CHƯƠNG 9: Sắp xếp

9.1 Bubble Sort:

- **Ý tưởng :** Gọi a[i...j] là các phần tử a[i], a[i+1], ..., a[j]. Chuyển phần tử nhỏ nhất của a[0...n-1] tới a[0], sau đó Chuyển phần tử nhỏ nhất của a[1...n-1] tới a[1], . . ., chuyển phần tử nhỏ nhất của a[n-2...n-1] tới a[n-2]
- **Algorithm :**

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

Độ phức tạp : $O(n^2)$

Ví dụ Bubble Sort:

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

i	0	1	2	3	4
	7	2	5	4	3
0					

BT :

1) Định nghĩa hàm BubbleSort
2) Viết chương trình sắp xếp dùng hàm BubbleSort

Complex Bubble Sort: Một số qui ước :

- Mỗi lần kiểm tra điều kiện của một vòng lặp sẽ tốn một khoảng thời gian t. Nếu vòng lặp thực hiện n lần lặp (các lệnh trong thân vòng lặp thực hiện n lần) thì thời gian thực hiện vòng lặp (không kể thời gian thực hiện các lệnh trong thân vòng lặp) sẽ là (n+1)t.

Ví dụ :

```
for ( i = 1 ; i <= n ; i++ )
```

thời gian cho toàn bộ vòng lặp là (n+1)t.

- if (C) {. . .} : Số lần kiểm tra điều kiện C được xem là 1. Nếu C gồm nhiều biểu thức luận lý thì xem thời gian thực hiện C là t là tổng thời gian kiểm tra tất cả các biểu thức trong C.
- Lệnh gán *x = biểu thức* có thời gian thực hiện được tính tương tự như tính cho C ở trên

Complex Bubble Sort:

- t_1 : thời gian cho một lần kiểm tra điều kiện của for(i=0; . . .)
- t_2 : thời gian cho một lần kiểm tra điều kiện của for(j=n-1; . . .)
- t_3 : thời gian cho một lần kiểm tra điều kiện của if()
- t_4 : thời gian cho một lần lệnh gán ở (1)

```
for ( i = 0 ; i < n-1 ; i++ )
        for ( j = n-1 ; j > i ; j-- )
                if ( a[j] < a[j-1] )
                { t = a[j] ; a[j] = a[j-1]); a[j-1]=t;}  (1)
```

Complex Bubble Sort:

Lập bảng cho trường hợp tốt nhất (Các lệnh ở (1) không được thực hiện. **Trong trường hợp nào ?**):

Lệnh	Số lần
for (i=0)	n
for (j=n-1)	$\sum_{i=0}^{n-2}[((n-1)-(i+1)+1)+1]$ $= \sum_{i=0}^{n-2}(n-i) = n(n+1)/2 + 1$
if ()	$\sum_{i=0}^{n-2}(n-i) = n(n+1)/2 + 1$

Complex :

$$T(n) = nt_1 + (n(n+1)/2 + 1)t_2 + (n(n+1)/2 + 1)t_3$$

$$= n^2(t_2+t_3)/2 + n(2t_1 + t_2 + t_3)/2 + t_2 + t_3$$

$$= O(n^2)$$

Complex Bubble Sort:

Lập bảng cho trường hợp xấu nhất (Các lệnh ở (1) luôn được thực hiện. **Trong trường hợp nào ?**):

Lệnh	Số lần
for (i=0)	n
for (j=n-1)	$\sum_{i=0}^{n-2}[((n-1)-(i+1)+1)+1]$ $= \sum_{i=0}^{n-2}(n-i) = n(n+1)/2 + 1$
if ()	$\sum_{i=1}^{n-1}(n-i) = n(n+1)/2 + 1$
Lệnh (1)	$3(n(n+1)/2 + 1)$

Complex :

$T(n) = nt_1 + (n(n+1)/2 + 1)t_2 + (n(n+1)/2 + 1)t_3 + 3(n(n+1)/2 + 1)t_4$

$= n^2(t_2+t_3)/2 + n(2t_1 + t_2 + t_3)/2 + t_2 + t_3 + 3(n(n+1)/2 + 1)t_4$

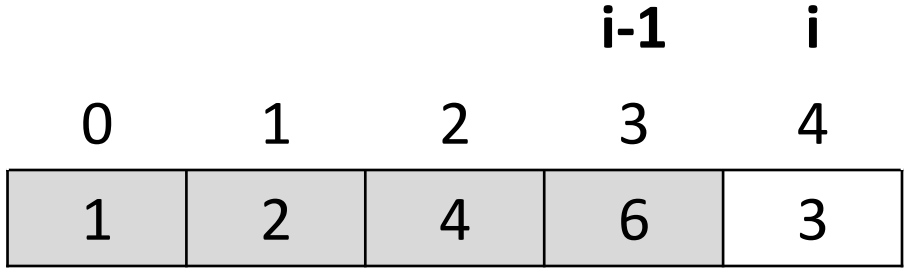
$= O(n^2)$

9.2 Insertion Sort :

- **Ý tưởng :** Giả sử mảng a[0...i-1] đã có thứ tự tăng dần. Tìm j bé nhất , $0 \le j \le i$, sao cho $a[i] \le a[j]$, chèn a[i] vào vị trí a[j].
- **Algorithm :**

```
for ( i = 1 ; i < n ; i++ ) {
        j = i;  t = a[j];
        while ( j > 0 && t < a[j-1] ) {
                a[j] = a[j-1];  j--
        }
        a[j] = t;
}
```

