Análisis de Algoritmos, Sem: 2018-2, 3CV1, Práctica 1, 15 de Septiembre de 2018

Práctica 4: Divide y Vencerás: QuickSort

Reyes Valenzuela Alejandro, Gutiérrez Povedano Pablo.

Escuela Superior de Cómputo
Instituto Politécnico Nacional, México
*areyesv*11@*gmail.com*, *gpovedanop*@*gmail.com*

Resumen: En este reporte se abordará el paradigma de divide y vencerás para la determinación de la complejidad del algoritmo Quicksort, tanto en la separación de los arreglos como en el propio ordenamiento.

Palabras Clave: Divide, Vence, Combina, Bloques.

1 Introducción

En la práctica anterior utilizamos el paradigma de divide y vencerás para analizar la complejidad del algortimo MergeSort, ahora reutilizaremos el paradigma para analizar el algoritmo Quicksort, el cuál también es un algoritmo de ordenamiento, y que también se apoya de una función auxiliar para funcionar, por lo que calcularemos la complejidad de ambos tanto analíticamente como gráficamente.

2 Conceptos Básicos

El paradigma de Divide y Vencerás separa un problema en subproblemas que se parecen al problema original, de manera recursiva resuelve los subproblemas y, por último, combina las soluciones de los subproblemas para resolver el problema original.

Como divide y vencers resuelve subproblemas de manera recursiva, cada subproblema debe ser más pequeo que el problema original, y debe haber un caso base para los subproblemas.

Para ello recurrimos a tres pasos fundamentales:

- **Divide:** Dividir el problema en un número de subproblemas que son instancias más pequeas del mismo problema.
- **Vence:** Resolver los subproblemas de manera recursiva. Si son los suficientemente pequeños, resolver los subproblemas como casos base.
- **Combina:** Juntar las soluciones de los subproblemas en la solución para el problema original.

Utilizaremos el paradigma para calcular el órden de complejidad del alogritmo Quicksort, el cual se auxilia de otra función denominada Partition que se encarga de separar arreglos.

Algoritmo Partition

$$
\begin{aligned}
&Partition(A[p,...,r],p,r)\\
&\quad x = A[r]\\
&\quad i = p-1\\
&\quad for(j = p; j \leq r-1; j++)\\
&\quad\quad if\ A[j] \leq x\\
&\quad\quad\quad i++\\
&\quad\quad\quad exchange(A[i],A[j])\\
&\quad exchange(A[i+1],A[r])\\
&\quad return\ i+1
\end{aligned}
$$

Algoritmo QuickSort

$$
\begin{aligned}
&QuickSort(A[p,...,r],p,r)\\
&if(p < r)\\
&\quad q = Partition(A,p,r)\\
&\quad QuickSort(A,p,q-1)\\
&\quad QuickSort(A,q+1,p)
\end{aligned}
$$

A continuacin obtendremos la complejidad de ambos algorítmos en teóricamente y prácticamente.

3 Experimentación y Resultados

Calculando el órden de complejidad de Partition:

$$
\begin{array}{ll}
Partition(A[p,...,r],p,r) & \\
\quad x = A[r] & \Theta(1) \\
\quad i = p-1 & — \\
\quad for(j = p; j \leq r-1; j++) & \Theta(r-p+1) \\
\quad\quad if\ A[j] \leq x & — \\
\quad\quad\quad i++ & — \\
\quad\quad\quad exchange(A[i], A[j]) & — \\
\quad exchange(A[i+1], A[r]) & \Theta(1) \\
\quad return\ i+1 & —
\end{array}
$$

