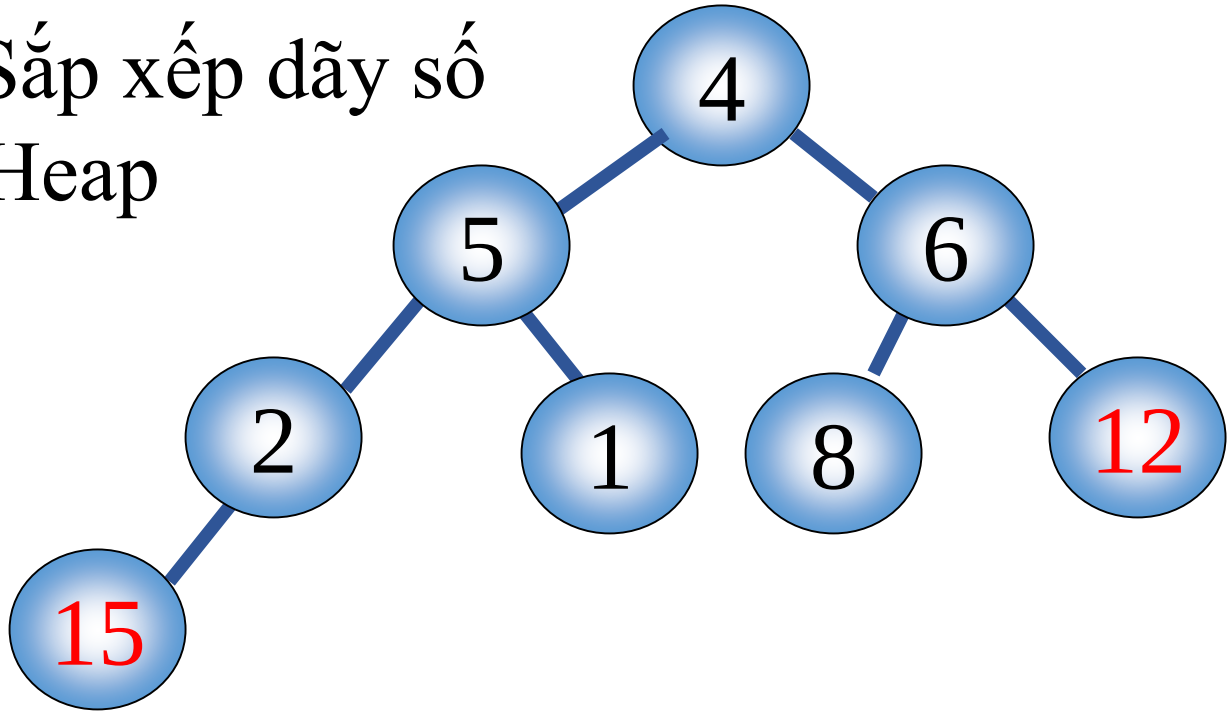
Bước 2: Sắp xếp dãy số dựa trên Heap

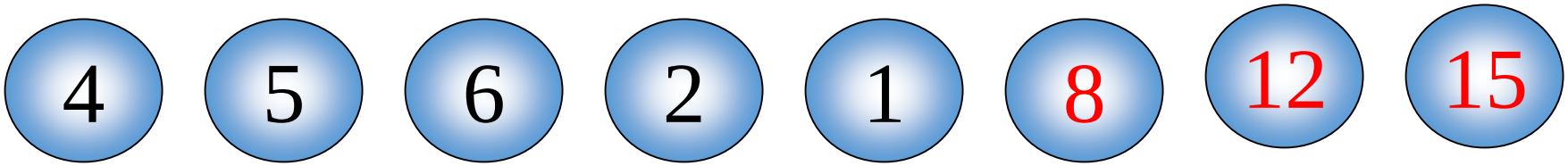




3.2.5 HeapSort

Bước 2: Sắp xếp dãy số dựa trên Heap

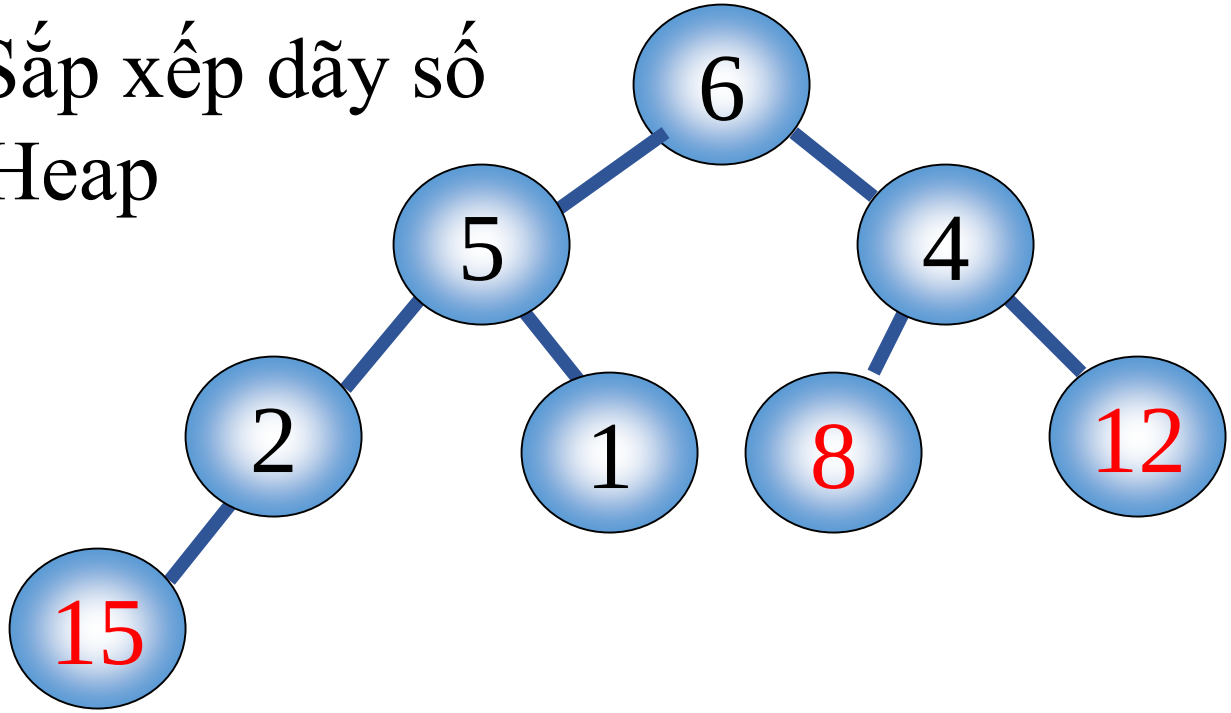


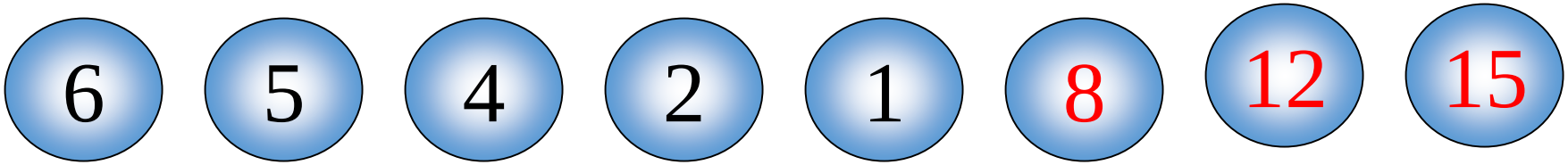


