Workshop 3: Best, Worst, and Average Case Analysis Experiments

Santiago Mercado David Salgado

Abstract

En clase hemos analizado las diferencias en tiempo y efectividad del mejor, el peor y el promedio de los casos para diferentes algoritmos. El objetivo de este laboratorio es analizar estos 3 casos en un algoritmo que ya teníamos implementado, este algoritmo crea un archivo con N enteros aleatorios, luego todas esas matrices se transfieren a una matriz. El programa lee cada elemento de la matriz y almacena los números únicos en una matriz diferente. También cuenta la cantidad de veces que aparece cada número único e imprime la información al usuario. Este proceso se realiza al menos 200 veces para cada archivo con n enteros para encontrar el tiempo de ejecución promedio para el programa dado. También fuerza el mejor de los casos (todos los N enteros son 0) para analizarlo y compararlo con el promedio y el peor de los casos (N números diferentes).

I. Introducción

"Actualmente, se ha hecho más visible la necesidad de contar con algoritmos más eficientes para nuestras aplicaciones, ya sea por la cantidad de datos procesados o por la necesidad de respuestas rápidas." [**?**]. El análisis de los algoritmos y su eficiencia respecto a memoria usada y tiempo de ejecución es algo extremadamente importante hoy en día. Tener la posibilidad de mejorar tu algoritmo es algo que siempre debemos buscar como ingenieros de sistemas para proporcionar a nuestros clientes los mejores resultados. Por esa razón, es necesario analizar en que situaciones nuestros algoritmos presentan una eficiencia mayor que en otros casos en promedio y poder proporcionar diferentes casos y saber el comportamiento de nuestro algoritmo.

II. Definición del Problema

Adaptar la implementación del algoritmo que cuenta los duplicados en un archivo de texto sin formato para que devuelva la cantidad de comparaciones y el tiempo transcurrido necesario para lograrlo. Posteriormente es necesario graficar los resultados del tiempo y comparaciones en gráficas para poder observar y analizar el comportamiento de nuestro algoritmo en los diferentes casos.

Algorithm 1 illustrates the iterative algorithm for counting duplicates in an array/text file..

Algorithm 1 Counting Duplicates

while $z < tamañovector$ **do**
  $x \leftarrow vectorprincipal[z]$
  $z++$
  $pos \leftarrow 0$
  $keep \leftarrow true$
  while $keep = true$ **and** $pos < cantidadenterosunicosactuales$ **do**
    if $enteros_unicos[pos] = x$ **then**
      $contador[pos]++$
      $keep \leftarrow false$
    else
      $pos++$
    end if
  end while
  if $actualsize = pos$ **and** $keep = true$ **then**
    $contador[cantidadenterosunicosactuales] \leftarrow x$
    $enterosunicos[cantidadenterosunicosactuales]++$
    $cantidadenterosunicosactuales++$
  end if
end while

III. Metodología

Para desarrollar el problema, lo primero que hace el algoritmo es escribir un archivo con n números enteros aleatorios. Posteriormente, se toma este archivo y se leen los distintos números en él hasta llegar el final del archivo. Cada entero se almacena en un vector al que llamaremos vectorprincipal. Luego, se crean 2 vectores de apoyo inicializados en 0, uno contador al que llamaremos contador, y otro con cada número distinto en el texto que llamaremos enterosunicos. Con esto, se recorren todos los enteros del vector principal uno por uno. Se verifica si ese entero es distinto a los almacenados o si es un repetido al compararlo con todos los números del vector enterosunicos, que se recorre desde el inicio hasta la última posición con un número único (esta posición se almacena en la variable (cantidadenterosunicosactuales).

De esta forma si hay un número repetido se hacen menos comparaciones que con uno nuevo. Se aumenta el contador correspondiente al número en el vector contador y si es un número nuevo se añade al vector enterosunicos.Esto se evidencia en el algoritmo anterior.

Este algoritmo se corre para los 3 posibles casos: mejor caso, caso promedio y peor caso. Para el mejor caso, se crea un archivo con n cantidad de 0; para el caso promedio el archivo tiene n números aleatorios que van desde 0 hasta n; y para el peor caso el archivo tiene n números distintos. Para determinar el comportamiento de la gráfica tiempo vs n para cada caso, se ejecuta el proceso anterior contando cada condicional (Si) que utiliza. Este procedimiento se realiza 300 veces, y se halla un promedio del tiempo de ejecución y complejidad (uso de condicionales). Para el mejor caso se realiza variando el valor de n desde 32 hasta 2000000 aumentando n al doble cada vez. En cambio, para el caso promedio y el peor caso los valores de n van desde 32 hasta 5000, aumentando n en factor de 3/2. Esta diferencia se debe a que un mayor valor de n produciría un tiempo de ejecución muy alto (hace overflow en la variable) y aumentar n al doble en lugar de un factor de 3/2 causaría que la gráfica tuviera muy pocos datos.