- **Độ phức tạp :** $O(n^2)$



Ví dụ Insertion Sort :

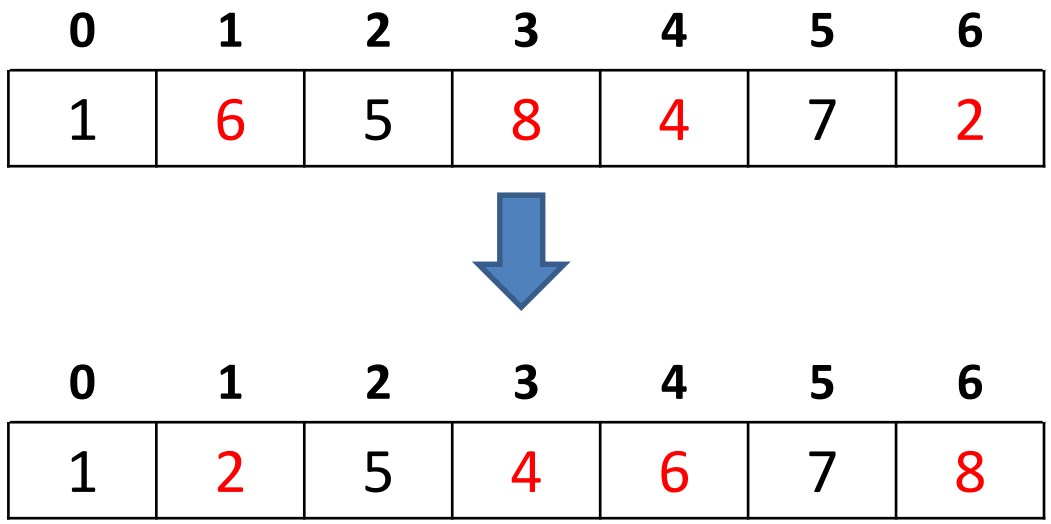
```
for ( i = 1 ; i < n ; i++ ) {
        j = i;  t = a[j];
        while ( j > 0 && t < a[j-1] ) {
                a[j] = a[j-1];  j--
        }
        a[j] = t;
}
```

			i-1	i		
0	**1**	**2**	**3**	**4**	**5**	**6**
1	2	4	6	3	7	5

i	**0**	**1**	**2**	**3**	**4**	**5**	**6**
	4	7	6	5	3	2	1
1							
2							

Bài tập Insertion Sort : Sắp xếp các số chẵn theo chiều tăng dần (vẫn giữ nguyên vị trí các phần tử).

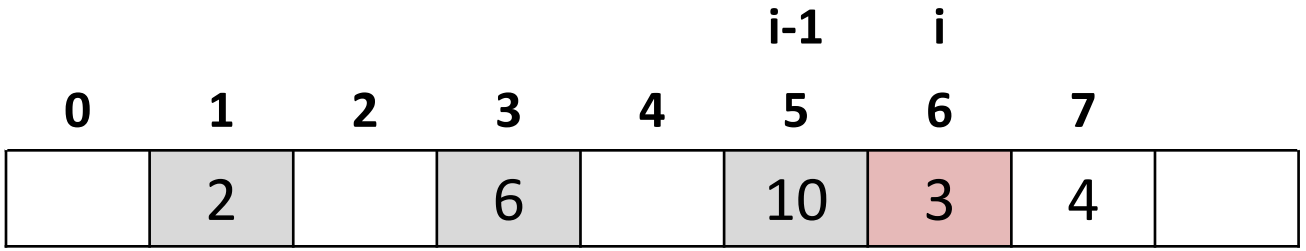
```
for ( i = 1 ; i < n ; i++ ) {
        j = i;    t = a[j];
        while ( j > 0 && t < a[j-1] ) {
                a[j] = a[j-1];  j--
        }
        a[j] = t;
}
```