r=5

3.2.5 HeapSort

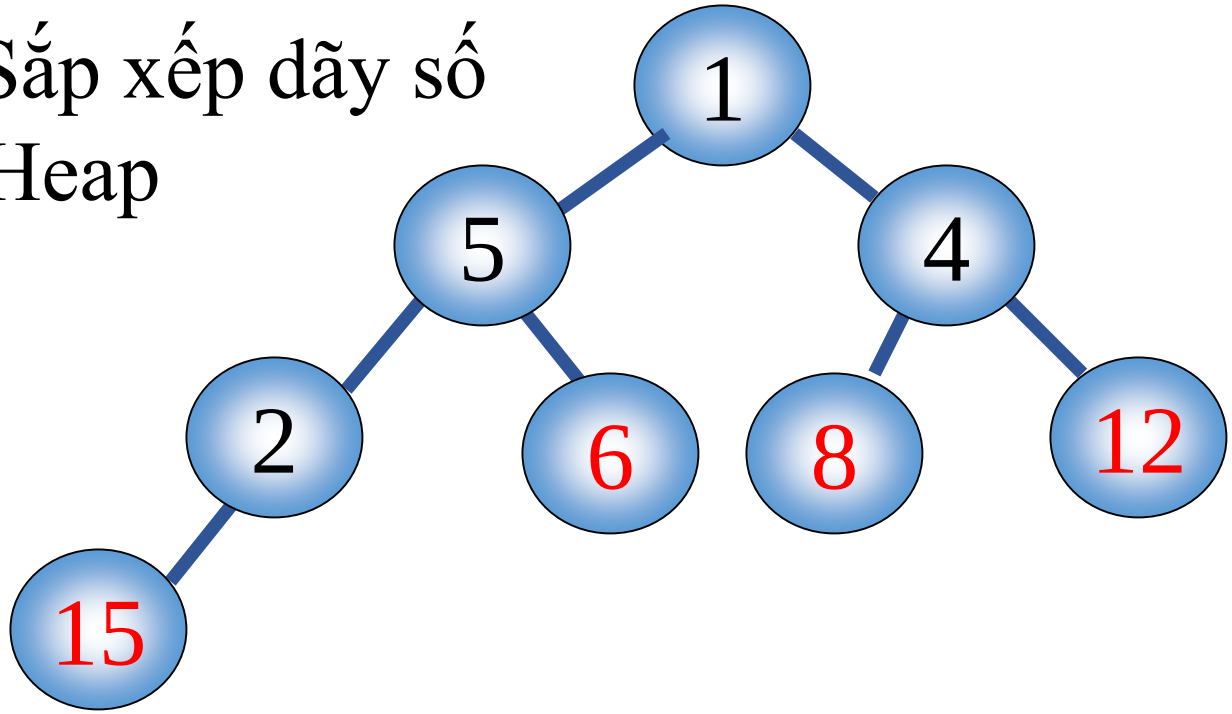
Bước 2: Sắp xếp dãy số dựa trên Heap

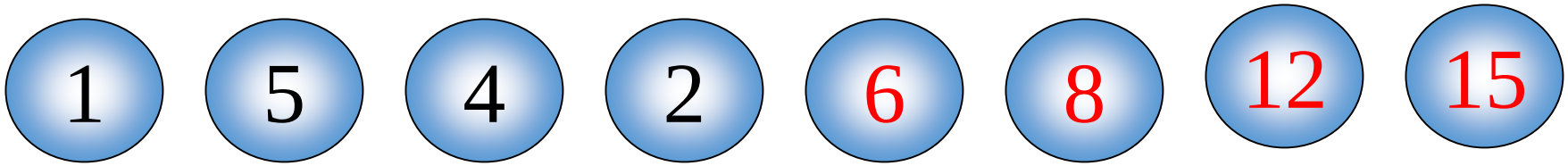




3.2.5 HeapSort

Bước 2: Sắp xếp dãy số dựa trên Heap





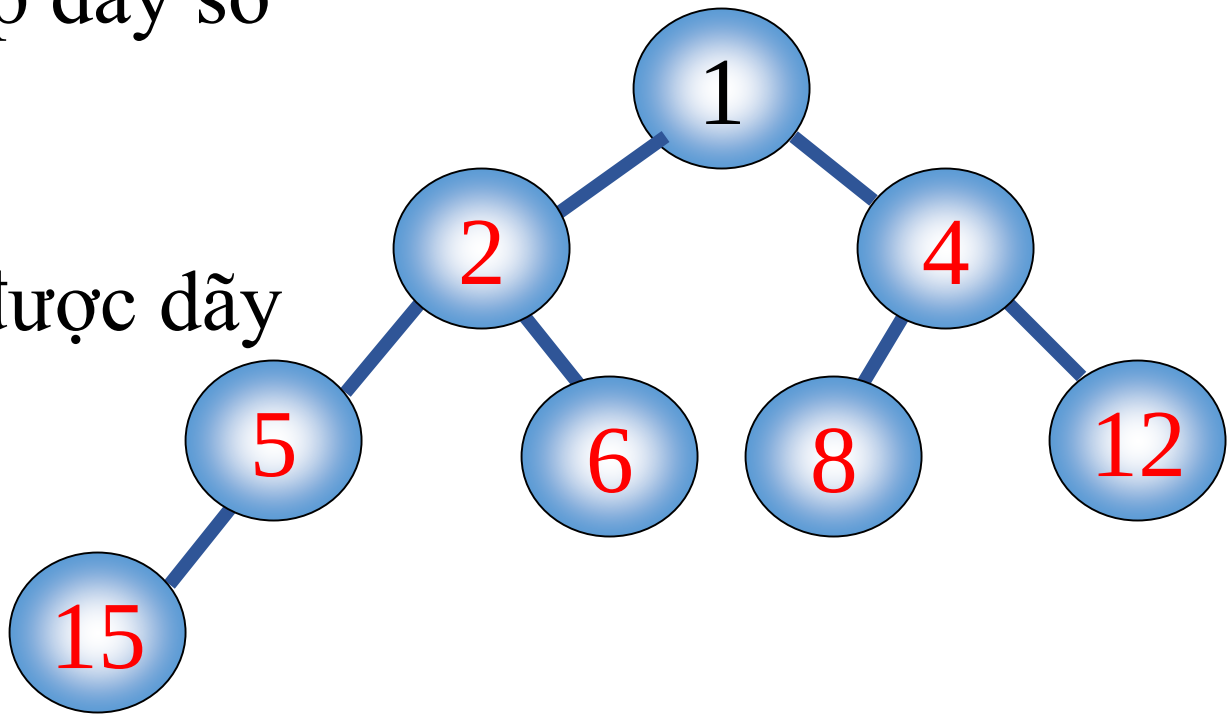
