Escuela Superior de Cómputo.
Instituto Politécnico Nacional, México.

Práctica 4: Divide y Vencerás: QuickSort.

Blancas Pérez Bryan Israel
orionmunecaycanica@gmail.com

Resumen: Aplicar la técnica "Divide y Vencerásçon el algoritmo de ordenamiento QuickSort, y encontrar el orden de complejidad en diferentes casos.

Palabras Clave: Divide y Vencerás, Partition, QuickSort.

1. Introducción

En está práctica se implementará el algoritmo de ordenamiento QuickSort. Este algoritmo es conocido por ser uno de los más rápidos, sin embargo en esta práctica, veremos que el orden de complejidad es cambiante debido al algoritmo que usa el QuickSort para apoyarse, llamado "Partition".

2. Conceptos Básicos

Partition.

Partition, es un algoritmo que divide un arreglo tamaño $n = p - r + 1$ en dos arreglos de tamaños $q - p + 1$ y $r - q$, de tal forma que $A[q] > A[i] \; \forall \; p \leq i < q$ y $A[q] < A[i] \; \forall \; q < i \leq r$.

QuickSort.

EL algoritmo QuickSort, es un algoritmo creado por el científico británico Hoare, basado en la técnica Divide y Vencerás [1]. EL algoritmo ordena de la siguiente forma:

1. Se elige un elemento de la lista, llamado pivote.
2. Se acomoda la lista de tal forma que todos los números del lado izquierdo del pivote son menores a él, y todos los números del lado derecho del pivote, son mayores a él.
3. Al quedar la lista separada en dos sublistas, se repite el proceso de forma recursiva con cada una de ellas. Hasta que cada sublista sea de tamaño 1.
4. Una vez terminado este proceso, todos los elementos estarán ordenados.

3. Experimentación y Resultados

Ejercicio 1.
Implementar el algoritmo QuickSort.

i) Mediante gráficas, muestre que el algoritmo Partition tiene orden de complejidad lineal.
Pseudocódigo del algoritmo Partition:

```
1 Partition(a,p,r)
2   x=a[r]
3   j=p-1
4   for i = p to r-1 do
5     if a[i] < x then
6       j++
7       exchange(a[j],a[i])
```

```
8     j++
9     exchange(a[j],x)
10    return j
```

El algoritmo Partition, recibe como entrada un arreglo $A[p, ..., r]$. Primeramente, elige como pivote al último número del arreglo y sitúa dos indices i y j, i apuntando en el inicio del arreglo y j=i-1. Después, compara cada elemento del resto de números del arreglo con el pivote. Si el pivote es mayor al número en el que se está actualmente, se intercambia de posición los elementos en la posición j e i. Al final de este proceso, el arreglo a, queda dividido en dos subarreglos, separados por el pivote. Como salida, Partition devuelve la posición del pivote.

Gráfica de la función Partition

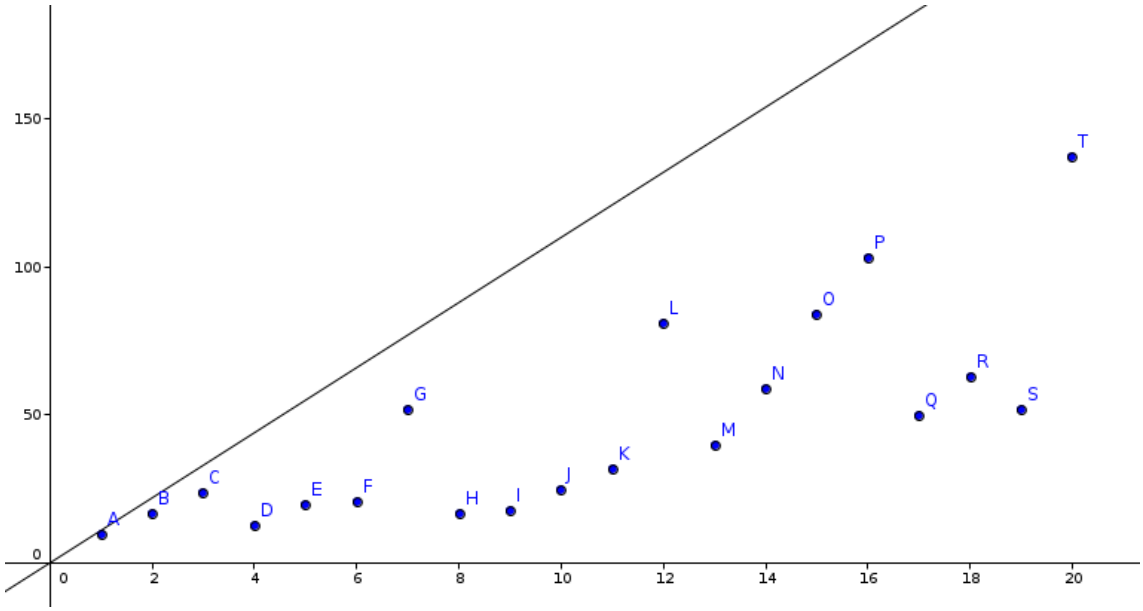

Figura 1: Ejecución del algoritmo Partition.

