Ordenamiento Quicksort en C++

Quicksort es un algoritmo de ordenación considerado entre los más rápidos y eficientes. Fue diseñado en los años 60s por Hoare un científico en computación.

El algoritmo usa la técnica divide y vencerás que básicamente se basa en dividir un problema en subproblemas y luego juntar las respuestas de estos subproblemas para obtener la solución al problema central.

Se tiene un vector de n elementos, se toma un valor del vector como pivote (usualmente el primero), se separa los elementos menores a este pivote a la izquierda y los mayores a la derecha, es decir, se divide el vector en 2 subvectores.

Con estos subvectores se repite el mismo proceso de forma recursiva[1] hasta que estos tengan más de 1 elemento. Por lo tanto, la función ordenarQuicksort quedaría de la siguiente manera:

```
// Función recursiva para hacer el Ordenamiento Quicksort
void ordenarQuicksort(int v[MAX], int inicio, int fin)
{
    int pivote;
    if (inicio < fin) {
        pivote = dividir(v, inicio, fin);
        // Ordeno la lista de los menores
        ordenarQuicksort(v, inicio, pivote - 1);
        // Ordeno la lista de los mayores
        ordenarQuicksort(v, pivote + 1, fin);
    }
}
```

La magia está en la función dividir que es la que se va explicar a continuación:

Se empieza creando un vector de n elementos, por ejemplo,se utilizó la función rand() de C++ para generar los números aleatorios del 1 al 15, quedando el siguiente arreglo:

v [] = {8, 1, 5, 14, 4, 15, 12, 6, 2, 11, 10, 7, 9};

izquierda derecha

8	1	5	14	4	15	12	6	2	11	10	7	9
0	1	2	3	4	5	6	7	8	9	10	11	12

[1] **Recursividad:** Un algoritmo recursivo es un algoritmo que expresa la solución de un problema en términos de una llamada a sí mismo. La llamada a sí mismo se conoce como llamada recursiva o recurrente.

Se toma como pivote el 8 y se utilizan 2 índices que indican la posición del vector:

- Uno que vaya de izquierda a derecha buscando los elementos mayores al pivote. **v[izq]**
- Y un índice que busque de derecha a izquierda los elementos menores al pivote. **v[der]**

El índice izquierdo irá aumentando en 1 mientras el vector en la posición izquierda sea menor o igual al pivote:

```
while ((izq < der) && (v[izq] <= pivote)) {
    izq++;
}
```

El índice derecho irá reduciéndose en 1 mientras el vector en la posición derecha sea mayor al pivote.

```
while (v[der] > pivote) {
    der--;
}
```

Si al final de estas 2 operaciones, el índice izquierdo es menor al derecho se intercambian las posiciones **v[izq]** con **v[der]** usando una variable auxiliar:

Primer recorrido: El índice izquierdo encuentra al 14 (mayor al pivote) y el índice derecho al 7 (menor al pivote), y se intercambian los índices:

8	1	5	**14**	4	15	12	6	2	11	10	**7**	9
8	1	5	**7**	4	15	12	6	2	11	10	**14**	9
0	1	2	3	4	5	6	7	8	9	10	11	12

Segundo recorrido: El índice izquierdo encuentra al 15 (mayor al pivote) y el índice derecho al 2 (menor al pivote), se intercambian:

8	1	5	7	4	**15**	12	6	**2**	11	10	14	9
8	1	5	7	4	**2**	12	6	**15**	11	10	14	9
0	1	2	3	4	5	6	7	8	9	10	11	12

Tercer recorrido: El índice izquierdo encuentra al 12 (mayor al pivote) y el índice derecho al 6 (menor al pivote), se intercambian:

8	1	5	7	4	2	**12**	**6**	15	11	10	14	9
8	1	5	7	4	2	**6**	**12**	15	11	10	14	9
0	1	2	3	4	5	6	7	8	9	10	11	12