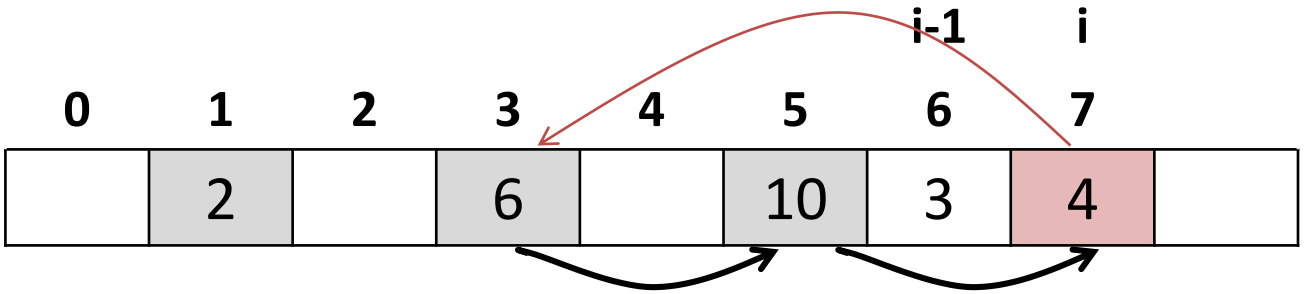
Bài tập Insertion Sort

- **Ý tưởng :** Giả sử các phần tử chẵn trong mảng a[0...i-1] đã có thứ tự tăng dần.

- Nếu a[i] có giá trị lẻ thì SKIP (không thực hiện gì cả).



- Nếu a[i] có giá trị chẵn tìm j bé nhất , $0 \le j \le i$, sao cho a[j] chẵn và $a[i] \le a[j]$, chèn a[i] vào vị trí a[j].



Bài tập Insertion Sort : Sắp xếp các số chẵn theo chiều tăng dần (vẫn giữ nguyên vị trí các phần tử).

OLD :

```
for ( i = 1 ; i < n ; i++ )
{
    j = i; t = a[j];
    while ( j > 0 && t < a[j-1] )
    {
          a[j] = a[j-1];  j--
    }
    a[j] = t;
}
```

NEW :

```
for ( i = 1 ; i < n ; i++ )
{
      j = i; t = a[j];
      while ( j > 0 )
      {
           if (t < a[j-1] )
           {
                a[j] = a[j-1];  j--;
           }
           else break;
      }
       a[j] = t;
}
```

Bài tập Insertion Sort

- Nếu a[i] có giá trị chẵn tìm j bé nhất, $0 \le j \le i$, sao cho a[j] chẵn và $a[i] \le a[j]$, chèn a[i] vào vị trí a[j].

```
for ( i = 1 ; i < n ; i++ )
    if (a[i] % 2==0)
    {
        j = i; t = a[j];
        while ( j > 0 )
        {
            k = Tìm chỉ số (< j ) của phần tử chẵn gần j nhất;
            ( k < 0, nếu không tìm thấy.)
            if (k >=0 && t < a[k])  { a[j] = a[k]; j = k ;}
            else break;
        }
        a[j] = t;
    } // if (a[i]%2==0)
```

Bài tập Insertion Sort và các thuật toán khác :

1. Sắp xếp các số lẻ theo chiều tăng dần (vẫn giữ nguyên vị trí các phần tử).
2. Sắp xếp các số lẻ theo chiều tăng dần, các số chẵn theo chiều giảm dần (vẫn giữ nguyên vị trí các phần tử).
3. Sắp xếp các số nguyên tố theo chiều tăng dần (vẫn giữ nguyên vị trí các phần tử).
4. Sắp xếp các số nguyên tố theo chiều tăng dần, các số chẵn theo chiều giảm dần (vẫn giữ nguyên vị trí các phần tử). Số 2 thuộc tập số nguyên tố.
5. Sắp xếp các số nguyên tố theo chiều tăng dần, các số Fibonacci theo chiều giảm dần (vẫn giữ nguyên vị trí các phần tử). Các số Fibonacci là số nguyên tố thì thuộc tập Fibonacci.

   Dãy fibonacci : 0, 1, 1, 2, 3, 5, 8, . . . ($a_n = a_{n-1} + a_{n-2}$)

   Số Fibonacci là số trong dãy Fibonacci.

6. Sắp xếp các số hoàn hảo (HH) theo chiều tăng dần, các số Fibonacci theo chiều giảm dần (6 = 1 + 2 +3, 6 là số hoàn hảo). Số HH là số Fibonacci thì thuộc tập Fibonacci.

