# Practica 2-Algoritmos de Ordenamiento. Parte 2.

**Objetivo:** El estudiante conocerá e identificará la estructura de los algoritmos de ordenamiento Quick Sort y Heap Sort.

**Actividades generales**

Implementar el algoritmo Quick Sort en algún lenguaje de programación para ordenar una secuencia de datos.

Implementar el algoritmo Heap Sort en algún lenguaje de programación para ordenar una secuencia de datos.

**Antecedentes**

- Análisis previo de los algoritmos en clase teórica.
- Manejo de arreglos o listas, estructuras de control y funciones en Python 3.

## Desarrollo:

**Actividad 1 (clase)** . Antes de implementar el algoritmo de Quick Sort, se trabajará con el algoritmo particionar que divide la secuencia en 2 de acuerdo a un elemento pivote seleccionado.

Utilizar el algoritmo particionar de la figura 1 y visto en clase, para realizar un programa en Python 3 que llame a una función que divida una lista en 2 considerando el elemento pivote como el último elemento. El programa debe mostrar como se lleva a cabo el algoritmo. Utilizar como ejemplo la figura 2 y ejemplo de salida.

```
Particionar(A,p,r)
Inicio
   x=A[r]
    i=p-1
    Para j=p hasta r-1
              Si A[j]<=x
                        i=i+1
                        intercambiar A[i] con A[j]
             Fin Si
    Fin para
  intercambiar A[i+1] con A[r]
retornar i+1
Fin
```

```python
def intercambia( A, x, y ):
    tmp = A[x]
    A[x] = A[y]
    A[y] = tmp

def Particionar(A,p,r):
    x=A[r]
    i=p-1
    for j in range(p,r):
        if (A[j]<=x):
            i=i+1
            intercambia(A,i,j)
    intercambia(A,i+1,r)
    return i+1
```

**Figura 1**

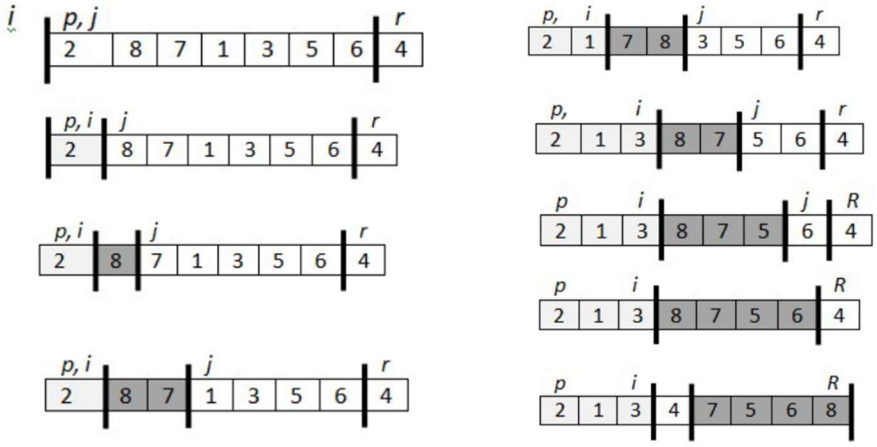


Figura 2.

Ejemplo de salida

```
[2, 8, 7, 1, 3, 5, 6, 4]
lista [] [] [2, 8, 7, 1, 3, 5, 6, 4]
lista [2] [] [8, 7, 1, 3, 5, 6, 4]
lista [2] [8] [7, 1, 3, 5, 6, 4]
lista [2] [8, 7] [1, 3, 5, 6, 4]
lista [2, 1] [7, 8] [3, 5, 6, 4]
lista [2, 1, 3] [8, 7] [5, 6, 4]
lista [2, 1, 3] [8, 7, 5] [6, 4]
lista [2, 1, 3] [8, 7, 5, 6] [4]
lista [2, 1, 3] 4 [7, 5, 6, 8]
[2, 1, 3, 4, 7, 5, 6, 8]
```

**Actividad 2 (clase)**

Elaborar un programa que ordene mediante el algoritmo Quick Sort de la figura 3, la lista de ejemplo de de la actividad 1. Además, se visualice en pantalla el cálculo del elemento pivote y de las secuencias izquierda y derecha. Un ejemplo de salida se muestra en la figura 4.