Cuando los índices se juntan o se cruzan ponemos el pivote en el lugar que le corresponde en el vector:

```
// Los índices ya se han cruzado, se pone el pivote en el lugar que le corresponde
aux = v[der];
v[der] = v[inicio];
v[inicio] = aux;
```

Se intercambian el 8 con el 6 y el vector quedaría así:

8	1	5	7	4	2	**6**	12	15	11	10	14	9
6	1	5	7	4	2	**8**	12	15	11	10	14	9
0	1	2	3	4	5	6	7	8	9	10	11	12

Ahora la función ordenarQuicksort se llamaría 2 veces recursivamente para los 2 subvectoresque se tienen:

```
ordenarQuicksort(v, 0, 5) // el pivote está en la posición 6
ordenarQuicksort(v, 7, 12) // el pivote está en la posición 6
```

Se repite el mismo proceso con este primer subvector ordenarQuicksort (v, 0, 5)

El pivote es 6, se encuentra por la izquierda al 7 y por la derecha al 2. Se intercambian y como se cruzaron los índices se mueve el pivote a su lugar:

6	1	5	**7**	4	**2**	8	12	15	11	10	14	9
6	1	5	**2**	4	**7**	8	12	15	11	10	14	9
4	1	5	**2**	**6**	**7**	8	12	15	11	10	14	9
0	1	2	3	4	5	6	7	8	9	10	11	12

Otra vez tenemos 2 subvectores. Entonces se vuelve a llamar la función.

```
ordenarQuicksort(v, 0, 3) // la posición del pivote es 4
ordenarQuicksort(v, 5, 5) // no se ejecuta nada, el inicio no es menor al final
```

El mismo proceso. Se ejecutar ordenarQuicksort (v, 0, 3). El pivote ahora es 4, se encuentra por el índice izquierdo al 5 y por el derecho a 2. Se intercambian y como se juntaron los índices se mueve el pivote a su lugar.

4	1	**5**	**2**	6	7	8	12	15	11	10	14	9
4	1	**2**	**5**	6	7	8	12	15	11	10	14	9
2	1	**4**	**5**	6	7	8	12	15	11	10	14	9
0	1	2	3	4	5	6	7	8	9	10	11	12

```
ordenarQuicksort(v, 0, 1) // la posición del pivote es 2
ordenarQuicksort(v, 3, 3) // no se ejecuta nada, el inicio no es menor al final
```

Cuando se ejecuta Quicksort (v, 0, 1) se intercambia los índices otra vez.

2	**1**	4	5	6	7	8	12	15	11	10	14	9
1	**2**	4	5	6	7	8	12	15	11	10	14	9
0	1	2	3	4	5	6	7	8	9	10	11	12

Ahora se llamaría a ordenarQuicksort (v, 0, 0), se divide en 2 elementos el subvector y no hay nada más que hacer porque solo tienen 1 elemento.

Ahora se ejecuta el mismo proceso con el segundo subvector delvector original. ordenarQuicksort (v, 7, 12)

Se encuentra el 15 y el 9, se intercambian y como luego se juntan los índices, se coloca el pivote a su lugar:

2	1	4	5	6	7	8	**12**	**15**	11	10	14	**9**
1	2	4	5	6	7	8	**12**	**9**	11	10	14	**15**
1	2	4	5	6	7	8	10	**9**	11	**12**	14	**15**
0	1	2	3	4	5	6	7	8	9	10	11	12

Se tienen otros 2 subvectores y se vuelve a llamar la función ordenarQuicksort.

```
ordenarQuicksort(v, 7, 10) // posición del pivote es 10 y la primera 7
ordenarQuicksort(v, 11, 12) // posición del pivote es 10 y la última 12
```

Se intercambian los índices y nos quedan otros 2 subvectores de 1 solo elemento entonces no se ejecuta nada.

