Instituto Politécnico Nacional

Escuela Superior de Computo

(ESCOM)

Materia:

Análisis y Diseño de Algoritmos.

Tema:

Reporte de algoritmo QUICK SORT.

Alumno:

Casiano Granados Brandon Antonio

Carrera:

Ingeniería en Sistemas Computacionales.

Grupo:

3CV14

Profesora:

Moreno Galván Elizabeth

Nuestro código está constituido por las funciones:

- main
- imprime
- qs

La función main solo se encarga de la secuencia de invocación de qs e imprime.

Imprime solo se encarga de mostrar lo que contiene el arreglo que se le envía como parámetro.

Lo interesante viene en la función qs.

```
private static void qs (int vector [], int inf, int sup)
 {
    int izq, der, pivote, aux;
    izq = inf;
    der = sup;
    pivote = vector [(izq + der) /2];
    do
    {
       while((vector[izq] < pivote) && (izq < sup))
          izq++;
       while((vector[der] > pivote) && (der > inf))
          der--;
       if (izq <= der)
       {
          aux = vector[izq];
          vector[izq] = vector[der];
          vector[der] = aux;
          izq++;
          der++;
       }
    } while (izq <= der);
```

```
    if (inf < der)
        qs (vector, inf, der);
    if (izq < sup)
        qs (vector, izq, sup);
    return;
}
```

Podemos observar que la función “qs” recibe como parámetro un “vector []” que es nuestro conjunto de números a ordenar, en valor “inf” que nos indicara el principio de nuestro vector y “sup” que es el final de nuestro vector.

Declaramos una variable “pivote” la cual nos ayudara a dividir el vector en subvectores, subvector de números menores (lado izquierdo del vector) y subvector de números mayores (lado derecho del vector).

La variable “izq” nos ayudara a movernos en los nodos del vector que sean menores al “pivote”.

La variable “der” nos ayudara a movernos en los nodos del vector que sean mayores al “pivote”.

Para los siguientes análisis nos basaremos en un vector que ya esta ordenado solo para tener una mejor abstracción de lo que se desea.

Vector =

1	2	3	6	5	4	7	8	9	10

Pivote = “6”.

1. **<u>Analizaremos el bucle While “while((vector[izq] < pivote) && (izq < sup)) izq++;”.</u>**

En esta línea recorremos al vector por la izquierda desde la posición marcada como izquierda que inicia en el principio de nuestro vector, hasta la posición menor al pivote. La condición es: “vector[izq] < pivote”.

La condición “izq < sup”, es para evitar el desborde del arreglo por la derecha, esta condición entra con mayor robustez cuando se trabaja con los subvectores, ya que las variables “inf” y “sup” marcan el inicio y fin del vector padre y todos los subvectores que se crean, evita que evaluemos posiciones de vector que no corresponden.

Cada que encuentre un numero que sea mayor al pivote el bucle, se dejara de ejecutar, pero se quedara almacenado la posición en la variable “izq”, esto indica que por cada numero que sea mayor al pivote en el lado izquierdo del vector, existe un numero menor al pivote que se encuentra en el lado derecho del vector.

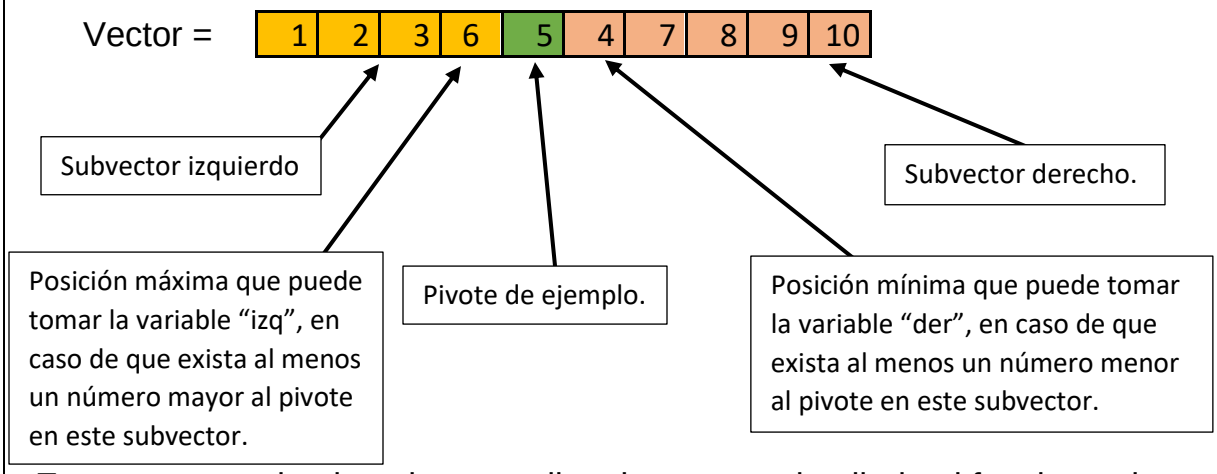
2. **Analizaremos el bucle while “while((vector[der] > pivote) && (der > inf)) der--;”**

En esta línea recorremos al vector por la derecha desde la posición marcada como derecha que inicia en el final de nuestro vector, hasta la posición mayor al pivote. La condición es: “**vector[der] > pivote**”.

La condición “**der > inf**”, es para evitar el desborde del arreglo por la izquierda, esta condición entra con mayor robustez cuando se trabaja con los subvectores, ya que las variables “inf” y “sup” marcan el inicio y fin del vector padre y todos los subvectores que se crean, evita que evaluemos posiciones de vector que no corresponden.

Cada que encuentre un número que sea menor al pivote el bucle, se dejara de ejecutar, pero se quedara almacenado la posición en la variable “der”, esto indica que por cada número que sea menor al pivote en el lado derecho del vector, existe un número mayor al pivote que se encuentra en el lado izquierdo del vector.

3. **Con lo mencionado en los puntos 1 y 2, se pretende que el vector tenga un comportamiento al siguiente:**



En este vector de ejemplo se explica de manera detallada el funcionamiento de los puntos 1 y 2.

4. **Analizaremos el código siguiente:**

```
“if (izq <= der)
{
aux = vector[izq];
vector[izq] = vector[der];
vector[der] = aux;
izq++;
der++;
```

}”

La condición del if “**izq <= der**”, se coloca por que si esto no se cumple significa que ya no existen números mayores en el subvector izquierdo y números menores en el subvector derecho. Como se mostro en la imagen la variable “izq” puede llegar como máximo un lugar antes del pivote siempre y cuando exista al menos un numero mayor al pivote en el subvector izquierdo.

Las líneas de código que conforman este if son para intercambiar los números que se encuentran en los subvectores que no le corresponden, e incrementa las variables “izq” y “der” en una posición para continuar con las demás evaluaciones.

5. **Analizaremos el do while “while (izq <= der);”.**

Este bucle pretende que cuando el vector ya se dividió en subvectores, estos tengan por un lado los números menores y por el otro los mayores.

6. **Analizaremos el código**

“if (inf < der)

qs (vector, inf, der);

if (izq < sup)

qs (vector, izq, sup);

return;”

Cada uno de los if evalúa cada subvector, mientras tengan más de un elemento contenido se llamará a la misma función, donde los subvectores se convertirán en vectores y se repetirá el código anterior.

Cuando los vectores ya solo tengan un elemento, se romperá la recursividad y el vector principal estará totalmente ordenado.