CHƯƠNG 9: Sắp xếp

9.1 Bubble Sort:

- **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 :** $O(n^2)$

CHƯƠNG 9: Sắp xếp

9.2 Insertion Sort :

- **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 :** $O(n^2)$

CHƯƠNG 9: Sắp xếp

9.3 Selection Sort :

- **Algorithm :**

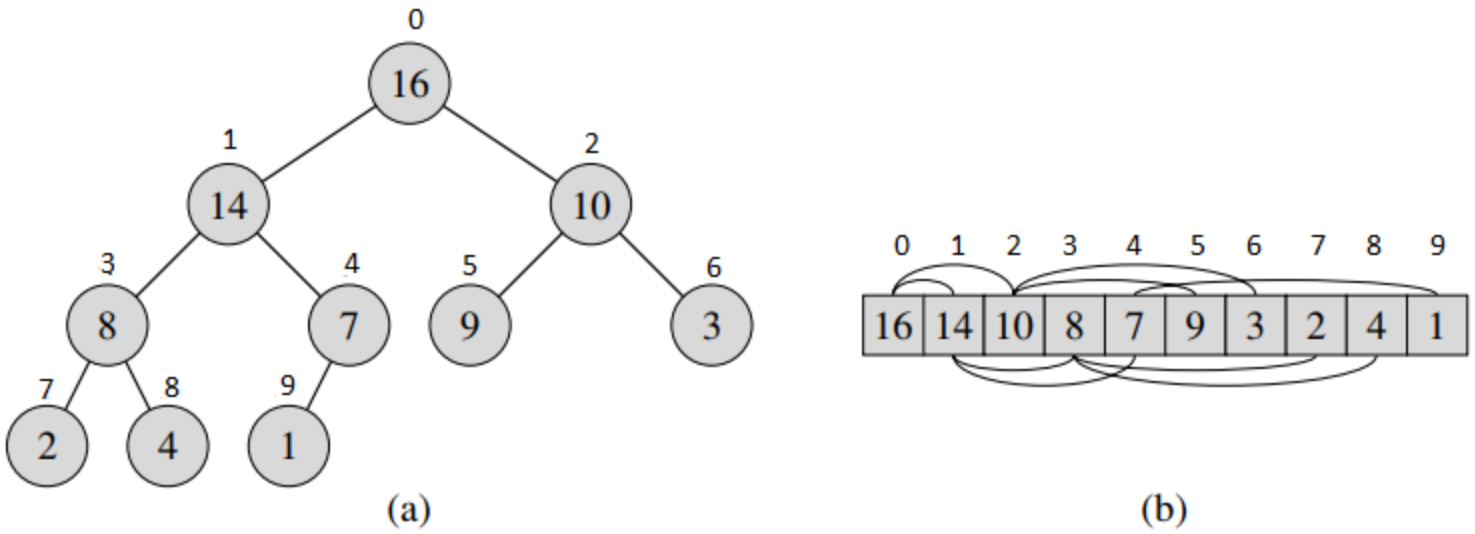
```
for ( i = 0 ; i < n-1 ; i++ ) {
        1. k = i ;
        2. for ( j = i+1 ; j < n ; j++ )
                3. if ( a[j] < a[k] ) k = j ;
        4. swap (a[i] ,a[k]);
}
```

- **Độ phức tạp :** $O(n^2)$

CHƯƠNG 9: Sắp xếp

9.3 Heap Sort :

- 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)**.