En la figura 1, se observa como puntos azules a las ejecuciones del algoritmo ($n\ vs\ T(n)$). Como se puede observar existe una recta $y = 11x$, la cual acota a los puntos, demostrando que el tiempo computacional del algoritmo puede variar para un mismo n, pero jamás superará su cota superior. **Por lo tanto el algoritmo Partition tiene orden lineal**.

ii) Demuestre analíticamente que el algoritmo Partition tiene orden de complejidad lineal.

Análisis del Algoritmo Partition	
Línea de Código	**Orden de complejidad**
2	$\theta(1)$
3	$\theta(1)$
4	$\theta(n)$
5	$\theta(1)$
6	$\theta(1)$
7	$\theta(1)$
8	$\theta(1)$
9	$\theta(1)$
10	$\theta(1)$

Cuadro 1: Análisis del algoritmo.

Aplicando el teorema de la suma, nos quedamos con el máximo orden de los bloques nivelados.

Por lo tanto $T(n)\ \epsilon\ \theta(n)$.

iii) Mediante gráficas, muestre que el algoritmo QuickSort complejidad $\theta(nlogn)$. (Para obtener sus conclusiones, considere diferentes valores para un arreglo tamaño n).

Pseudocódigo del algoritmo QuickSort:

```
1 QuickSort(a,p,q)
2   if p < r then
3     q = partition(a,p,r)
4     QuickSort(a,p,q-1)
5     QuickSort(a,q+1,r)
```

EL algoritmo QuickSort, recibe como parámetros de entrada, un arreglo a con su inicio (p) y el fin (r). Lo que hace el algoritmo es verificar que el arreglo a sea de tamaño mayor a uno; si se cumple esa condición, se llama a la función partition para que divida el arreglo en dos subarreglos y se vuelve

a llamar a la función QuickSort, con p y r modificados, para que siga dividiendo el arreglo hasta que sea de tamaño uno.

Gráfica de la función QuickSort

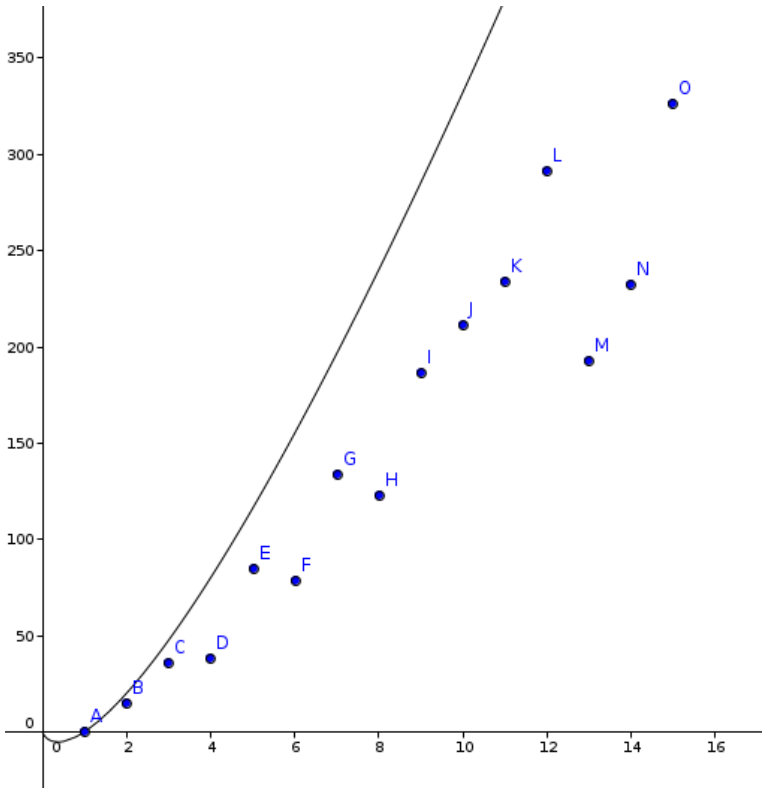

Figura 2: Ejecución del algoritmo QuickSort.