Luego, los datos obtenidos se guardan en un archivo con 3 columnas: n, comparaciones, tiempo de ejecución. De esta forma se tienen 3 archivos (uno por cada caso), los cuales son usados para graficar los resultados de n vs tiempo con la ayuda de Python y MatLab. Por último, se hace un ajuste de la gráfica para determinar su comportamiento y sacar las conclusiones del experimento.

IV. Results

Para los resultados fueron necesarios escribirlos dentro un TXT distinto para cada caso y además diferente al que estábamos leyendo dentro de nuestro algoritmo. Dentro de estos TXT de resultados se presentan 3 columnas. La primera columna hace referencia a la cantidad de números que se encuentran dentro del TXT que estamos leyendo. La segunda columna hace referencia a la cantidad de comparaciones realizadas durante todo el algoritmo y finalmente la tercera columna hace referencia al tiempo que se demoró el algoritmo en realizar todo el trabajo solicitado. Posteriormente se presentarán los resultados para los 3 diferentes casos.

Mejor Caso:

Los resultados obtenidos para el mejor caso se presentan donde todos los números dentro del texto son igual a 0, estos resultados se presentan a continuación:

TABLE I
El número de operaciones y del tiempo transcurrido (nanosegundos) como una función del tamaño de entrada.

Size	Operations	Elapsed time
32	63	3016
64	127	2420
128	255	1456
256	511	1613
512	1023	2194
1024	2047	4243
2048	4095	7209
4096	8191	12951
8192	16383	27492
16384	32767	50415
32768	65535	96842
65536	131071	191310
131072	262143	390439
262144	524287	766497
524288	1048575	1643614
1048576	2097151	3789424

Una gráfica del número promedio de operaciones y tiempo de ejecución para el mejor caso se muestra en la figure 3.

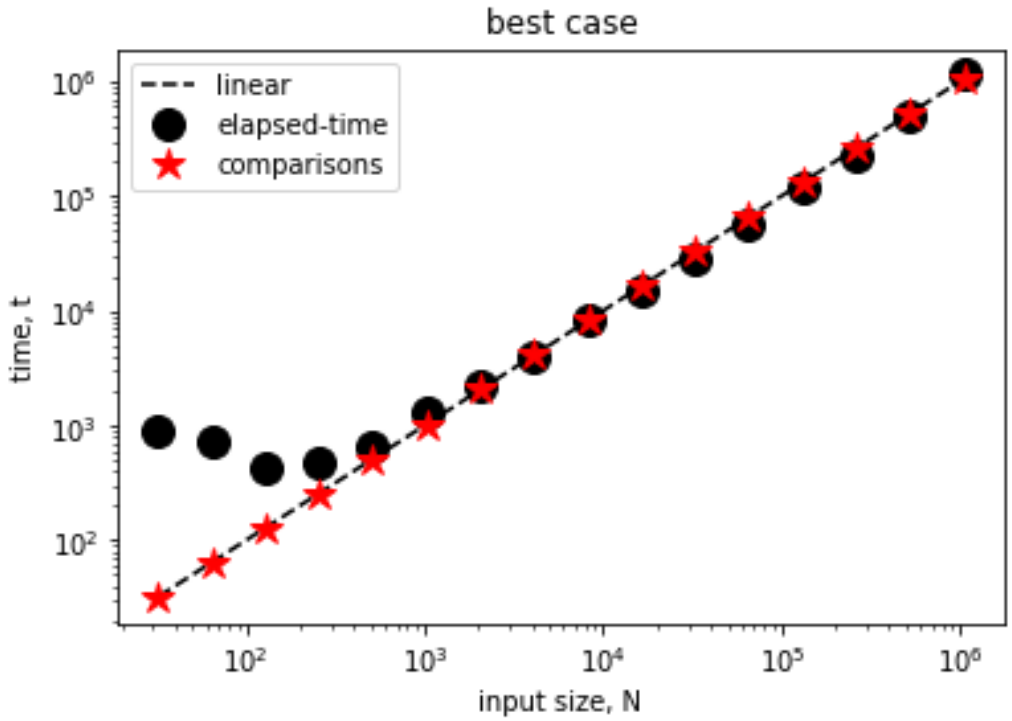

Fig. 1. Número promedio de operaciones y tiempo de ejecución como función del tamaño de entrada para el mejor caso.N.