r=4

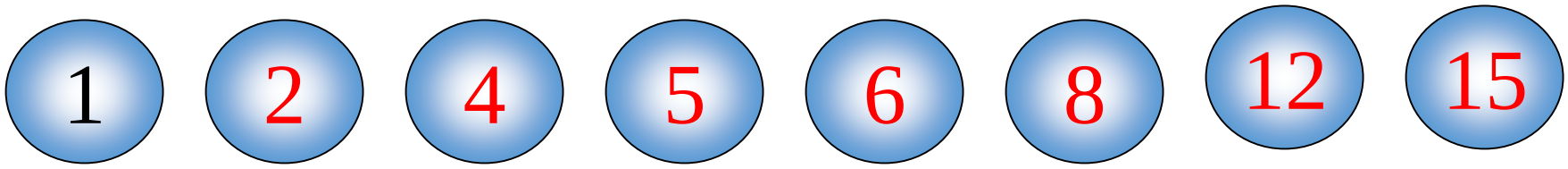
3.2.5 HeapSort

Bước 2: Sắp xếp dãy số dựa trên Heap

Làm tương tự

Cuối cùng thu được dãy đã sắp xếp





r=1 Bài toán kết thúc

3.2.5 HeapSort

Ưu điểm	Nhược điểm
Trong trường hợp xấu nhất tốt hơn Quick Sort.	Trường hợp trung bình chạy chậm hơn so với Quick Sort.
Thời gian sắp xếp là $O(n*log_2 n)$ trong mọi trường hợp.	Chạy chậm hơn Quick Sort, Merge Sort.
Không dùng đệ quy.	
Mảng đã được sắp xếp một phần thì chạy nhanh hơn Quick Sort.	