Probar que se observa cuando se utiliza un secuencia con los elementos todos iguales.

```
QuickSort(A,p,r)
Inicio
   Si p < r entonces
      q =Particionar(A,p,r)
       QuickSort(A,p,q-1)
      QuickSort(A,q+1,r)
    Fin Si
Fin
Particionar(A,p,r) Inicio
   x=A[r]
    i=p-1
    Para j=p hasta r-1
              Si A[j]<=x
                    i=i+1
                    intercambiar A[i] con A[j]
            Fin Si
    Fin para
   intercambiar A[i+1] con A[r]
retornar i+1
Fin
```

```python
def intercambia( A, x, y ):
    tmp = A[x]
    A[x] = A[y]
    A[y] = tmp

def Particionar(A,p,r):
    x=A[r]
    i=p-1
    for j in range(p,r):
        if (A[j]<=x):
            i=i+1
            intercambia(A,i,j)
    intercambia(A,i+1,r)
    return i+1


def QuickSort(A,p,r):
    if ( p < r ):

        q=Particionar(A,p,r)
        QuickSort(A,p,q-1)
        QuickSort(A,q+1,r)
```

Figura 3

```
Lista no ordenada [2, 8, 7, 1, 3, 5, 6, 4]
====Quick Sort====
pivote 4
[2, 1, 3] ,--4 -- [7, 5, 6, 8]
pivote 3
[2, 1] ,--3 -- []
pivote 1
[] ,--1 -- [2]
pivote 8
[7, 5, 6] ,--8 -- []
pivote 6
[5] ,--6 -- [7]
[1, 2, 3, 4, 5, 6, 7, 8]
```

Figura 4

**Actividad 3. Para trabajo en equipo** Otra variante de Quick Sort es el algoritmo de Hoare. Realizar un programa que implemente dicho algoritmo y se muestre el funcionamiento.

```
Quicksort(A,p,r){
        if (p<r) {
                q = Partition(A,p,r);
                Quicksort(A,p,q);
                Quicksort(A,q+1, r);
        }
}
```

```
Partition(A,p,r) {
    x=A[p];
    i=p-1;
    j=r+1;
    while(1) {
        do  j--;
        until (A[j] ≤ x)
        do  i++;
        until (A[i] ≥ x)
        if (i < j)
                exchange A[i] <-> A[j];
        else
                return j;
    }
}
```

Figura 5

**Actividad 4 Clase**

Se proporcionan algoritmos en pseudocódigo para el algoritmo Heap Sort así como su implementación en Python (Figura 6). Se requiere utilizarlas para elaborar un programa que ordene una lista de 20 elementos y se muestre en pantalla como se van acomodando los elementos al final de la lista. Ver figura 7 para ejemplo de salida.

```
MaxHeapify (A,i,TamañoHeap)
Inicio
    L= hIzq( i)
    R=hDer(i)
    Si L <=TamañoHeap-1 y A[L]>A[i]
        posMax=L
    En otro caso
        posMax = i
    Fin Si

    Si R<=TamañoHeapA-1 y A[R]> A[posMax]
        posMax =R
  Fin Si
  Si posMax ≠ i entonces
      Intercambia(A[i], A[posMax])
      MaxHeapify(A,posMax,tamañoHeap)
   Fin Si
Fin
```

```
import math

def hIzq(i):
    return 2*i+1
def hDer(i):
    return 2*i+2
def intercambia( A, x, y ):
    tmp = A[x]
    A[x] = A[y]
    A[y] = tmp
def MaxHeapify(A,i,tamanoHeap):
    L=hIzq(i)
    R=hDer(i)
    if ( L <= (tamanoHeap-1)  and A[L] > A[i] ):
        posMax=L
    else:
        posMax=i
    if (R <= (tamanoHeap-1) and A[R]>A[posMax]):
        posMax=R
    if (posMax != i):
        intercambia(A,i,posMax)
        MaxHeapify(A,posMax,tamanoHeap)
```

```
HijoIzquierdo(i)
Inicio
    retorna 2*i +1
Fin
HijoDerecho
Inicio
    retorna 2*i +2
Fin

construirHeapMaxIni( A, tamañoHeapA )
Inicio
   Para i=Piso((longitusDeA-1)/2) hasta 0
      MaxHeapify(A,i, tamañoHeapA)
   Fin Para
Fin

OrdenacionHeapSort( A, tamañoHeapA)
Inicio
        construirHeapMaxIni( A,tamañoHeapA)
        Para i=longiudDeA-1 hasta 1 hacer
              Intercambia(A[0], A[i])
            TamañoHeapA= TamañoHeapA-1;
            MaxHeapify (A,0,TamañoHeapA)
Fin
```

```
def construirHeapMaxIni(A,tamanoHeap):
    for i in range(math.ceil((tamanoHeap-1)/2),-1,-1):
        MaxHeapify(A,i,tamanoHeap)

def OrdenacioHeapSort(A,tamanoHeap):
    construirHeapMaxIni(A,tamanoHeap)
    for i in range (len(A)-1,0,-1):
        intercambia(A,0,i)
        tamanoHeap=tamanoHeap-1
        MaxHeapify(A,0,tamanoHeap)
```