A primera vista, se puede observar que en un principio los datos no demuestran ningún tipo de comportamiento, pero a medida que se incrementan los datos estos conforman un comportamiento lineal respecto al número de comparaciones realizadas en el algoritmo.

Caso promedio:

En este caso ya los números dentro del TXT a leer son completamente aleatorios, ya que son aleatorios se realizaron una menor cantidad de repeticiones y hasta un límite menor de números dentro del TXT a leer. Los resultados se muestran a continuación:

TABLE II
EL NÚMERO DE OPERACIONES Y DEL TIEMPO TRANSCURRIDO (NANOSEGUNDOS) COMO UNA FUNCIÓN DEL TAMAÑO DE ENTRADA.

Size	Operations	Elapsed time
32	320	2733
48	703	1909
72	1540	3017
108	3417	4789
162	7577	8642
243	17013	16854
364	38086	34028
546	84813	70789
819	191679	153415
1228	429594	333165
1842	962638	735745
2763	2169014	1616289
4144	4879156	3628981

Una gráfica del número promedio de operaciones y tiempo de ejecución para el caso promedio se muestra en la figure 3.

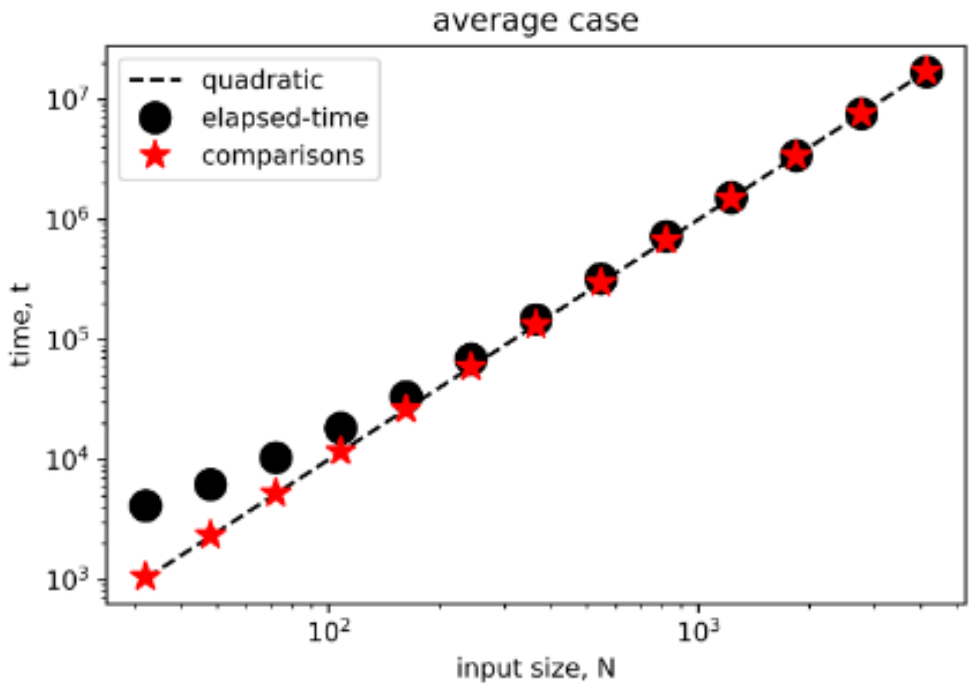

Fig. 2. Número promedio de operaciones y tiempo de ejecución como función del tamaño de entrada para el caso promedio.N^2.

Como se puede observar en la gráfica, el comportamiento de los datos se parece al cuadrático casi desde el principio de la toma de datos. Esto se debe a que es lógico pensar que en un mayor número de comparaciones se va a tomar mayor tiempo realizar el algoritmo.

Peor caso:

Dentro del peor caso también se presentan números aleatorios en el TXT a leer, por lo que en la práctica se realiza el mismo procedimiento que para el caso promedio. Los resultados se presentan a continuación:

TABLE III
EL NÚMERO DE OPERACIONES Y DEL TIEMPO TRANSCURRIDO (NANOSEGUNDOS) COMO UNA FUNCIÓN DEL TAMAÑO DE ENTRADA.