Câu hỏi củng cố bài

1. Trong trường hợp tốt nhất, thời gian chạy của giải thuật sắp xếp chèn (Insertion Sort) là gì?

A. O (n^2)

B. O ($n\log_2 n$)

C. O (n)

D. O ($\log_2 n$)

Câu hỏi củng cố bài

2. Sắp xếp lựa chọn (Selection Sort) là phương pháp:

A. Chọn khoá nhỏ nhất trong dãy khoá k[i], k[i+1], ...,k[n] đổi chỗ cho k[i] (1<=i<=n)

B. Tìm vị trí thích hợp cho khoá k để chèn vào dãy đã được sắp k[1], k[2], ...,k[i-1] đã được sắp (i=2,3,...,n)

C. Duyệt từ đáy lên đỉnh, nếu gặp hai khoá ngược thứ tự thì đổi chỗ của chúng cho nhau

D. Chọn một khoá ngẫu nhiên làm chốt. Mọi phần tử nhỏ hơn khoá chốt được xếp vào đầu dãy, mọi phần tử lớn hơn khoá chốt đều nằm ở cuối dãy.

Câu hỏi củng cố bài

3. Sắp xếp nổi bọt (Bubble Sort) là phương pháp:

A. Chọn khoá nhỏ nhất trong dãy khoá k[i], k[i+1], ...,k[n] đổi chỗ cho k[i] (1<=i<=n)

B. Tìm vị trí thích hợp cho khoá k để chèn vào dãy đã được sắp k[1], k[2], ...,k[i-1] đã được sắp (i=2,3,...,n)

C. Duyệt từ đáy lên đỉnh, nếu gặp hai khoá ngược thứ tự thì đổi chỗ của chúng cho nhau

D. Chọn một khoá ngẫu nhiên làm chốt. Mọi phần tử nhỏ hơn khoá chốt được xếp vào đầu dãy, mọi phần tử lớn hơn khoá chốt đều nằm ở cuối dãy.

Câu hỏi củng cố bài

4. Trong một lần lặp của vòng lặp bên ngoài của đoạn chương trình dưới đây, vòng lặp bên trong thực hiện bao nhiêu lần?

```
1.      for (i=1; i<=n-1; i++)
2.          for (j=n; j>=i+1; j--)
3.              if (a[j-1]> a[j])
4.              {   temp = a[j-1];
5.                  a[j-1] = a[j];
6.                  a[j] = temp;
7.              }
```

A. n - i

B. n - i + 1

C. i - n + 1

D. n(i - 1)

Câu hỏi củng cố bài

5. Cho dãy số sau: 3, 9, 4, 1, 5, 8, 0, 6, 2, 7

Kết quả của giải thuật sắp xếp Selection Sort để sắp xếp dãy số trên theo thứ tự tăng dần tại thời điểm i=4.

A. 0 1 2 3 4 5 9 6 8 7

B. 0 1 4 9 5 8 3 6 2 7

C. 0 1 2 9 5 8 3 6 4 7

D. 0 1 2 3 4 8 9 6 5 7

Multiple Choice

Câu hỏi củng cố bài

6. Cho dãy số sau: 3, 9, 4, 1, 5, 8, 0, 6, 2, 7

Kết quả của giải thuật sắp xếp chèn Insertion Sort để sắp xếp dãy số trên theo thứ tự tăng dần tại thời điểm i = 3.

