



INSTITUTO POLITÉCNICO NACIONAL



"ESCOM" (ESCUELA SUPERIOR DE CÓMPUTO)

DOCENTE: CECILIA ALBORTANTE MORATO

INTEGRANTES:

MONTEALEGRE ROSALES DAVID URIEL

VÁZQUEZ BLANCAS CÉSAR SAID

U.A.: ANÁLISIS Y DISEÑO DE ALGORITMOS

GRUPO: 3CM6

PRÁCTICA 3: QUICKSORT

Introducción.

El problema de ordenamiento es uno de los problemas fundamentales en la informática, y consiste en organizar un conjunto de elementos en un orden específico. Aunque puede parecer trivial, este problema es esencial para una amplia variedad de aplicaciones, desde la clasificación de datos hasta la generación de informes y la búsqueda eficiente de información.

Existen numerosos algoritmos de ordenamiento que se utilizan en la práctica, cada uno con sus ventajas y desventajas. Algunos de los algoritmos más comunes son el ordenamiento burbuja, el ordenamiento por selección, el ordenamiento por inserción y el ordenamiento por mezcla.

La elección del algoritmo de ordenamiento adecuado depende de la aplicación específica y del tamaño del conjunto de datos que se está ordenando. Para conjuntos pequeños, los algoritmos simples como el ordenamiento por inserción pueden ser suficientes, pero para conjuntos de datos más grandes se requieren algoritmos más eficientes, como el ordenamiento por mezcla.

En esta práctica se explorará el algoritmo quicksort, un algoritmo eficiente pero que en su peculiar forma de dividir el problema llega a ser bastante lento a comparación del merge sort por ejemplo, sin embargo tiene un grado de optimización y en esta práctica veremos eso, una forma común y una optimizada de tal algoritmo que será el buscar un numero random y tomar ese como pivote, el algoritmo quick sort funciona con recursividad y su función llamada partición, esta toma el pivote y pone de un lado a los mayores y del otro a los menores, esto hace que llegue a ser un algoritmo eficiente.

Desarrollo de la práctica

1.- Ordenamiento Quicksort.

La complejidad del algoritmo quicksort es:

El mejor de los casos: $O(n \log n)$

El peor de los casos: $O(n^2)$

Caso promedio: $O(n \log n)$

Quicksort es un algoritmo basado en el principio de "divide y vencerás". Al igual que todos los algoritmos que utilizan este principio, Quicksort comienza dividiendo una gran matriz en dos subarray más pequeños y luego aplica recursivamente la ordenación a los subarrays. El proceso consta fundamentalmente de tres pasos.

1. Selección de pivote: se elige un elemento, llamado pivote, de la matriz (generalmente el elemento más a la izquierda o más a la derecha de la partición).
2. Fraccionamiento: se reordena la matriz de manera que todos los elementos con valores menores que el pivote estén antes del pivote. Por el contrario, todos los elementos con valores mayores que el pivote vienen después de este. Los valores iguales pueden ir en cualquier dirección. Después de esta partición, el pivote está en su posición final.
3. Repetirse: se aplica recursivamente los pasos anteriores al sub-array de elementos con valores más pequeños que el pivote y por separado al subarray de elementos con valores mayores que el pivote.

```
void quicksort(int *a, int p, int r) {
    if (p < r)
    {
        int q = particion(a, p, r);
        quicksort(a, p, q-1);
        quicksort(a, q+1, r);
    }
}

void intercambiar(int *a, int i, int j) {
    int temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}
```

```
int particion(int *a, int p, int r) {
    int x = a[r];
    int i = p - 1;
    int j;
    for (j = p; j < r; j++) {
        if (a[j] <= x) {
            i++;
            intercambiar(a,i,j);
        }
    }
    intercambiar(a,i+1,r);
    return i + 1;
}
```

