Análisis de Algoritmos 2018/2019

Práctica 1

Alba Ramos, Javier Lozano, 1212.

Código	Gráficas	Memoria	Total

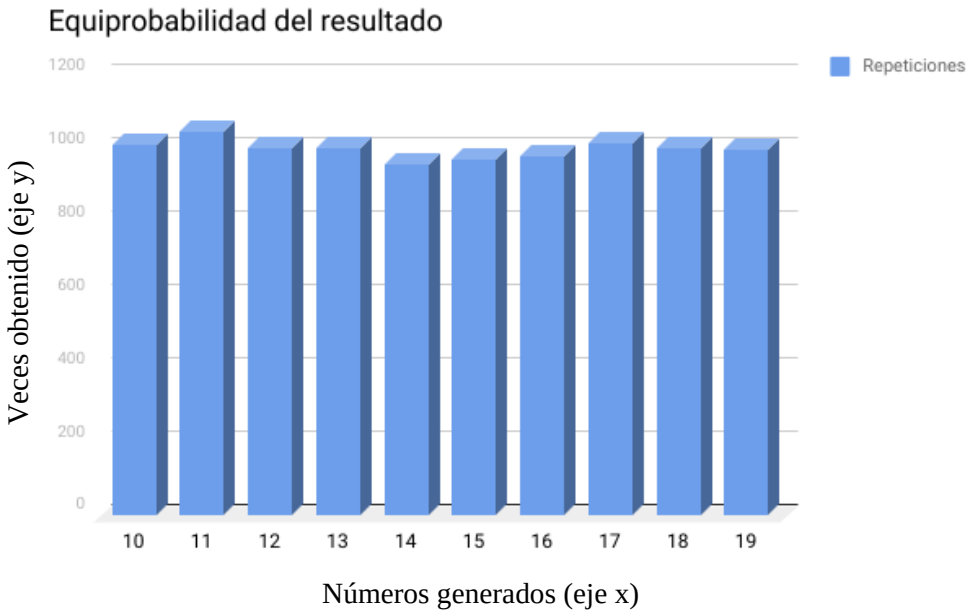
5. Resultados, Gráficas

5.1 Apartado 1

```
Ejecutando ejercicio1
   1011 10
   1049 11
   1004 12
   1005 13
    959 14
    972 15
    980 16
   1017 17
   1005 18
    998 19
      1 Grupo: 1212
      1 Practica numero 1, apartado 1
      1 Realizada por: Alba Ramos y Javier Lozano
```

Se han obtenido números aleatorios equiprobables dentro de un intervalo dado.

En la gráfica se muestra el número de veces que se repiten cada uno de los números generados entre 10 y 19. Como podemos observar, todos los números tienen un promedio de repetición de 1000 veces.

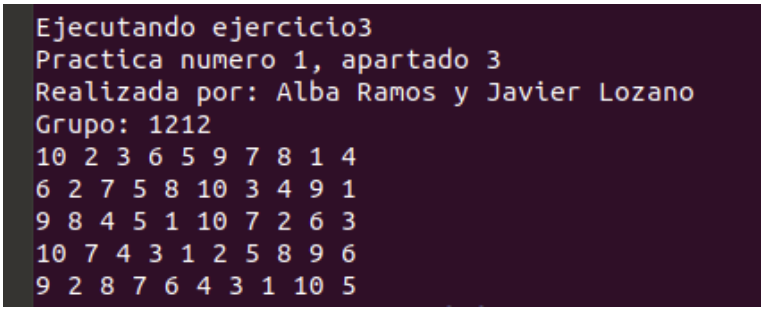


5.2 Apartado 2

```
Practica numero 1, apartado 2
Realizada por: Alba Ramos y JAvier Lozano
Grupo: 1212
8 9 3 1 7 10 5 4 6 2
10 4 5 2 8 6 9 3 1 7
8 6 7 3 4 1 5 9 2 10
2 10 8 5 9 6 7 1 4 3
1 3 6 2 5 4 9 10 7 8
```

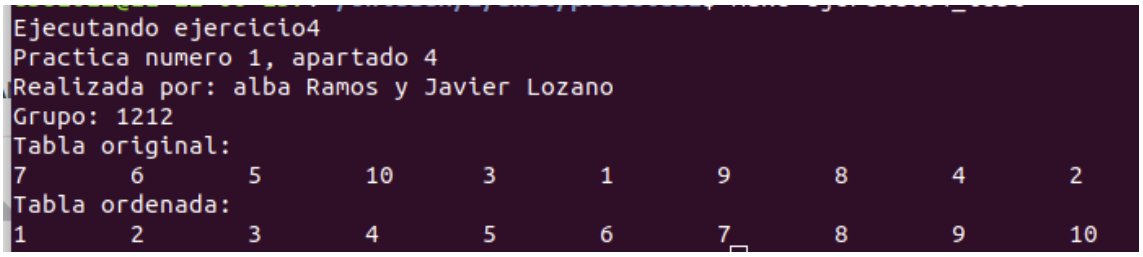
Se han obtenido 5 tablas de 10 elementos que van entre 1 y 10. En la imagen, cada fila es una tabla.

5.3 Apartado 3



Se ha obtenido una tabla que contiene 5 tablas de 10 elementos que van desde 1 hasta 10.