A. 3, 4, 9, 1, 5, 8, 0, 6, 2, 7

B. 1, 3, 4, 9, 5, 8, 0, 6, 2, 7

C. 0, 1, 3, 4, 5, 8, 9, 6, 2, 7

D. 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Câu hỏi củng cố bài

6. Cho dãy số sau: 3, 9, 4, 1, 5, 8, 0, 6, 2, 7

Kết quả của giải thuật sắp xếp Bubble Sort để sắp xếp dãy số trên theo thứ tự tăng dần tại thời điểm i = 2.

A. 1 3 4 5 8 9 0 6 2 7

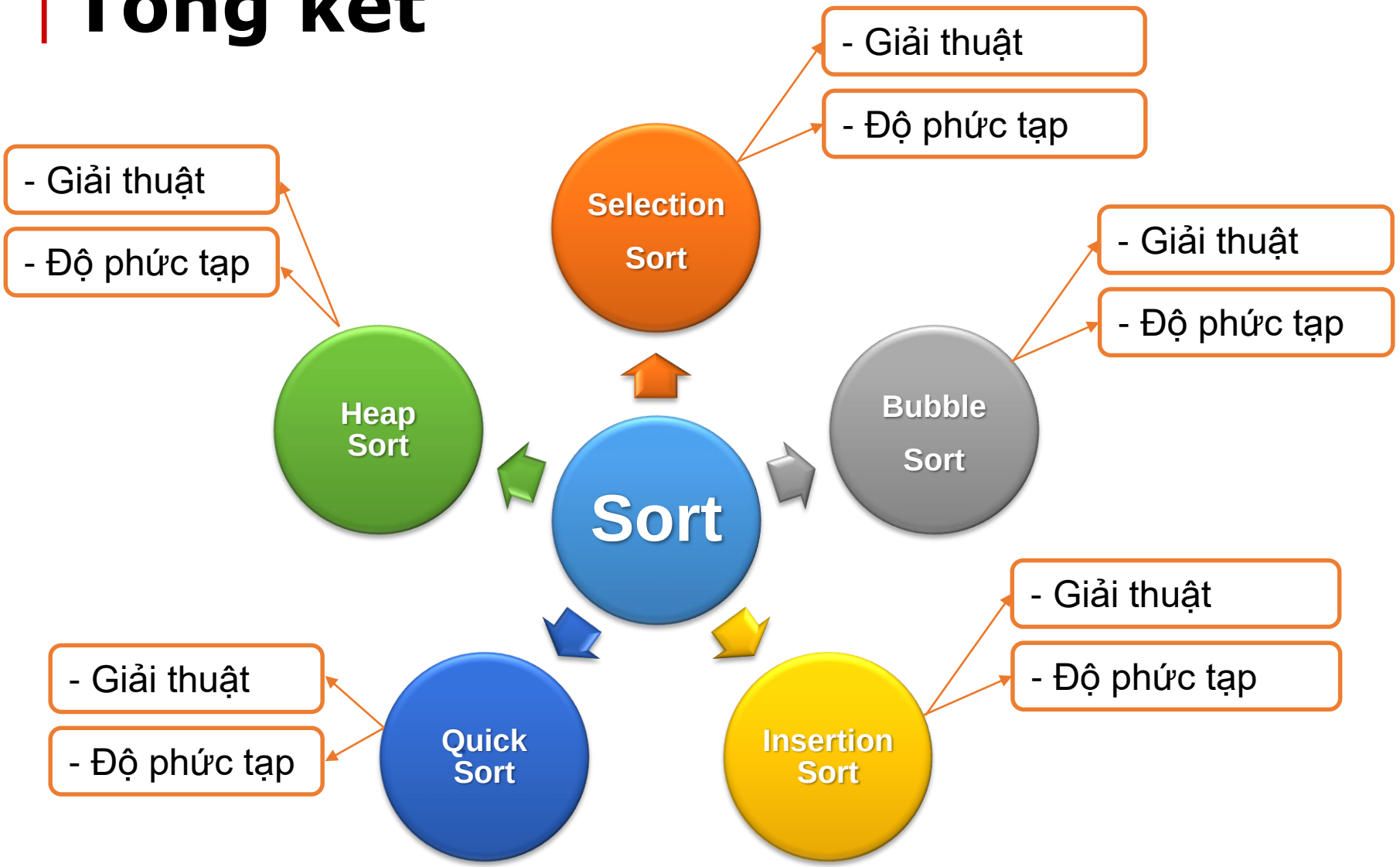
B. 1 3 0 4 2 5 6 7 8 9

C. 0 1 2 3 9 4 5 6 8 7

D. 0 1 2 3 4 9 8 6 5 7

Multiple Choice

Tổng kết


Sort
Selection Sort
- Giải thuật
- Độ phức tạp
Bubble Sort
- Giải thuật
- Độ phức tạp
Insertion Sort
- Giải thuật
- Độ phức tạp
Quick Sort
- Giải thuật
- Độ phức tạp
Heap Sort
- Giải thuật
- Độ phức tạp

Bài tập

Bài 1.

42 23 65 11 87 36 94 50 79 68

a) Viết giải thuật sắp xếp Selection Sort để sx dãy khoá theo thứ tự giảm dần

b) Minh hoạ với dãy khoá trên

Bài 2. Cho dãy khoá như trong bài 1

a) Viết giải thuật sắp xếp Bubble Sort để sắp xếp dãy khoá theo thứ tự giảm dần

b) Minh hoạ với dãy khoá trên

Bài tập

Bài 3. Cho dãy khoá

42 23 65 11 87 36 94 50 79 68

a) Viết giải thuật sắp xếp Insertion Sort để sx dãy khoá theo thứ tự giảm dần

b) Minh hoạ với dãy khoá trên

Bài 4. Cho dãy khoá sau

42 23 65 11 68 36 94 50 79 87