9.3 Selection Sort :

- **Ý tưởng :** Tương tự **Bubble Sort**.
- **Algorithm :**

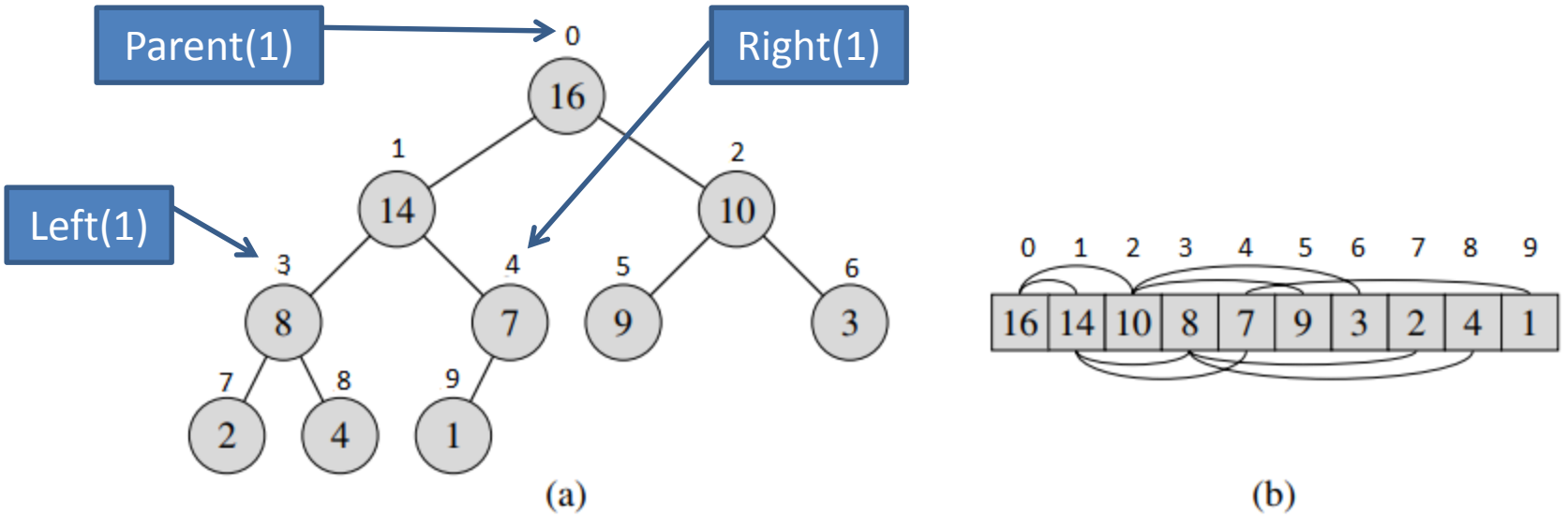
```
for ( i = 0 ; i < n-1 ; i++ ) {
        1. k = i ;
        2. for ( j = i+1 ; j < n ; j++ )
                3. if ( a[j] < a[k] ) k = j ;
        4. swap (a[i] ,a[k]);
}
```

- **Độ phức tạp :** $O(n^2)$

9.3 Heap Sort :

1. Các định nghĩa :

- Cấu trúc dữ liệu heap (nhị phân) là một mảng A các đối tượng được nhìn như một cây nhị phân.



- Gốc của cây là A[0].

- Nút (chỉ số) i có cha là **Parent(i)**, con trái là **Left(i)**, con phải là **Right(i)**.

- PARENT(i)

  if i lẻ return $\lfloor i/2 \rfloor$

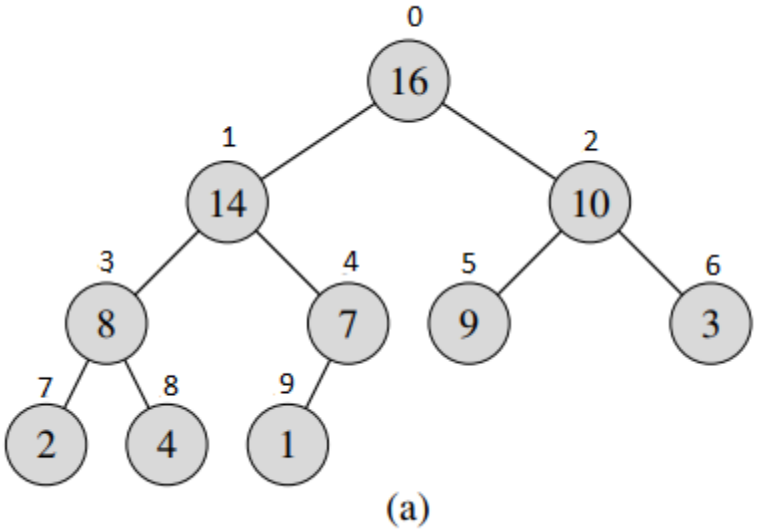
  else $\lfloor (i-1)/2 \rfloor$

- LEFT(i)

  return 2i + 1

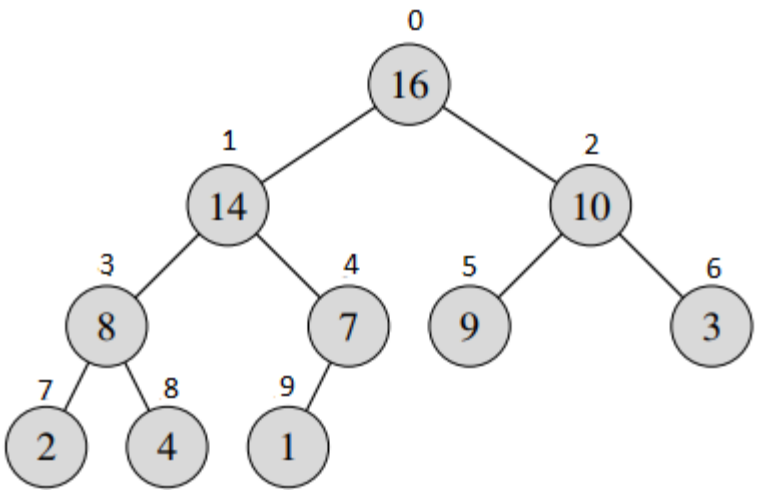
- RIGHT(i)

