Práctica 3: Quick Select. Programación Dinámica

Pablo Sánchez Redondo. David Kaack Sánchez

Pareja 03

I. El Problema de la selección

1. Aplicando merge-sort a una lista de 5 elementos, la lista se parte en dos listas de tamaño 2 y 3. Ordenar la lista de 2 resulta en una comparación. La de 3 se parte en listas de un elemento y dos elementos. Ordenando la lista de dos elementos y luego fusionar con la lista de un elemento resulta en tres comparaciones. Fusionando las dos listas ordenadas se necesita un máximo de 4 comparaciones. En total son 8 comparaciones.

   Podemos reducir las comparaciones totales para encontrar la mediana cambiando la estrategia en el fusionamiento de las sublistas de 2 y 3 elementos. Para ello comparamos el primer elemento de la lista de 2 con el elemento medio de la lista de 3. Pueden pasar varias cosas:

   s1 = [a,b], s2 = [c,d,e]

   Compara a y d

   Opción 1: a > d:

   List = [c,d,a,e] y mediana = a, porque sabemos que b > a. Metiendo b no cambia la posición de a.

   Opción 2: a < d:

   List = [c,a,d,e],

   Compara b y d:

Opción 1: b > d:

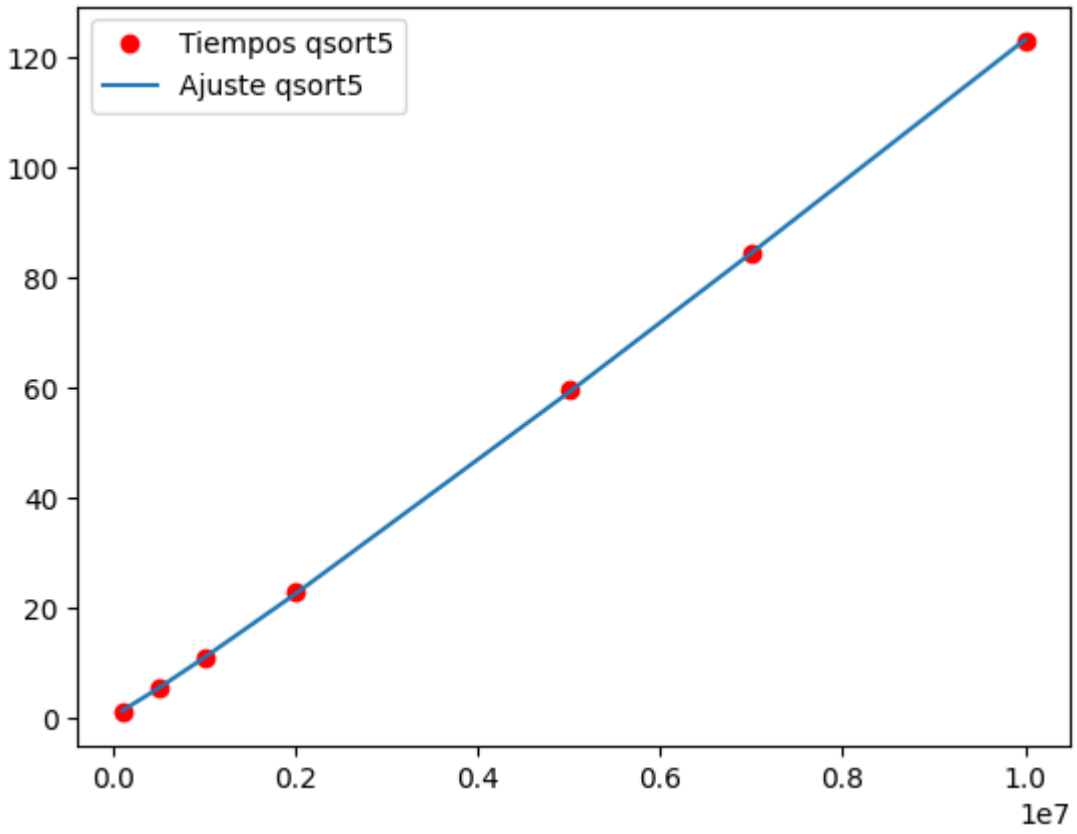
List = [c,a,d,…] y mediana = d porque comparando b y e no cambia posición de d.

Opción 2: b < d:

List = [c,a,b,d,e] y mediana = b.

Como se puede ver solo necesitamos un máximo de dos comparaciones para averiguar la media, lo que baja el número de comparaciones total a 6 como máximo.

2. La diferencia entre el peor caso de Quicksort y el peor caso de Quicksort con pivot 5 es que el método pivot 5 nos permite escoger el pivote más en el medio posible. Pudiendo así evitar el peor caso de Quicksort, que sería elegir el número más grande o pequeño siempre. Dejando el peor caso en algo que se acerca más caso medio de Quicksort que es O(n*logn).



II. Programación dinámica

1. Para los algoritmos de programación dinámica implementados el espacio requerido es de $(k+1)^2$. Donde *k* es el número de elementos que tenemos, ya sean monedas o cosas que meter en la mochila.

   Para reducir el coste en memoria, como sólo nos importa el valor inmediatamente encima del que se está calculando en la tabla, podemos usar una tabla de una sola fila, e ir sobrescribiendo los valores calculados en la misma.

2. Para este segundo caso, la fórmula de una solución sería una formula recursiva tal que:

   dp[w] = max(dp[w-peso[I], i] + valor[i], dp[w,(i-1)])

   Efectivamente, calculando el valor óptimo de cada mochila de peso *w*. Una vez calculado, se elijen los pesos que maximizan el valor final.