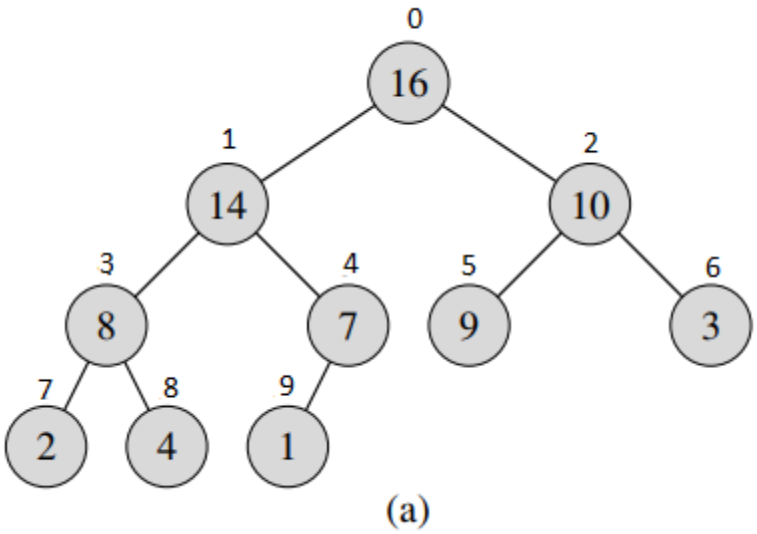
CHƯƠNG 9: Sắp xếp

9.3 Heap Sort :

- PARENT*(i)*

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

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



(a)

- LEFT*(i)*

  return $2i + 1$

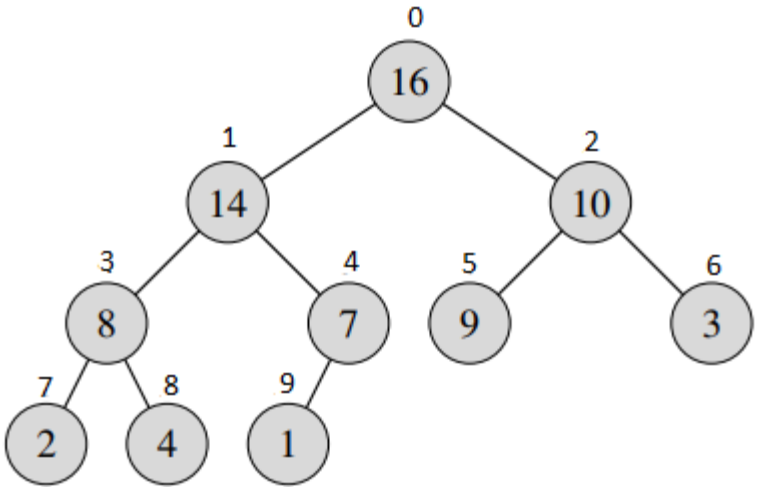
- RIGHT*(i)*

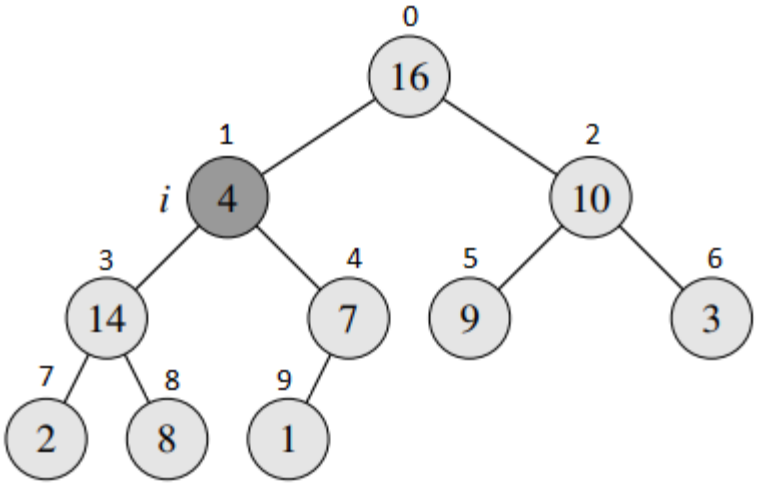
  return $2i + 2$

CHƯƠNG 9: Sắp xếp

9.3 Heap Sort :

- ***max-heap property*** : Với mọi nút i khác gốc, *A*[PARENT*(i)*] ≥ *A*[*i*].