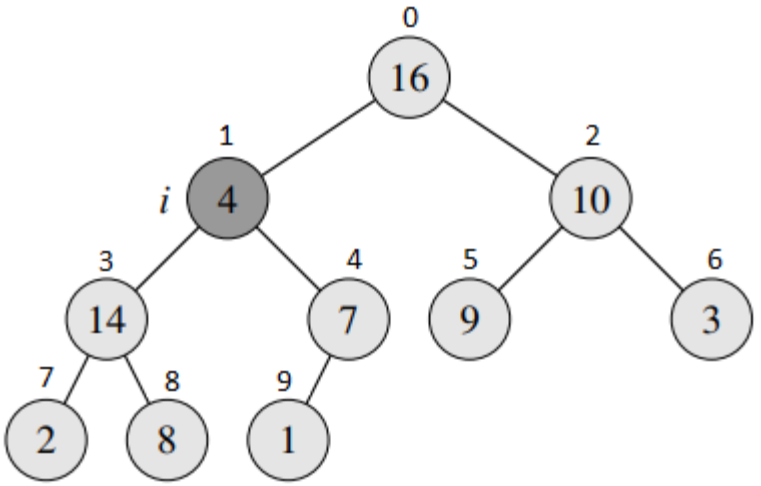
  return 2i + 2



(a)

- ***max-heap property*** : Mảng A là một max-heap nếu với mọi nút i khác gốc, A[PARENT(i)] ≥ *A*[i].



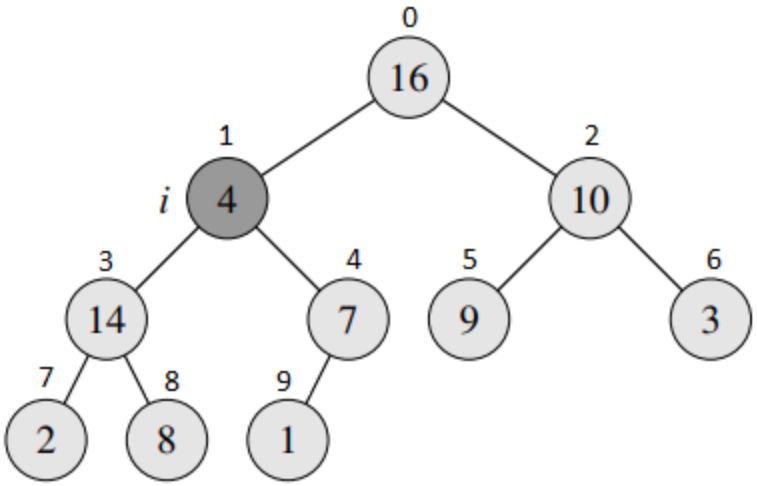


Bài tập : A = {23, 17, 14, 6, 13, 10, 1, 5, 7, 12} là một max heap ? Vẽ cây cho A.

2. Các thủ tục :

2.1 Thủ tục Max_Heap(A, i, heap_size) :

- **Giả sử :** Cây con trái và cây con phải của nút i thỏa *max-heap property* . Ví dụ i = 1 như hình dưới.



- Thủ tục sẽ đưa giá trị của nút i vào vị trí sao cho cây với gốc là nút i thỏa *max-heap property.*

Thủ tục Max_Heap(A, i, heap_size)

{

1. l = LEFT(i); r = RIGHT(i);

2. if ((l < heap_size) && A[l] > A[i])

largest = l;

3. else largest = i;

4. if ((r < heap_size) && A[r] > A[largest])

largest = r;

5. if (largest!= i)

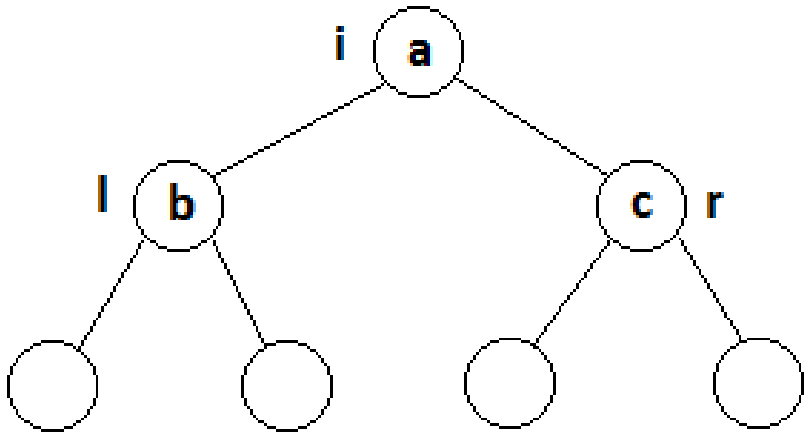
{ **6.** swap (A[i] , A[largest]);

7. Max_Heap (A, largest,

heap_size);