Podemos ver que el primer y último bloque se ejecuta un número constante de veces cada vez que la función es llamada, por eso su complejidad es $\Theta(1)$.
En el caso del ciclo for el número de veces que se ejecutará está dado por los valores pivote (p y r) que representan el primer y el último elemento del arreglo, por lo que está definido por la cantidad de elementos del mismo, por lo que tenemos que r-p+1=n, por lo tanto $\Theta(r-p+1) = \Theta(n)$. Al juntar la solución de todos los bloques concluimos que $Partition(A[p,...,r],p,r)\ \epsilon\ \Theta(n)$.
Ahora calcularemos la complejidad de QuickSort:

$$
\begin{array}{ll}
QuickSort(A[p,...,r],p,r) & \\
if(p < r) & \\
\quad q = Partition(A,p,r) & \Theta(1) \\
\quad QuickSort(A,p,q-1) & T(\frac{n}{2}) \\
\quad QuickSort(A,q+1,p) & T(\frac{n}{2})
\end{array}
$$

Podemos observar que existe recursividad en nuestra función, por lo que procederemos a calcular la ecuación de recurrencia. Para ello consideramos que nuestros arreglos son divididos en dos de la misma longitud.

$$
T(n) = \begin{cases} \Theta(1) & si \quad n = 1 \\ \\ 2T(\frac{n}{2}) + \Theta(n) & si \quad n > 1 \end{cases}
$$

Ahora resolveremos la recurrencia, como tenemos un cociente, debemos asegurarnos que nuestro resultado sea entero, para ello se propone el cambio de variable $n = 2^k$, donde $k = \log_2(n)$, por lo que ahora resolveremos:

$$
T(n) = 2T(\tfrac{n}{2}) + C(n) = 2T(2^{k-1}) + C(2^k)
$$

Haciendo iteraciones:

$$T(2^k) = 2(2T(2^{k-2})+C(2^{k-1}))+C(2^k)$$

Llegando al término i:

$$T(2^k) = 2^i((T(2^{k-i})+iC(2^k))$$

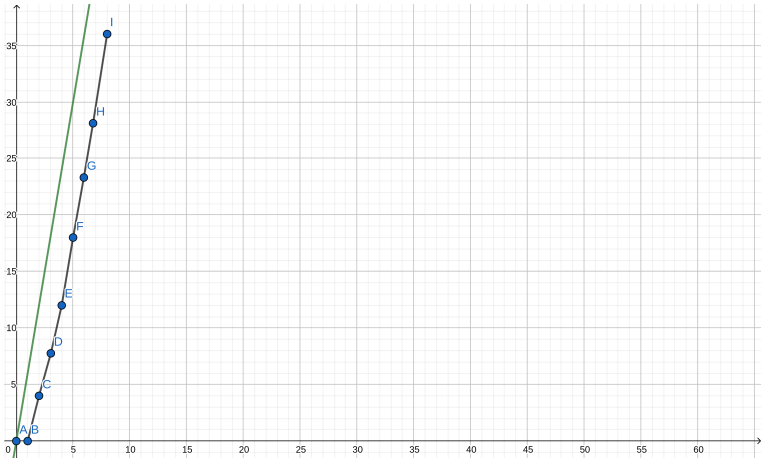
Llegando a la condición de frontera:

$$T(2^k) = 2^k((T(1)+kC(2^k))$$

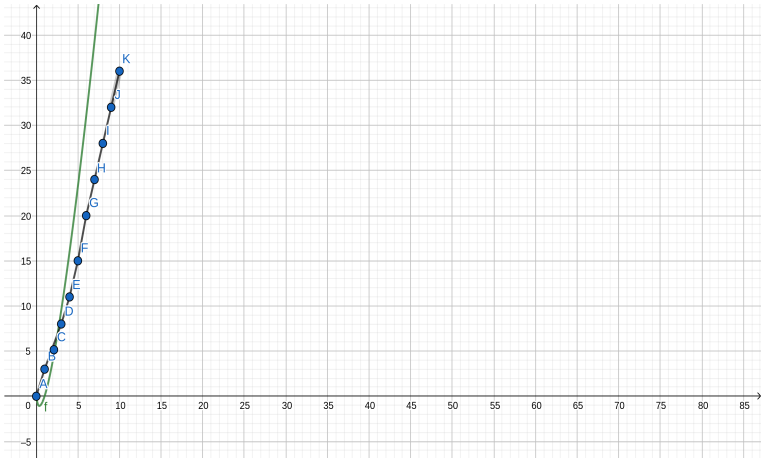
Finalmente:

$$T(2^k) = c(n) + \log_2(n) + c(n)$$

Por lo que QuickSort ϵ $\Theta(n \log_{10}(n))$. Ahora comprobaremos lo establecido mediante gráficas: Algoritmo Partition (comparando los tiempos de longitud 1 a 10 con n=6n)

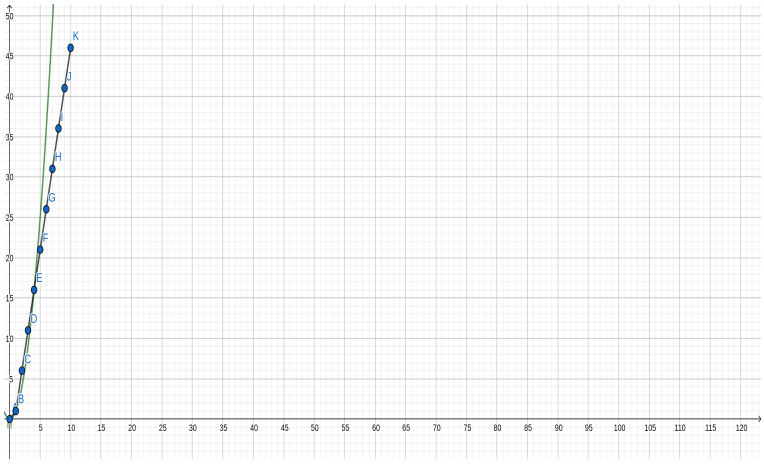


Algoritmo QuickSort (comparando los tiempos de longitud 1 a 10 con n=2nlog(n)



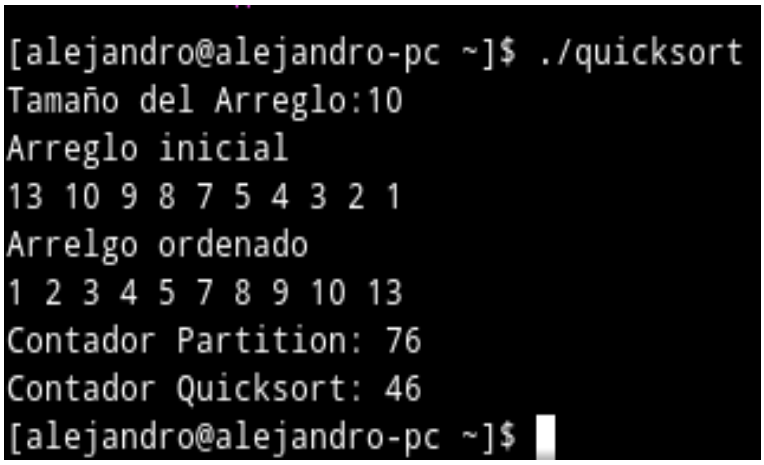
Podemos ver que las funciones propuestas acotan a los puntos de las funciones Partition y Quicksort, por lo que comprobamos que QuickSort $\epsilon\ \Theta(n \log_{10}(n))$ y que Partition $\epsilon\ \Theta(n)$.

Ahora, propondremos el arreglo [13,10,9,8,7,5,4,3,2,1] para graficar Quicksort. Para graficar en este caso quitaremos elementos de derecha a izquierda para las longitudes del arreglo. Acotamos con la función n^2



Podemos observar que la funcin cuando tiene sus elementos ordenados de manera decreciente tiene complejidad cuadrática.