Size	Operations	Elapsed time
32	528	1472
48	1176	2286
72	2628	3731
108	5886	6795
162	13203	13253
243	29646	26586
364	66430	55426
546	149331	118454
819	335790	260113
1228	754606	574604
1842	1697403	1280200
2763	2834846	2834846

Una gráfica del número promedio de operaciones y tiempo de ejecución para el peor caso se muestra en la figure 3.

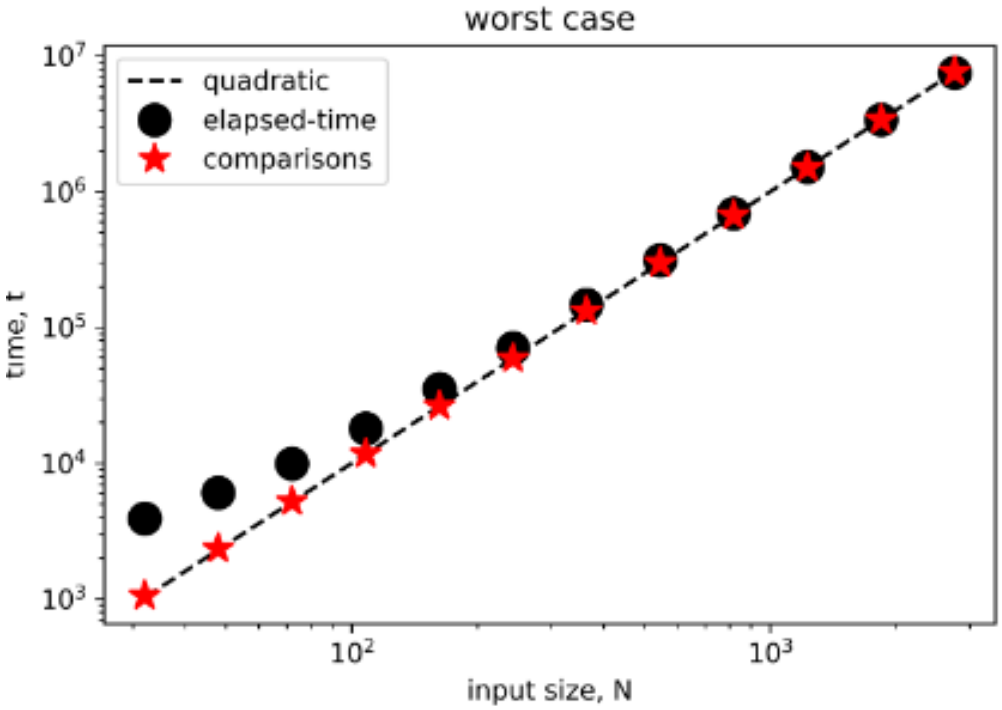

Fig. 3. Número promedio de operaciones y tiempo de ejecución como función del tamaño de entrada para el peor caso.N^2.

Como se puede evidenciar arriba, la gráfica a simple vista se ve muy parecida sino igual al del caso promedio. Estos resultados presentan el mismo comportamiento cuadrático para los datos.

V. Conclusion

En conclusión, podemos decir que todos los objetivos de este laboratorio se cumplieron de manera exitosa al poder observar y analizar el comportamiento de los datos (input size, comparaciones y tiempo de ejecución) para los 3 diferentes casos. Como se logra evidenciar en los resultados de todos los casos, a mayor es el input size (n) mayor es el tiempo de ejecución y también es mayor el número de comparaciones. Sin embargo, el comportamiento de las 3 variables varía según el caso. Para el Mejor caso, podemos concluir que con el aumento de los elementos se aprecia un comportamiento lineal, demostrando que el número de comparaciones es directamente proporcional al tiempo de ejecución del algoritmo. Por otro lado, para el caso promedio y el peor caso a pesar de que las gráficas parecen que tuvieran un comportamiento lineal, esto es simplemente debido al ajuste de la gráfica con la ayuda de Python. El comportamiento que tienen en realidad es cuadrático demostrando que también así la proporcionalidad entre el número de comparaciones y el tiempo de ejecución, sin embargo, esta proporcionalidad se muestra a mayor escala ya que a medida que aumenta el número de comparaciones, el tiempo de ejecución también aumenta, pero a mayor escala. Por otro lado, durante este laboratorio no se nos presentaron muchos inconvenientes específicos, simplemente fue necesario aprender el algoritmo para crear, editar y leer archivos .TXT para posteriormente utilizar el algoritmo otorgado por el profesor en Python para graficar nuestros resultados y lograr nuestros objetivos. Finalmente se lograron todos los resultados esperados y que se habían mencionado en clase, concluyendo así el laboratorio satisfactoriamente.

References