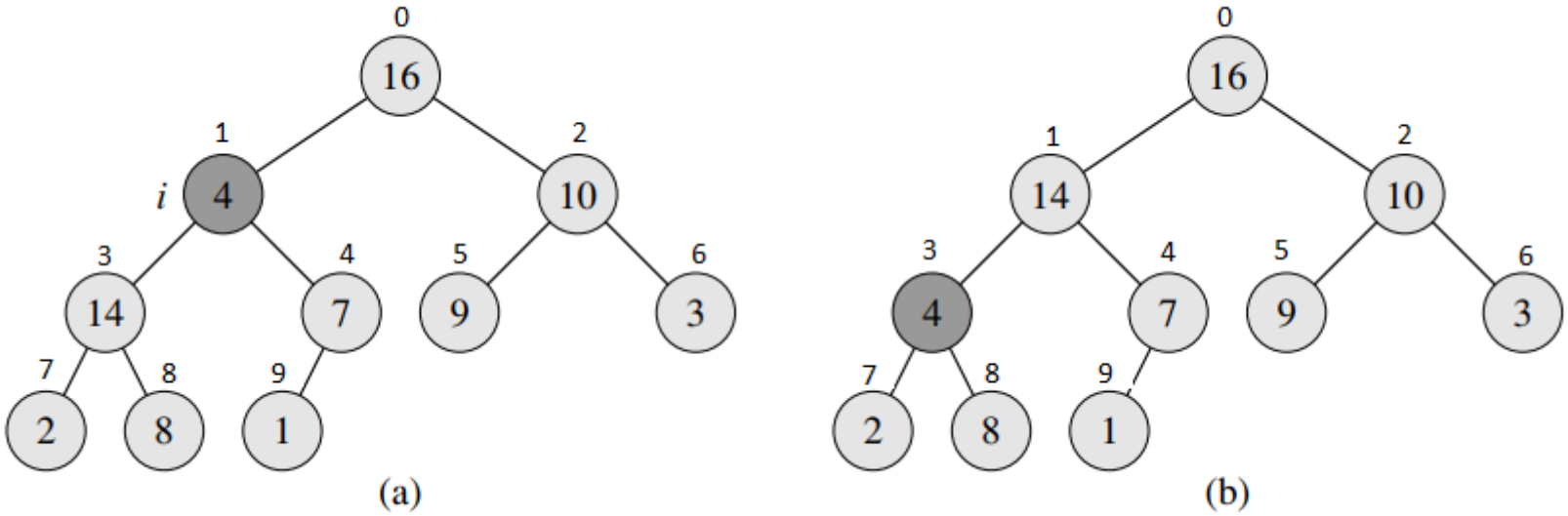
}

}



1. Nếu a < b < c thì hoán vị i và r
2. Nếu a < c < b thì hoán vị i và l
3. Giả sử 1. thực hiện. Nếu cây con gốc r không thỏa ***max-heap property*** thì gọi ***Thủ tục Max_Heap***

Ví dụ : ***Max_Heap(A, 1, 10)***



Bài tập : Hãy vẽ hình từng bước thực hiện ***Max_Heap(A, 2, 14)*** trên mảng A = {27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0} .