En esta práctica se explora el caso del pivote, los algoritmos implementados son 2, uno que es el quicksort normal, que toma como pivote al último elemento del array, esto hace que vaya dividiendo el array en 2, una parte de los mayores y la otra de los menores, y el pivote al final queda en posición, sin embargo este algoritmo hace que pensemos en una cosa, que pasa si un lado queda más desbalanceado que otro, pues si eso sucede hace que el tiempo de ejecución no se divida a la mitad, si no un poco más o menos, esto hace que el algoritmo tarde más en dividir y resolver los problemas, y más si depende del último, pues si queda desbalanceado, hará más llamadas de lo habitual que sería si dividiera en 2, y esto hace que el peor de los casos sea n^2 .

```
void random_quicksort(int *a, int p, int r) {
    if (p < r) {
        int q = random_particion(a, p, r);
        random_quicksort(a, p, q - 1);
        random_quicksort(a, q + 1, r);
    }
}
```

```
int particion(int *a, int p, int r) {
    int x = a[r];
    int i = p - 1;
    int j;
    for (j = p; j < r; j++) {
        if (a[j] <= x) {
            i++;
            int temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }
    int temp = a[i + 1];
    a[i + 1] = a[r];
    a[r] = temp;
    return i + 1;
}
```

```
int random_particion(int *a, int p, int r) {
    int i = rand() % (r - p + 1) + p;
    intercambiar(a, i, r); // Intercambia el pivote de forma segura
    return particion(a, p, r);
}

void intercambiar(int *a, int i, int j) {
    int temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}
```

En el otro algoritmo es básicamente lo mismo, solo que ahora se elegir un pivote random para garantizar o ganar más posibilidades de tomar como pivote a un número que balance el arreglo, con este random ganamos posibilidades de que el array pueda quedar lo más balanceado posible, en todas o la mayoría de sus llamadas a la función recursiva

Resultados:

Quicksort con pivote al final:

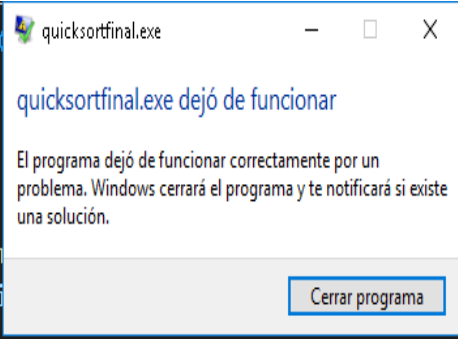
```
Escribe los numeros con los que quieres trabajar: 5000
Tiempo de ejecucion de quicksort: 0.003000 segundos
Tiempo de ejecucion de quicksort en el peor de los casos es: 0.195000 segundos

Escribe los numeros con los que quieres trabajar: 10000
Tiempo de ejecucion de quicksort: 0.009000 segundos
Tiempo de ejecucion de quicksort en el peor de los casos es: 0.733000 segundos

Escribe los numeros con los que quieres trabajar: 20000
Tiempo de ejecucion de quicksort: 0.032000 segundos
Tiempo de ejecucion de quicksort en el peor de los casos es: 2.896000 segundos

Escribe los numeros con los que quieres trabajar: 30000
Tiempo de ejecucion de quicksort: 0.082000 segundos
Tiempo de ejecucion de quicksort en el peor de los casos es: 6.508000 segundos
```

```
74         if (
75             }
76     }
77     }
78 }
79 intercambiar
80 return i
81 }
```



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
> C/C++ Compile Run + ▾ □ □
```

```
Tiempo de ejecucion de quicksort: 0.032000 segundos
Tiempo de ejecucion de quicksort en el peor de los casos es: 2.896000 segundos
PS C:\Users\pvl_2\Documents\package\C\C\c\output> cd 'c:\Users\pvl_2\Documents\package\C\C\c\output'
PS C:\Users\pvl_2\Documents\package\C\C\c\output> & .\'quicksortfinal.exe'
Escribe los numeros con los que quieres trabajar: 30000
Tiempo de ejecucion de quicksort: 0.082000 segundos
Tiempo de ejecucion de quicksort en el peor de los casos es: 6.508000 segundos
PS C:\Users\pvl_2\Documents\package\C\C\c\output> cd 'c:\Users\pvl_2\Documents\package\C\C\c\output'
PS C:\Users\pvl_2\Documents\package\C\C\c\output> & .\'quicksortfinal.exe'
Escribe los numeros con los que quieres trabajar: 40000
Tiempo de ejecucion de quicksort: 0.138000 segundos
```

```
Escribe los numeros con los que quieres trabajar: 400000
Tiempo de ejecucion de quicksort: 11.631000 segundos
```