a) Viết giải thuật sắp xếp Quick Sort để sắp xếp dãy khoá theo thứ tự giảm dần

b) Minh hoạ với dãy khoá trên

Bài tập

Bài 5. Cho dãy khoá

55 22 66 11 88 33 99 44 77 60

a) Viết giải thuật sắp xếp Quick Sort để sx dãy khoá theo thứ tự tăng dần

b) Minh hoạ với dãy khoá trên

Bài 6. Cho dãy khoá như trong bài 5

a) Viết giải thuật sắp xếp Heap Sort để sắp xếp dãy khoá theo thứ tự tăng dần

b) Minh hoạ với dãy khoá trên

Bài tập

Bài 7. Cho dãy khoá

42 23 80 11 87 36 94 55 79 68

a) Viết giải thuật sắp xếp Heap Sort để sx dãy khoá theo thứ tự tăng dần

b) Minh hoạ với dãy khoá trên

Bài 8.

Viết chương trình C++. Nhập vào một mảng gồm n phần tử nguyên rồi lựa chọn thực hiện các thao tác sắp xếp trong bài học và in kết quả.

Tài liệu tham khảo

[1]. Giáo trình Cấu trúc dữ liệu và giải thuật – Lê Văn Vinh, NXB Đại học quốc gia TP HCM, 2013

[2]. Cấu trúc dữ liệu & thuật toán, Đỗ Xuân Lôi, NXB Đại học quốc gia Hà Nội, 2010.

[3]. Trần Thông Quế, *Cấu trúc dữ liệu và thuật toán (phân tích và cài đặt trên C/C++)*, NXB Thông tin và truyền thông, 2018

[4]. Robert Sedgewick, *Cẩm nang thuật toán*, NXB Khoa học kỹ thuật, 2004 .

[5]. PGS.TS Hoàng Nghĩa Tý, *Cấu trúc dữ liệu và thuật toán*, NXB xây dựng, 2014