Práctica 2: Análisis y corrección de los algoritmos

Jorge Aznar López - 721556 Abel Chils Trabanco - 718997

1. Introducción a los algoritmos utilizados

El objetivo es analizar la ejecución del algoritmo "Quicksort" según tres criterios de selección del pivote que divide el vector original en los dos vectores a los cuales se aplicarán los mismos pasos de ordenación recursivamente hasta tener el vector ordenado.

Los tres criterios son:

- Criterio 1. El pivote es el elemento correspondiente al índice mínimo o máximo.
- Criterio 2. El pivote es la mediana, calculada con un algoritmo de coste asintótico lineal en el caso peor ("Insertion Sort")
- Criterio 3. El pivote se elige de forma aleatoria, utilizando un generador de números pseudo-aleatorios para determinar el índice del mismo.

2. Corrección de los algoritmos

➢ Criterio 1

La corrección del algoritmo según el criterio 1 se apoya en dos pilares fundamentales que demuestran su correcto funcionamiento.

El primero es que el algoritmo divide dado un vector y un pivote calculado según el criterio uno, ordena el elemento pivote de manera correcta, posicionando a su izquierda los elementos menores y a su derecha los mayores pasando de tener "n" elementos desordenados a "n-1".

El segundo es que el algoritmo que invoca dicha función realiza este proceso recursivamente tanto sobre la izquierda del pivote ya ordenado como su derecha, dejando así "n - 3" elementos desordenados, equivalente a "$n - 2^0 - 2^1$", esto se realiza de manera recurisva hasta quedar dos elementos por ordenar, los cuales se ordenan por inserción.

Explicado esto se puede concluir que el vector quedara ordenado tras tener "n - n" elementos desordenados donde "$n = 2^0 + 2^1 + 2^2 + 2^3 + ... + 2^k$"

➢ Criterio 2

La corrección del algoritmo según el criterio 2 es la misma que la del criterio 1, pues el algoritmo de división no varía, pero se debe tener en cuenta que el pivote es la mediana del vector, y está debe ser calculada mediante el algoritmo de ordenación por inserción, fragmentando el vector original en sub-vectores de 5 elementos (o menos hasta un mínimo de 1 elemento) y hallando sus medianas, para posteriormente aplicar este algoritmo al vector creado con las medianas obtenidas para recursivamente hallar la mediana del vector original, e utilizarla como pivote (realmente se hallará el índice, y de ahí de extraerá la mediana).

Se centrará la corrección de este proceso al algoritmo que selecciona la mediana, puesto que de otro modo, si no se hallase correctamente la misma, el algoritmo original no funcionaría de acuerdo a lo especificado.

El algoritmo de ordenamiento por inserción (acotado a 5 elementos), basa su funcionamiento en el hecho de que inicialmente sólo existe un elemento ordenado (el primero) y que cuando existen "n" elementos ordenados, se selecciona el elemento "n+1", se compara con los elementos ordenados y se inserta en el punto en el cual existe un elemento menor que el mismo, obteniendo así de nuevo un vector de "n" elementos ordenados, aplicando este proceso iterativamente de ordena el vector y una vez ordenado el sub-vector de 5 elementos, se extrae la mediana como el elemento central del sub- vector y se añade a las medianas existentes, en caso de ser un sub-vector con menos de 5 elementos, por interés, se extraería el elemento medio de índice menor.

➢ Criterio 3

La corrección del algoritmo según el criterio 3 es la misma que la del criterio 1, pues el algoritmo de división no varía, pero se debe tener en cuenta que el pivote es un elemento aleatorio seleccionado generando un índice mediante números pseudo-aleatorios, lo cual no influye en su corrección pero si en su coste.