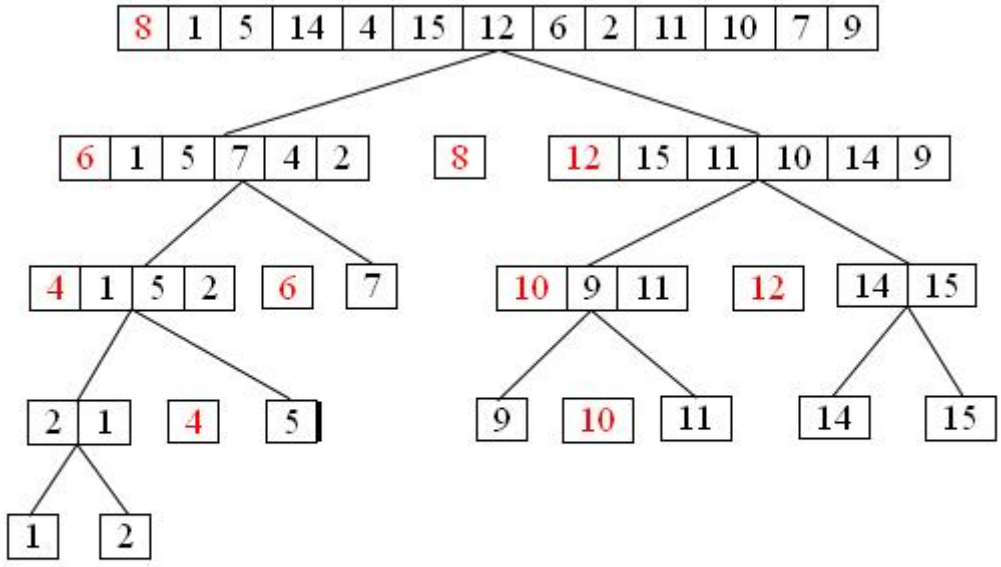
1	2	4	5	6	7	8	**10**	9	11	12	14	15
1	2	4	5	6	7	8	9	**10**	11	12	14	15
0	1	2	3	4	5	6	7	8	9	10	11	12

Con el anterior subvector (14, 15) se llama a ordenarQuicksort(v, 11, 12).

1	2	4	5	6	7	8	9	10	11	12	**14**	15
0	1	2	3	4	5	6	7	8	9	10	11	12

Se divide en 2 elementos este subvector y no hay nada más que hacer porque los vectores que contienen al 14 y al 15 solo tienen 1 elemento.

De manera que el árbol recursivo de ordenación queda así:



Juntando los elementos, el arreglo quedaría ordenado

1	2	4	5	6	7	8	9	10	11	12	14	15

Código completo de las funciones en C++.

```c
// Función recursiva para hacer el Ordenamiento Quicksort
void ordenarQuicksort(int v[MAX], int inicio, int fin)
{
    int pivote;
    if (inicio < fin) {
        pivote = dividir(v, inicio, fin);
        // Ordeno la lista de los menores
        ordenarQuicksort(v, inicio, pivote - 1);
        // Ordeno la lista de los mayores
        ordenarQuicksort(v, pivote + 1, fin);
    }
}
// Función para dividir el array y hacer los intercambios
int dividir(int v[MAX], int inicio, int fin) {
    int izq, der, pivote, aux;
    pivote = v[inicio];
    izq = inicio;
    der = fin;
    // Mientras no se cruzen los índices
    while (izq < der) {
        while (v[der] > pivote) {
            der--;
        }
        while ((izq < der) && (v[izq] <= pivote)) {
            izq++;
        }
        // Si todavía no se cruzan los índices segue intercambiando
        if (izq < der) {
            aux = v[izq];
            v[izq] = v[der];
            v[der] = aux;
        }
    }
    // Los índices ya se han cruzado, se pone el pivote en el lugar que le corresponde
    aux = v[der];
    v[der] = v[inicio];
    v[inicio] = aux;
    // La nueva posición del pivote
    return der;
}
```