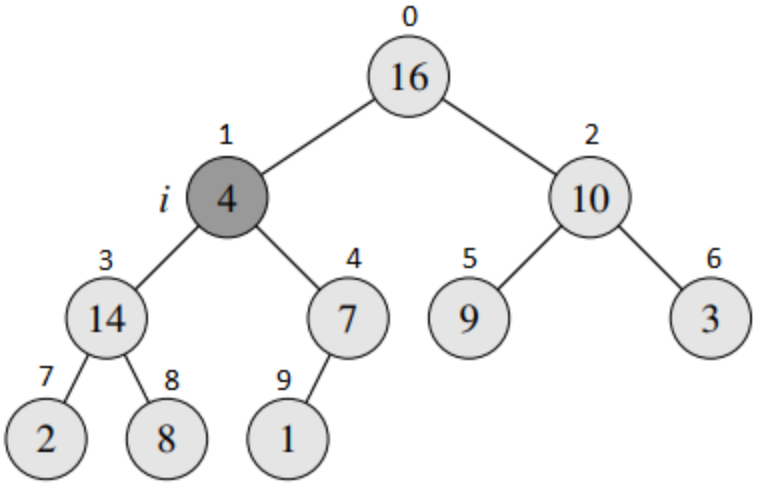




CHƯƠNG 9: Sắp xếp

9.3 Heap Sort :

- ***Thủ tục Max_Heap(A, i)*** :
  Giả sử : Cây con trái cây con phải của nút i thỏa *max-heap property* . Ví dụ i=4 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.*

CHƯƠNG 9: Sắp xếp

9.3 Heap Sort :

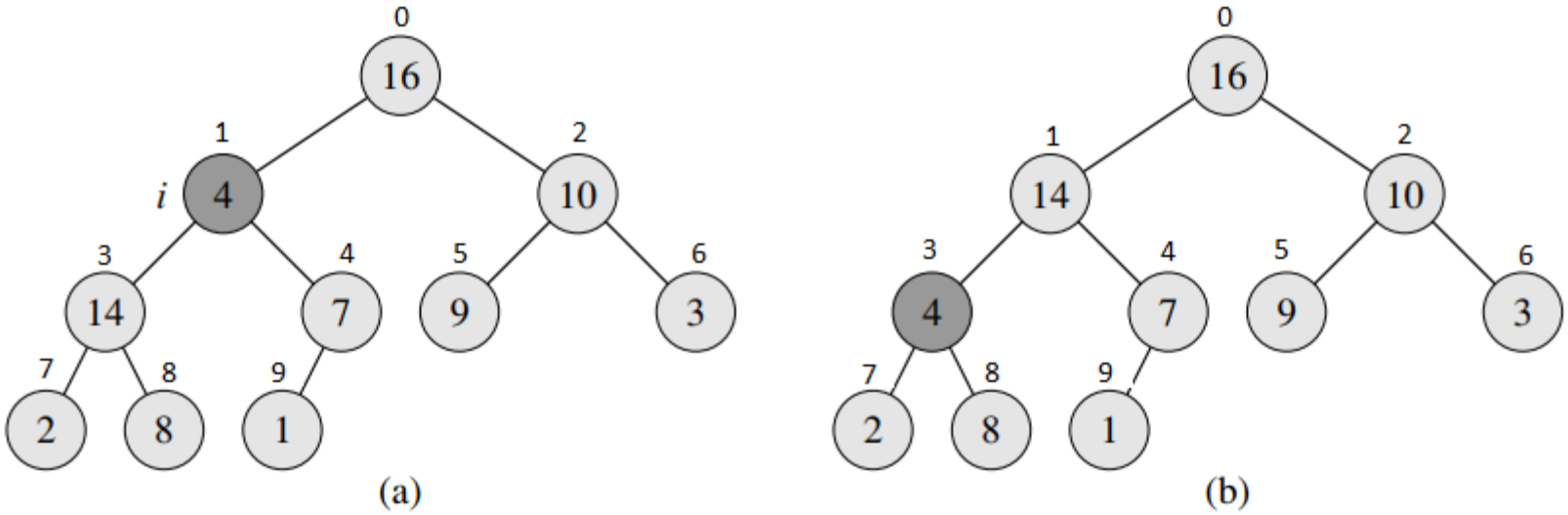
Thủ tục Max_Heap(A, i)

```
{
        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);
           }
}
```