```
Escribe los numeros con los que quieres trabajar: 500000
Tiempo de ejecucion de quicksort: 18.177000 segundos
```

```
Escribe los numeros con los que quieres trabajar: 600000
Tiempo de ejecucion de quicksort: 29.217000 segundos
```

```
Escribe los numeros con los que quieres trabajar: 700000
Tiempo de ejecucion de quicksort: 39.370000 segundos
```

```
Escribe los numeros con los que quieres trabajar: 800000
Tiempo de ejecucion de quicksort: 51.518000 segundos
```

Quick Sort con pivote random :

```
Escribe los numeros con los que quieres trabajar: 5000
Tiempo de ejecucion de quicksort: 0.002000 segundos
Tiempo de ejecucion de quicksort random en el peor de los casos: 0.002000 segundos
```

```
Escribe los numeros con los que quieres trabajar: 10000
Tiempo de ejecucion de quicksort: 0.007000 segundos
Tiempo de ejecucion de quicksort random en el peor de los casos: 0.007000 segundos
```

```
Escribe los numeros con los que quieres trabajar: 20000
Tiempo de ejecucion de quicksort: 0.025000 segundos
Tiempo de ejecucion de quicksort random en el peor de los casos: 0.033000 segundos
```

```
Escribe los numeros con los que quieres trabajar: 30000
Tiempo de ejecucion de quicksort: 0.067000 segundos
Tiempo de ejecucion de quicksort random en el peor de los casos: 0.064000 segundos
```

```
Escribe los numeros con los que quieres trabajar: 400000
Tiempo de ejecucion de quicksort: 8.621000 segundos
Tiempo de ejecucion de quicksort random en el peor de los casos: 8.718000 segundos
```

```
Escribe los numeros con los que quieres trabajar: 500000
Tiempo de ejecucion de quicksort: 13.612000 segundos
Tiempo de ejecucion de quicksort random en el peor de los casos: 13.304000 segundos
```

```
Escribe los numeros con los que quieres trabajar: 600000
Tiempo de ejecucion de quicksort: 20.999000 segundos
Tiempo de ejecucion de quicksort random en el peor de los casos: 20.619000 segundos
```

```
Escribe los numeros con los que quieres trabajar: 700000
Tiempo de ejecucion de quicksort: 27.960000 segundos
Tiempo de ejecucion de quicksort random en el peor de los casos: 27.974000 segundos
```

```
Escribe los numeros con los que quieres trabajar: 800000
Tiempo de ejecucion de quicksort: 37.441000 segundos
Tiempo de ejecucion de quicksort random en el peor de los casos: 37.301000 segundos
```

PRÁCTICA 1: RESOLUCIÓN DE PROBLEMAS

El código implementado con el pivote random es en cierto aspecto más eficiente , no por mucho pero si llega a hacer una diferencia significativa que va a ir creciendo conforme crece la entrada lo que lo hace más eficiente a grandes rasgos , eso en el caso promedio, en el peor de los casos es donde se hace la diferencia pues en el pivote final el tiempo de ejecución va creciendo mucho , pues cuando el caso promedio está en milisegundos o menos, el peor de los casos ya está en los segundos sin embargo cuando llega a más de 30 k el algoritmo se rompe, dejando solo el caso promedio, y ya no dando el resultado en el peor de los casos, en cambio con el pivote random , el tiempo de ejecución es menor o casi igual al caso promedio , siendo menor a veces por muy poco, pero a diferencia del pivote al final, este si compila y retorna un resultado y resultado que es muy similar a un caso promedio.

Caso promedio:

Algoritmo	5000	10 000	20000	30k	400k	500k	600k	700k	800k
Quicksort con pivote final	0s	0s	0.03 s	0.08s	11.63s	18.17s	29.21s	39.37s	51.51s
Quicksort con pivote random	0s	0s	0.025s	0.06s	8.62s	13.61	20.61s	27.96s	37.44s

Peor de los casos:

Algoritmo	5000	10 000	20000	30k	400k	500k	600k	700k	800k
Quicksort con pivote final	0.19s	0.733s	2.89s	6.5s	/	/	/	/	/
Quicksort con pivote random	0s	0.007s	0.033s	0.064s	8.71s	13.3s	20.61s	27.97s	37.3s