3. Análisis del coste

➢ Criterio 1

La selección del pivote en este caso se corresponde con el peor caso posible para el algoritmo Quicksort, debido a que elegir el elemento de índice mayor o menor supone una partición desequilibrada, y si consecutivamente se realizaran particiones desequilibradas el coste sería equivalente a ordenar sobre "n" elementos con un coste "c" de realizar el ordenamiento, luego "c*n-1", y sucesivamente hasta "c*2" elementos, donde la suma de los costes equivaldría a "cn+c(n-1)+...+2c", lo cual equivale a la serie aritmética, cuyo coste es $\Theta(n^2)$.

En el mejor caso posible, las particiones se llevarían a cabo equilibradamente, y en cada llamada recursiva tuviera a lo más "n/2" elementos a ordenar (la otra "n/2 - 1"), lo cual es similar al algoritmo "Merge Sort", el cual es de complejidad $\Theta(n \log(n))$.

➢ Criterio 2

Este criterio selecciona la mediana como pivote, lo cual equivale a que en la primera partición tendremos los elementos separados según se ha explicado en el caso mejor, de manera equilibrada, lo cual daría lugar a un coste $\Theta(n \log(n))$, pero esto no es así, puesto que se ha comprobado que este algoritmo invierte la mayor parte de su tiempo en la búsqueda de esta mediana, lo cual eleva innecesariamente el coste.

El algoritmo de búsqueda de la mediana está basado en dividir el vector en fragmentos de 5 elementos máximo y realizar sobre ellos una ordenación por inserción para obtener su mediana, y recursivamente hallar sobre estas medianas, la nueva mediana, hasta obtener el elemento que hará de pivote.

Este algoritmo tiene coste aproximado en $\Theta(n)$, pues el algoritmo de ordenación por inserción esta acotado a 5 elementos, lo cual al no existir elementos repetidos, dará lugar a un máximo de 2 elementos mayores que la mediana a seleccionar, lo cual reduce bastante el tiempo que originalmente tomaría el algoritmo $\Theta(n^2)$ a $\Theta(n)$.

➢ Criterio 3

Este caso del algoritmo funciona particularmente de manera similar al primer criterio, debido a que al elegir aleatoriamente el pivote, e intercambiarlo, existe una determinada posibilidad de que el elemento elegido este en el medio del vector, lo cual aumenta a más elementos del vector seleccionemos para intercambiar.

Explicado esto, se puede concluir que este algoritmo se asemejaría a un caso algo mejor, que el intermedio para este algoritmo, y el caso intermedio equivaldría a realizar particiones casi equilibradas, lo cual daría lugar a un coste $\Theta(n \log(n))$, pues incluso si la mitad de las veces se tuviera el peor caso de la división y la otra mitad una división de "3 a 1", por ejemplo, el tiempo de ejecución sería alrededor del doble del tiempo de ejecución de siempre obtener una división de "3 a "1, lo cual es solo un factor constante y se absorbe en la notación O grande. Así que sí se alterna entre el peor caso para las divisiones y el caso de "3 a 1", el tiempo de ejecución es $\Theta(n \log(n))$.

4. Conjunto de pruebas realizadas

Para realizar las pruebas que verifiquen el correcto funcionamiento del algoritmo según los tres criterios definidos, se ha programado un script en bash el cual ejecuta un número "x" seleccionado por el programador de iteraciones en las cuales primero se genera el vector, se ordena según el criterio y se comprueba si esta correctamente ordenado, informando de la correcta ejecución del algoritmo con un mensaje [SUCESS] o deteniendo el programa e informando del criterio que ha dado error con un mensaje [ERROR] : "criterio".

Se decidió realizar una comparativa entre los tres criterios para distintos tamaños del vector, con un número fijado de iteraciones en 100, y el resultado dado con una precisión de nanosegundos. Esta fue la tabla obtenida:

Coste medio del algoritmo (ns) según el criterio para 100 iteraciones

	TAMAÑO DEL VECTOR			
CRITERIO	1000	10000	100000	1000000
1	243231	3436310	45035204	569415253
2	20041230	249592580	2813430851	34533744443
3	300690	3759936	46849040	584117172