**Figura 6**

```
Heap maximo inicial
 [16, 14, 10, 8, 7, 9, 3, 2, 4, 1]
Empieza ordenamiento

intercambio de A[0]=, 16 con  A[9]=", 1
 Después de intercambio no es heap max
[1, 14, 10, 8, 7, 9, 3, 2, 4] [16]

Reconstruccion del heap
[14, 8, 10, 4, 7, 9, 3, 2, 1] [16]

intercambio de A[0]=, 14 con  A[8]=", 1
 Después de intercambio no es heap max
[1, 8, 10, 4, 7, 9, 3, 2] [14, 16]

Reconstruccion del heap
[10, 8, 9, 4, 7, 1, 3, 2] [14, 16]

intercambio de A[0]=, 10 con  A[7]=", 2
 Después de intercambio no es heap max
[2, 8, 9, 4, 7, 1, 3] [10, 14, 16]

Reconstruccion del heap
[9, 8, 3, 4, 7, 1, 2] [10, 14, 16]

intercambio de A[0]=, 9 con  A[6]=", 2
 Después de intercambio no es heap max
[2, 8, 3, 4, 7, 1] [9, 10, 14, 16]

Reconstruccion del heap
[8, 7, 3, 4, 2, 1] [9, 10, 14, 16]

intercambio de A[0]=, 8 con  A[5]=", 1
 Después de intercambio no es heap max
[1, 7, 3, 4, 2] [8, 9, 10, 14, 16]

Reconstruccion del heap
[7, 4, 3, 1, 2] [8, 9, 10, 14, 16]

intercambio de A[0]=, 7 con  A[4]=", 2
 Después de intercambio no es heap max
[2, 4, 3, 1] [7, 8, 9, 10, 14, 16]

Reconstruccion del heap
[4, 2, 3, 1] [7, 8, 9, 10, 14, 16]

intercambio de A[0]=, 4 con  A[3]=", 1
 Después de intercambio no es heap max
[1, 2, 3] [4, 7, 8, 9, 10, 14, 16]

Reconstruccion del heap
[3, 2, 1] [4, 7, 8, 9, 10, 14, 16]

intercambio de A[0]=, 3 con  A[2]=", 1
 Después de intercambio no es heap max
[1, 2] [3, 4, 7, 8, 9, 10, 14, 16]

Reconstruccion del heap
[2, 1] [3, 4, 7, 8, 9, 10, 14, 16]

intercambio de A[0]=, 2 con  A[1]=", 1
 Después de intercambio no es heap max
[1] [2, 3, 4, 7, 8, 9, 10, 14, 16]

Reconstruccion del heap
[1] [2, 3, 4, 7, 8, 9, 10, 14, 16]
[1, 2, 3, 4, 7, 8, 9, 10, 14, 16]
```

**Actividad 5 Para trabajo en equipo**

Modificar el algoritmo para que se utilice un heap mínimo y realizar un programa que ordene utilizando el uso de este. Al ejecutar su programa hay diferencia con respecto al uso del heap máximo. Explicar

**Actividad 6 Para trabajo en equipo**

Realizar las gráficas de comparación entre QuickSort y HeapSort y MergeSort como las realizadas en práctica 1. Describir el resultado del comportamiento de las gráficas.