Prueba de escritorio:



4 Conclusiones

Conclusión General: Observamos que el paradigma fue bastante útil, ya que nos ahorramos el análisis línea por línea y nos enfocamos únicamente a los bloques clave, siend en este caso los ciclos los que nos permitieron saber la complejidad de Partition y la recurrencia logarítmica para Quicksort. Sin embargo, se consideró un caso donde la complejidad para Quicksort sea de la forma solicitada en la práctica, ya que existen casos donde la complejidad se puede tornar cuadrática.

Conclusión Gutiérrez Povedano: Utilizando el paradigma divide y vencers se puede resolver un problema y generar un algoritmo cuya complejidad sea menor como en el caso del algoritmo merge sort el cual tiene un orden de complejidad Θ(n*log(n)) para realizar el ordenamiento de forma ascendente de un arreglo de enteros el cual es menor al orden de complejidad de otros algoritmos que resuelven el mismo problema por ejemplo el algoritmo burbuja cuya complejidad en el peor caso es $O(n^2)$.

Conclusión Reyes Valenzuela: En este algoritmo tuve algunos problemas para el caso del exchange, ya que originalmente sólo utilizaba los valores de las variables para realizarlos, sin embargo, al momento de realizar el ordenamiento, no los tomaba en cuenta y era como si no hubiese hecho nada, por lo que utilicé apuntadores para solucionar esto. Por otro lado, el algoritmo tiene un funcionamiento similar al propuesto en la práctica anterior, por lo que se esperaba que tuviesen la misma complejidad.

5 Anexo

Resolver los siguientes problemas:

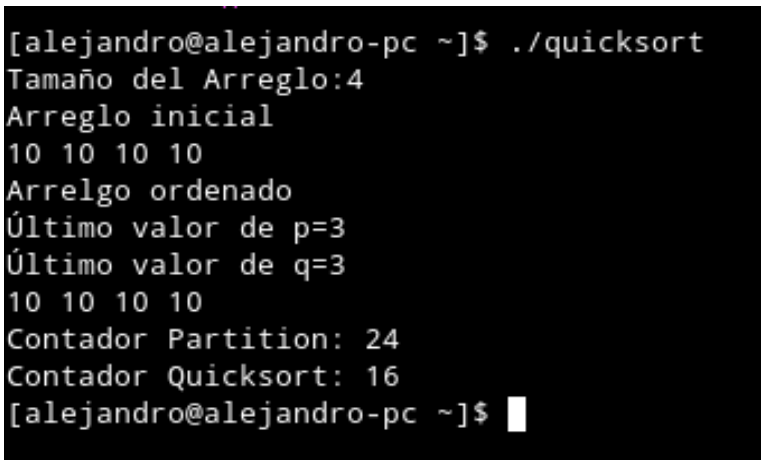
1. Qué valor de q retorna Partition cuando todos los elementos en el arreglo A[p, ..., r] tienen el mismo valor?.

2. Cuál es el tiempo de ejecución de QuickSort cuando todos los elementos del arreglo tienen el mismo valor?.

Soluciones

1. Debido a que los valores son iguales, el contador de la función en p inicia en teoría en -1 (dado que el índice p siempre inicia en 0) la condicíon del if siempre se cumnplirá, debido a esto aumentará por cada elemento, al ser r el índice n-1, p aumentará n-1 veces, por lo que podemos decir que al final Partition retornará el mismo valor asociado a r.

2. El tiempo de ejecución es lineal, dado que el número de comparaciones será de alguna regido únicamente por el tama *n*o del arreglo y no por los valores de este.



6 Bibliografía

Khanacademy.org (2018). [online] Available at: https://es.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/divide-and-conquer-algorithms. [Accessed 13 Sep. 2018].

Slideshare.net (2018). [online] Available at: https://es.slideshare.net/balamoorthy39/divide-and-conquer-quick-sort. [Accessed 13 Sep. 2018].