Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas
Estructuras de Datos
Taller 1: complejidad, 2024-10

1 Objetivo

Analizar la ejecución de los algoritmos:

- Exponencial de un número (versión iterativa y recurrente).
- Ordenamiento de un vector de n elementos (*bubblesort*, *quicksort* y *heapsort*).

2 Desarrollo del taller

El desarrollo del taller consistirá en generar un informe escrito (en formato `.pdf`, ***cualquier otro formato implica una nota final de 0.0/5.0***), el cual debe enviarse a través de la respectiva asignación de BrightSpace. Junto con este enunciado se entrega un archivo comprimido que contiene el conjunto de programas a usar. Descomprima el archivo en una ubicación adecuada (fácil de acceder por línea de comandos) y utilizando la consola o línea de comandos ubíquese en la carpeta que acaba de descomprimir.

2.1 (15%) Análisis inicial

Estudie los archivos de código fuente que se adjuntan (`*h`, `*.hxx`, `*.cxx`). Responda en la primera parte de su reporte las siguientes preguntas:

1. **(40%)** Los programas escritos en los archivos "`test_*.cxx`" ¿qué hace cada uno?
   Nota: La respuesta para cada programa no debe exceder de 30 palabras.

La secuencia de comandos que se debe usar en este taller se encuentra en el archivo "`comandos.txt`". Analice los comandos allí presentes e identifique la sintaxis de cada uno. Tenga en cuenta que algunos comandos dependen del sistema operativo o entorno en el cual se está trabajando. Intente comprender qué resultado genera cada uno de los comandos. Para esto, copie y pegue cada comando individualmente en la consola de comandos y espere los resultados (algunos pueden tardar varios segundos o minutos).

2. **(30%)** ¿Cuántos archivos nuevos aparecieron luego de ejecutar todos los comandos? ¿Qué nombres tienen?
3. **(30%)** ¿Qué parecen contener o representar esos nuevos archivos? Genere una respuesta para cada archivo individual.

2.2 (35%) *Exponential*

La primera parte del archivo "`comandos.txt`" se refiere al análisis del algoritmo para calcular la potencia de un número en sus versiones "recurrente" e "iterativa". Para esto, se utiliza el programa compilado "`my_test_exponential`", el cual recibe como parámetros el valor de la base y el valor de la potencia, y genera una salida con 5 valores: el valor de la potencia, el resultado del algoritmo de exponenciación recurrente, el resultado del algoritmo de exponenciación iterativo, el tiempo de ejecución del algoritmo recurrente y el tiempo de ejecución del algoritmo iterativo. Este programa puede ejecutarse en la consola así:

```
En Linux, Mac y similares
$ ./my_test_exponential base potencia
En Windows
> .\my_test_exponential.exe base potencia
```

Ejemplo:

```
$ ./my_test_exponential 50 150
150 7.00649e+254 7.00649e+254 0 0
```

Para realizar el análisis empírico de la complejidad del algoritmo en sus dos versiones, el programa se ejecuta varias veces con la misma base (3) pero diferentes valores de exponente (entre 0 y 350, en este caso) para generar una tabla con los respectivos valores de ejecución ("`exponential_table.txt`"). Finalmente, es necesario utilizar algún programa para graficar datos (Microsoft Office, OpenOffice, LibreOffice, gnuplot, etc...) y los datos almacenados en la tabla para graficar el comportamiento de los algoritmos en términos de sus tiempos de ejecución (en nanosegundos, columnas 4 y 5 de la tabla).

A partir de esta imagen, escriba la segunda parte de su reporte respondiendo las siguientes preguntas:

1. **(45%)** Analice la gráfica. En otras palabras, escriba un párrafo donde demuestre cuál algoritmo es mejor, apoyándose únicamente en la información de la gráfica, la cual debe incluírse como imagen en su informe final.
2. **(40%)** ¿Cuál es la complejidad teórica (notación $O(n)$) de los dos algoritmos? Demuestre el cálculo manual del tiempo de ejecución para los dos algoritmos y relacione el análisis de la gráfica con esta información.
3. **(15%)** Calcule 6^{396} y 6^{397} con el programa compilado. ¿Qué resultados se obtienen? ¿Ambos resultados son correctos? ¿Por qué? ¿Si no son correctos, qué se puede hacer para obtener los resultados adecuados?