2.2 Thủ tục xây dựng heap : Cho mảng A. Chuyển các giá trị trong A sao cho A thỏa *max-heap property .*

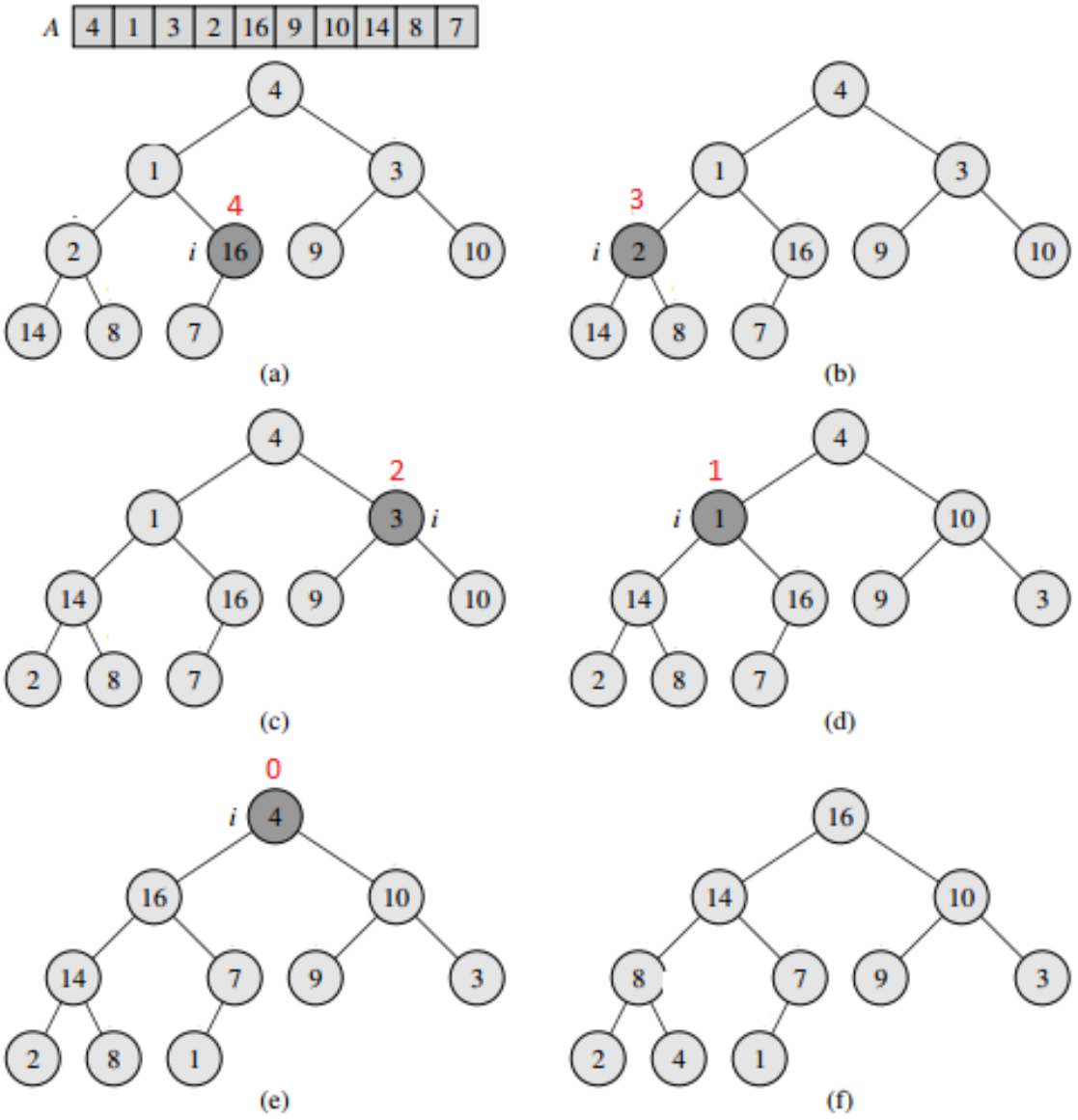
Build_Max_Heap(A, length)

```
{
        for(i =length/2-1 ;i >= 0; i--)
                Max_Heap(A, i, length );
}
```

- length : kích thước mảng A

Thủ tục xây dựng heap :

Ví dụ :



BT : Xây dựng heap cho mảng *A* = {5, 3, 17, 10, 84, 19, 6, 22, 9}

2.3 Thủ tục HeapSort(A, length)

```
{
        1. heap_size = length;
        2. Build_Max_Heap(A, length);
        3. for(i =length-1 ; i>=1; i--)
           {
                4. swap(A[0],A[i]);
                5. heap_size = heap_size - 1;
                6. Max_Heap (A, 0, heap_size);
           }
} :
```

- **Độ phức tạp :** O(n*lg(n))

9.4 Merge Sort :

```
MERGESORT(array a, int left, int right)
{
        if ( left < right )
        {
                1. mid = (left + right) / 2 ;
                2. MERGESORT(a, left, mid) ;
                3. MERGESORT(a, mid+1, right) ;
                4. MERGE(a, left, mid, right)
        }
}
```

Complex : T(n) = O(nlgn)

9.4 Merge Sort :

```
MERGE (array a, int left, int mid, int right) {
  1.  create new array b of size right-left+1;
  2. bcount = 0; lcount = left; rcount = mid+1;
  3. while ( (lcount <= mid) && (rcount <= right) ) {
        4. if ( a[lcount] <= a[rcount] ) b[bcount++] = a[lcount++];
        5. else  b[bcount++] = a[rcount++] ;
     } // end while
   7. if ( lcount > mid )
        8. while ( rcount <= right ) b[bcount++] = a[rcount++];
     else
        9. while ( lcount <= mid ) b[bcount++] = a[lcount++] ;
  10. for ( bcount = 0 ; bcount < right-left+1 ; bcount++ )
                    11. a[left+bcount] = b[bcount];
} // end MERGE
```

1	4	7	9	2	5	6	8

9.5 Quick Sort :

Ý chính của quick sort :

Divide: Chia mảng $A[p\,..\,r]$ thành hai mảng con $A[p\,..\,q-1]$ và $A[q+1\,..\,r]$ sao cho mỗi phần tử của $A[p\,..\,q-1]$ nhỏ hơn hay bằng $A[q]$, và A[q] nhỏ hơn hay bằng mỗi phần tử của $A[q+1\,..\,r]$. q được gọi là pivotindex.

Conquer: Sắp xếp $A[p\,..\,q-1]$ và $A[q+1\,..\,r]$ bằng cách gọi đệ qui hàm quicksort.

Combine: Không có.

Mảng $A[p\,..\,r]$ đã được sắp xếp.