CHƯƠNG 9: Sắp xếp

9.3 Heap Sort :

Ví dụ : ***Max_Heap(A, 1)***



CHƯƠNG 9: Sắp xếp

9.3 Heap Sort :

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

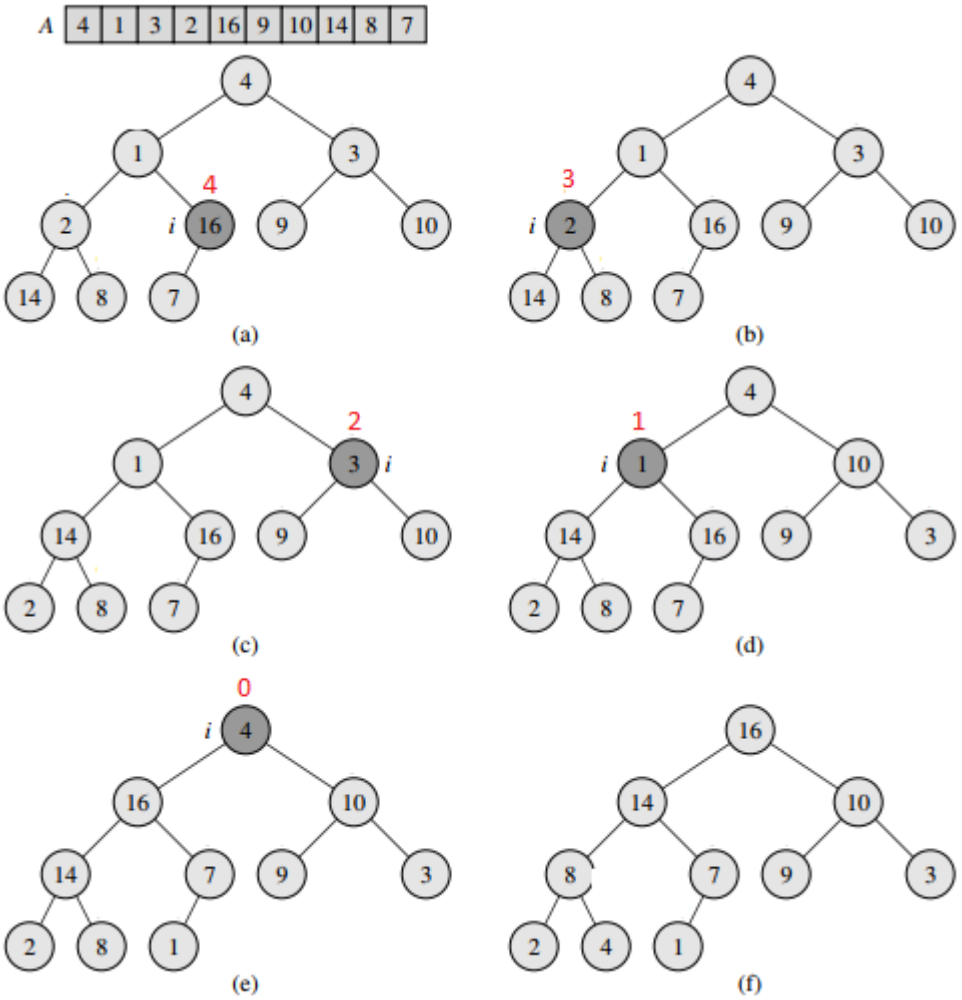
BUILD_MAX_HEAP(A, length)

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

CHƯƠNG 9: Sắp xếp

9.3 Heap Sort :

Ví dụ :



CHƯƠNG 9: Sắp xếp

9.3 Heap Sort :

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);
           }
} :
```

CHƯƠNG 9: Sắp xếp

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)
        }
}
```

CHƯƠNG 9: Sắp xếp

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
```

CHƯƠNG 9: Sắp xếp

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 gọ 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;
}
```

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 $A[k] = x$.

CHƯƠNG 9: Sắp xếp

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[]={h0 ,h1 , . . ., hm}; // h0 > h1 > . . . > hm = 1
  2. for(k=0; k<=m; k++)                  // Lần lượt xét h0 ,h1 , . . ., hm
       3. for (i = h[k]; i < n; i += 1)   // i = h0 ,h1 , . . ., hm
 // Sắp xếp các tập hợp khoảng-cách-hk 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.

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) 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]);
}
```