En la figura 2, se observa como puntos azules a las ejecuciones del algoritmo (n *vs* $T(n)$). Como se puede observar existe una función $f(n) = 10nlogn$, la cual acota a los puntos, demostrando que el tiempo computacional del algoritmo puede variar para un mismo n (pues depende el pivote que devuelva partition), pero jamás superará su cota superior. **Por lo tanto el algoritmo Partition tiene orden** $nlogn$.

iv) Demuestra analíticamente que el algoritmo QuickSort tiene complejidad $\theta(nlogn)$, cuando el pivote divide al arreglo por la mitad.

Análisis del Algoritmo QuickSort	
Línea de Código	**Orden de complejidad**
2	$\theta(1)$
3	$\theta(n)$
4	$T(q)$
5	$T(n-q)$

Cuadro 2: Análisis del algoritmo.

Entonces.
$T(n) = T(q) + T(n-1) + \theta(n)$
Supongamos que q divide al arreglo por la mitad.
$T(n) = 2T(n/2) + \theta(n)$.
Por los ejercicios hechos en clase.
$T(n)\ \epsilon\ \theta(n)$

v) Mediante gráficas, proponga el orden de complejidad de QuickSort cuando todos los elementos del arreglo son distintos y están ordenados de forma decreciente.

Gráfica de la función QuickSort cuando sus elementos están ordenados de forma decreciente.

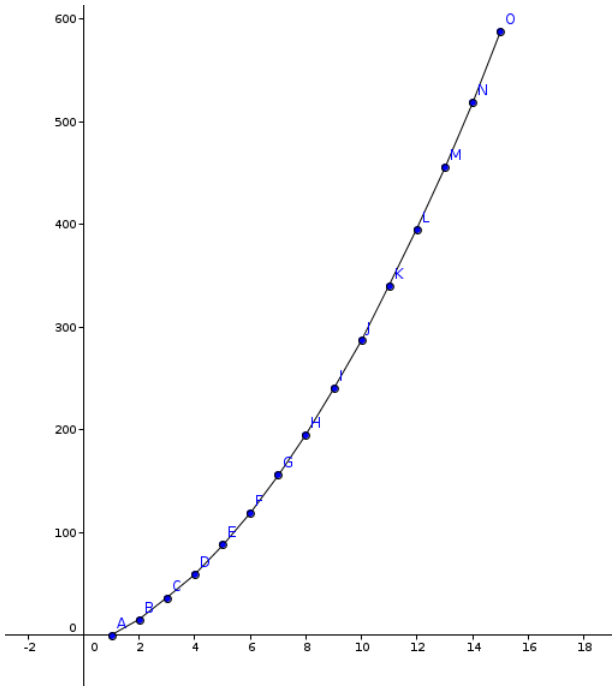

Figura 3: Ejecución del algoritmo QuickSort.

En la figura 3, se observa la gráfica (n *vs* $T(n)$). Como se puede observar el comportamiento de la gráfica es distinto cuando todos los elementos del arreglo son distintos y están ordenados de manera decreciente. **Por lo tanto el orden de complejidad del algoritmo bajo estas condiciones es** $T(n)\ \epsilon\ O(n^2)$.

4. Conclusiones

Esta práctica me pareció muy buena para darse cuenta de que no todos los algoritmos tiene un tiempo de ejecución constante. Es decir, QuickSort, como ya quedo demostrado, tiene un mejor y un peor caso. El mejor caso se da cuando el pivote termina en el centro de la lista de números, dando así un orden de complejidad de $nlogn$. Y su peor caso se da cuando los elementos están ordenados de manera decreciente, dando un orden de complejidad de n^2.
Sin duda esta práctica me gusto, por que además de todo, implementar el algoritmo tenia cierto grado de complejidad, ya que había que darse cuenta de pequeños casos que podrían impedir la buena ejecución del programa.

5. Anexo

Resolver los siguientes problemas.

i) ¿Qué valor de q retorna Partition cuando todos los elementos en el arreglo A[p, ..., r] tienen el mismo valor?.

R: 0.

ii) ¿Cuál es el tiempo de ejecución de QuickSort cuando todos los elementos del arreglo tienen el mismo valor?

R: el tiempo de ejecución se comporta de forma lineal.

6. Bibliografía

[1] https://es.wikipedia.org/wiki/Quicksort