Práctica 1. "Análisis experimental de la eficiencia de algoritmos de ordenamiento Parte 1"

Angel Zait Hernández López
No.Boleta: 2014080682
Análisis de Algoritmos, Grupo: 3CM4

I. Objetivo

El alumno realizará un análisis a posteriori (experimental) de 2 algoritmos de ordenamiento. Implementará y comparará la eficiencia de estos algoritmos en los casos mejor, peor y promedio.

II. Intrucciones

Para mostrar la eficiencia de diferentes algoritmos que solucionan un mismo problema, se consideró el problema de ordenamiento de una lista de números enteros positivos. Los algoritmos que se implementarán son:

1) Ordenamiento por inseción
2) El método de la burbuja

Considere como entrada, conjuntos de números enteros con $\{1000, 2000, 3000, \ldots, 10,000\}$ elementos. Deberá realizar la ejecución de cada algoritmo para el mejor caso, peor caso y caso promedio. Deberá reportar el tiempo de ejecución (en milisegundos) de cada instancia como se pide en la Tabla 1. Una vez que se tengan los datos del tiempo de ejecución grafique (de preferencia en python o gnuplot) estos resultados (ver ejemplo en Figura 1).

Deberá enviar a la plataforma classroom una carpeta comprimida con la implementación de los códigos python y el reporte en formato latex y pdf. La carpeta debe tener el nombre del alumno (comenzando por apellido paterno). La fecha límite de entrega es Miércoles 29 de febrero de 2019 a las 10:00 pm. Por cada día de retraso se penalizará al alumno con 20% de la calificación obtenida.

El reporte en latex debe considerar las siguientes secciones:

A. Algoritmos

Algorithm 1: Insertion Sort Algorithm

```
Data: A : list of sortable items
1  begin
2      InsertionSort(A)
3      for i ← 2 to n do
4          j ← i − 1;
5          while j ≥ 1 and A[j] > A[j + 1] do
6              swap(A[j], A[j+1]);
7              j ← j − 1;
8      return A;
9  end
```

Algorithm 2: Bubble Sort Algorithm

```
Data: A : list of sortable items
1  begin
2      n ← length(A);
3      repeat
4          swapped ← false;
5          for i ← 1 to n do
6              if A[i − 1] > A[i] then
7                  swap(A[i-1], A[i]);
8                  swapped ← true;
9          n ← n − 1;
10     until not swapped ;
11     return A;
12 end
```

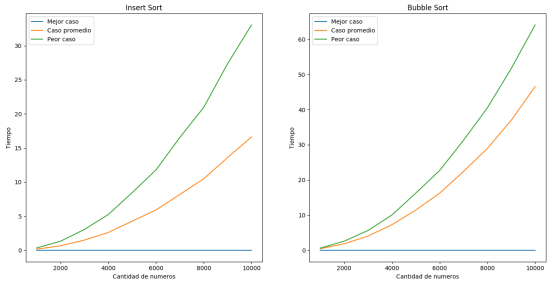

Fig. 1. Comparación del comportamiento de los algoritmos

La figura 1 muestra el comportamiento de las funciones.

n	Mejor Caso	
	insertion	bubble
1000	0.00037	0.000551
2000	0.000699	0.001098
3000	0.001015	0.001619
4000	0.001291	0.002099
5000	0.001704	0.002677
6000	0.001952	0.00316
7000	0.002356	0.003766
8000	0.002618	0.004207
9000	0.003046	0.004836
10000	0.003402	0.005479

TABLE I
Resultados de los tiempos en segundos para cada algoritmo en el mejor caso.

n	Peor Caso	
	insertion	bubble
1000	0.346261	0.618988
2000	1.332823	2.593066
3000	3.057279	5.641208
4000	5.243366	10.033722
5000	8.442194	16.257205
6000	11.787576	22.686411
7000	16.552769	31.290495
8000	20.963428	40.502695
9000	27.355983	51.701332
10000	33.041604	64.072419

TABLE II
RESULTADOS DE LOS TIEMPOS EN SEGUNDOS PARA CADA ALGORITMO EN EL PEOR CASO.

n	Caso promedio	
	insertion	bubble
1000	0.174848	0.463978
2000	0.655768	1.855052
3000	1.500251	4.066068
4000	2.614248	7.268449
5000	4.275675	11.457816
6000	5.913062	16.242843
7000	8.193118	22.426691
8000	10.490098	28.964692
9000	13.59884	36.980311
10000	16.620958	46.539923

TABLE III
RESULTADOS DE LOS TIEMPOS EN SEGUNDOS PARA CADA ALGORITMO EN EL CASO PROMEDIO.

III. CONCLUCIONES

Con la ayuda experimenta pudimos notar que el tiempo de ejecución de los algoritmos en los diferentes casos su diferencia es demasiada grande. Un claro ejemplo es cuando tenemos los datos en el peor caso, ya que a pesar de tener la misma cantidad de datos en el mismo orden, el método de Insert Sort tiene mayor velocidad, y más si se trata de cantidades demasiadas grandes. De igual forma, podemos ver que sin importar el tamaño de nuestro arreglo, este va a seguir creciendo, excepto si es un arreglo ya ordenado, ya que se mantiene el tiempo de ejecución casi constate. Por último, podemos notar que mientras más datos haya, mas tardado es el proceso de ordenamiento por el algoritmo de Bubble Sort, es decir, que para una gran cantidad de datos, es mejor usar el método de Insert Sort.

REFERENCES

[1] P. Moscato, *On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms,* Technical Report C3P Report 826, 1989.

[2] N. Krasnogor and J. Smith, *A Memetic Algorithm With Self-Adaptive Local Search: TSP as a case study,* Genetic Evolutionary Computation Conference pp. 987-994, 2000.