Thuật toán :

```
QUICKSORT(A, p, r)
{
  1. if (p < r)
     {
        2. q ← PARTITION(A, p, r) ;
        3. QUICKSORT(A, p, q - 1) ;
        4. QUICKSORT(A, q + 1, r) ;
     }
}
main()
{
QUICKSORT(A, 0, length[A]-1);
}
```

Thuật toán :

```
PARTITION(A, p, r)
{
1. x = A[r] ;
2. i = p – 1 ;
3. for (j = p ;j <= r – 1; j++)
        4. if (A[ j] ≤ x)
            {
                5. i = i + 1;
                6. swap (A[i], A[ j]) ;
            }
7. swap (A[i + 1], A[r]) ;
8. return i + 1;
}
```

Mệnh đề :
Ở mỗi đầu vòng lặp 3–6, với chỉ số k,
1. If $p \le k \le i$, then $A[k] \le x$.
2. If $i + 1 \le k \le j - 1$, then $A[k] > x$.
3. If $k = r$, then $A[k] = x$.

7	6	8	2	4	9	10	5

```
QUICKSORT(A, p, r)
{
  1. if (p < r)
     {
        2. q ← PARTITION(A, p, r) ;
        3. QUICKSORT(A, p, q - 1) ;
        4. QUICKSORT(A, q + 1, r) ;
     }
}
```

Complex :

- Nếu ở **2** luôn tạo ra 2 dãy A[p...q-1] = ∅ và A[q+1...r] có (r-p) thì T(n) = $O(n^2)$ (VD : A đã được sắp xếp !)
- Nếu ở **2** luôn tạo ra 2 dãy A[p...q-1] và A[q+1...r] có số phần tử gần bằng nhau thì T(n) = O(n lgn)

9.6 Shell Sort :

- Cho mảng a[0, . . ., n-1], các phần tử a[i], a[j], . . . thuộc cùng *tập hợp khoảng-cách-h* nếu j = k*h + i , k = 0, 1, . . .

Ví dụ : a[0, . . ., 11],

- Các *tập hợp khoảng-cách-5 : {a[0], a[5], a[10]}, {a[1], a[6], a[11]} , {a[3], a[8]}, {a[4], a[9]}, {a[5], a[10]},*
- Các *tập hợp khoảng-cách-3 :{a[0], a[3], a[6], a[9]}, {a[1], a[4], a[7], a[10]}, {a[2], a[5], a[8], a[11]},*
- Các *tập hợp khoảng-cách-1 : {a[0], a[1], a[2], . . ., a[11]}.*

9.6 Shell Sort :

Thuật toán :

void SHELLSORT(int a[], int n) // n : the number of elements in A

{ **1.** Khởi tạo h[]={h_0 ,h_1 , . . ., h_m}; // $h_0 > h_1 > \ldots > h_m = 1$, {5, 3, 1}

2. for(k=0; k<=m; k++) // Lần lượt xét h_0 ,h_1 , . . ., h_m

3. for (i = h[k]; i < n; i += 1) // i = h_0 ,h_1 , . . ., h_m

// Sắp xếp các *tập hợp khoảng-cách-h_k tăng dần bằng Insertion Sort*

{

4. temp = a[i];

5. for (j = i; j >= h[k] && a[j - h[k]] > temp; j -= h[k])

6. a[j] = a[j - h[k]];

7. a[j] = temp;

}

}

- Mục đích của Shell sort là cải thiện Insertion sort.

1. Khởi tạo h[]={5, 3, 1};

2. for(k=0; k<=m; k++)

3. for (i = h[k]; i < n; i += 1)

{ **4.** temp = a[i];

5. for (j = i; j >= h[k] && a[j - h[k]] > temp; j -= h[k])

6. a[j] = a[j - h[k]];

7. a[j] = temp;

}

	0	1	2	3	4	5	6	7	8	9	10	11	12
a	81	94	11	96	12	35	17	95	28	58	41	75	15
h=5													
h=3													
h=1													

9.7 Shaker Sort :

Thuật toán :

```
void ShakerSort( int A[], int n  )
{ do{    // Phần 1
        1. swapped = 0;
        2. for (i=0 ;i<=n - 2;i++)
              3. if (A[ i ] > A[ i + 1 ]) {
                    4. swap( &A[ i ], &A[ i + 1 ] );
                    5. swapped = 1;  // Có hoán vị
                 }
        6. if (swapped==0) break;  // Nếu không có hoán vị thì kết thúc
                                                  sắp xếp. Mảng tăng dần.
           . . .
         } while(swapped==1);
```

9.7 Shaker Sort :

Thuật toán :

```
void ShakerSort( int A[], int n  )
{     do {
            . . .
          // Phần 2
          1. swapped = 0;
          2. for(i=n - 2 ;i >= 0; i--)
                3. if (A[ i ] > A[ i + 1 ]) {
                      4. swap(&A[ i ],&A[ i + 1 ] );
                      5. swapped = 1; // Có hoán vị
                }
          } while(swapped==1);
}
```

9.8 Exchange Sort :

Thuật toán :

```
void EXCHANGESORT(int a[], int n)
{
        for(i = 0; i < n -1; i++)
            for (j=i + 1; j < n; j++)
                if (a[i] > a[j]) swap(&a[i], &a[j]);
}
```