2.3 (50%) *Sort*

La segunda parte del archivo "`comandos.txt`" se refiere al análisis de tres algoritmos de ordenamiento diferentes (*bubblesort*, *quicksort* y *heapsort*), ejecutados sobre arreglos de números enteros en 3 configuraciones diferentes: arreglos completamente desordenados, arreglos ordenados "al contrario" (descendentemente) y arreglos ya ordenados (ascendentemente). Para esto, se utilizan los programas compilados "`my_test_sort_random`", "`my_test_sort_inverse`" y "`my_test_sort_sorted`", los cuales reciben como único parámetro el tamaño del arreglo a ordenar, y generan cada uno una salida con 7 valores: el tamaño del arreglo, la verificación de ordenamiento con cada uno de los algoritmos (`1` si el arreglo queda ordenado o `0` si no queda ordenado al finalizar la ejecución), y el tiempo de ejecución de cada algoritmo. Cada uno de estos programas puede ejecutarse individualmente en la consola así:

En Linux, Mac y similares
```
$ ./my_test_sort_random tamano_arreglo
$ ./my_test_sort_inverse tamano_arreglo
$ ./my_test_sort_sorted tamano_arreglo
```
En Windows
```
> .\my_test_sort_random.exe tamano_arreglo
> .\my_test_sort_inverse.exe tamano_arreglo
> .\my_test_sort_sorted.exe tamano_arreglo
```

Ejemplo:

```
$ ./my_test_sort_random 18000
18000 1 1 1 1348 2 6
```

Para realizar el análisis empírico de la complejidad de los tres algoritmos de ordenamiento en las 3 configuraciones de arreglos, cada programa se ejecuta varias veces con diferentes tamaños de arreglo (entre 1000 y 20000, en este caso) para generar una tabla para cada algoritmo con los respectivos valores de ejecución ("`sort_random_table.txt`", "`sort_inverse_table.txt`" y "`sort_sorted_table.txt`"). Finalmente, es necesario utilizar nuevamente alguna suite de oficina y los datos almacenados en las tablas para graficar el comportamiento de los algoritmos en términos de sus tiempos de ejecución (en nanosegundos, columnas 5, 6 y 7 de cada tabla), generando así tres gráficas, una para cada configuración de los arreglos.

Utilice las imágenes generadas para escribir la tercera parte de su reporte, respondiendo las siguientes preguntas:

1. **(45%)** Analice las gráficas. En otras palabras, escriba varios párrafos donde argumente cuál algoritmo es mejor de acuerdo a las diferentes configuraciones de los datos de entrada, apoyándose únicamente en las gráficas e incluyéndolas (las tres) en su informe final.
2. **(15%)** Investigue la complejidad teórica de los algoritmos de ordenamiento (*bubblesort*, *quicksort* y *heapsort*). Póngala en su reporte, haciendo una referencia correcta a la fuente donde encontró esta información.
3. **(40%)** Relacione el análisis ya realizado de las gráficas con la complejidad teórica de los tres algoritmos.

3 Evaluación

La entrega se hará a través de la correspondiente asignación de BrightSpace, antes de la medianoche del próximo martes 6 de febrero. Se debe entregar un único archivo con el informe (único formato aceptado: `.pdf`), nombrado con los apellidos de los integrantes del grupo. Si la entrega contiene archivos en cualquier otro formato, será descartada y no será evaluada, es decir, la nota definitiva de la entrega será de 0 (cero) sobre 5 (cinco).

La evaluación del taller tendrá la siguiente escala para cada uno de los puntos (Análisis inicial, *Exponential*, *Sort*):

- **Excelente** (**5.0/5.0**): El estudiante presenta análisis y datos organizados (información) que le permiten llegar a una conclusión coherente sobre la comparación de los algoritmos. Además, el estudiante es capaz de explicar la coherencia de los resultados.
- **Bueno** (**4.0/5.0**): El estudiante tiene una idea de cómo comparar los algoritmos e interpretar los resultados, pero falla en el formalismo. Además, no presenta datos organizados o no hizo un análisis de la coherencia de los resultados.
- **Regular** (**3.0/5.0**): El estudiante logra esbozar una idea de comparación de los algoritmos e interpretación de los resultados, pero falla en presentarla de manera clara y contundente. No presenta datos organizados ni su correspondiente análisis.
- **Malo** (**1.5/5.0**): El estudiante no logra entender las instrucciones presentadas, y su análisis es vago o ambiguo.
- **No entregó** (**0.0/5.0**): No entregó el archivo del informe, o el archivo entregado no está en formato PDF.