5.4 Apartado 4

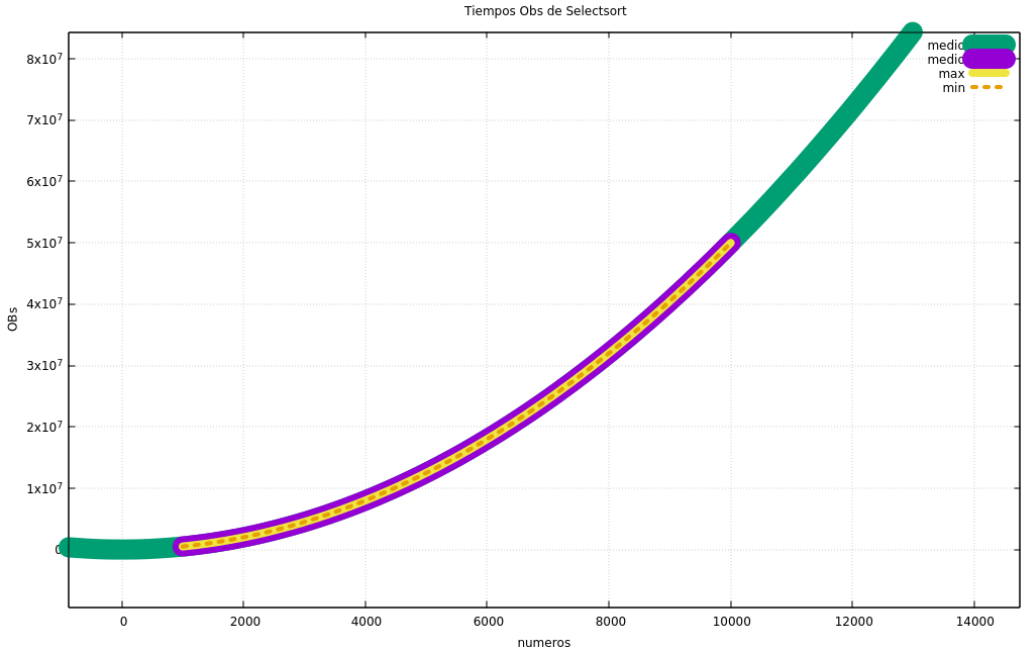


Se ha ordenado una tabla de 10 elementos del 1 al 10 utilizando el método de ordenación SelectSort.

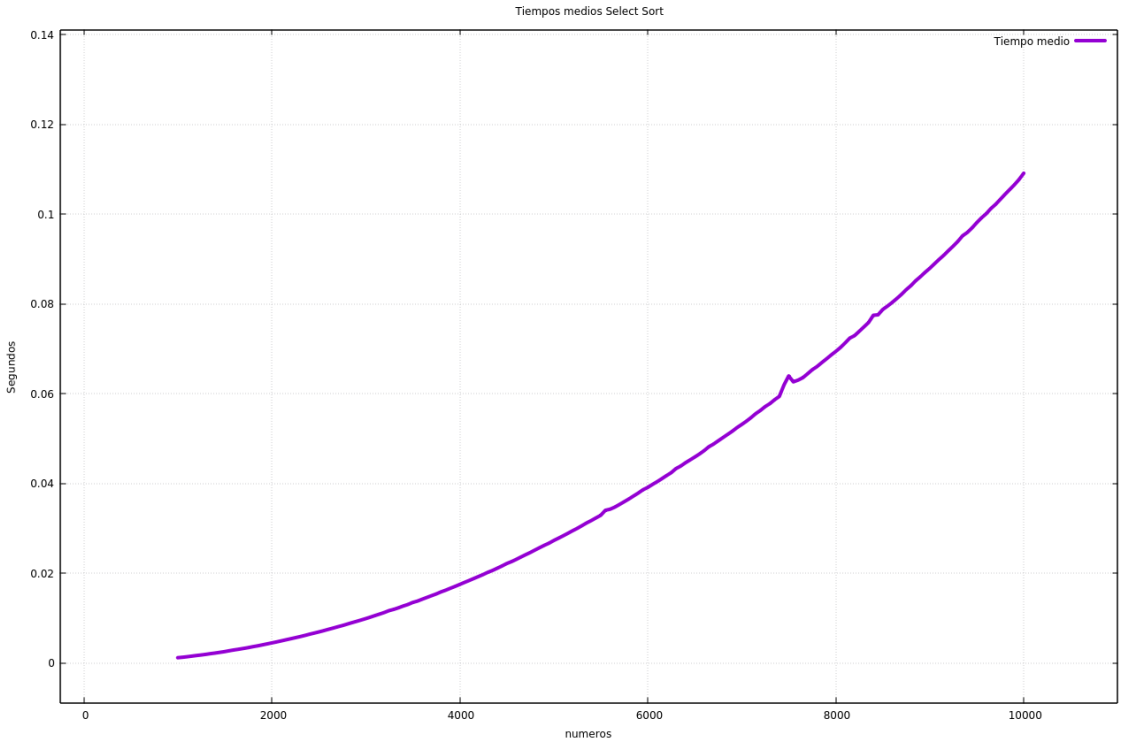
5.5 Apartado 5

En este apartado hemos medido los tiempos de ejecución del algoritmo SelectSort para tamaños de 1000 a 10.000 en incrementos de 50. Hemos generado un fichero ejercicio5.log en el que se recoge el tiempo medio de reloj, el tiempo medio en Obs, el tiempo máximo en Obs y el tiempo mínimo en Obs para cada N.

En la siguiente gráfica se representan los tiempos mejor (color naranja), peor (color amarillo) y medio (color morado) en OBs para SelectSort. Como podemos observar, se trata de curvas que se ajustan perfectamente al valor teórico (color verde) esperado: $N^2/2 + O(N)$. Los tiempos mejor, medio y peor son los mismos para este algoritmo, porque al no tener flags, todos los casos tardan lo mismo.



La siguiente gráfica muestra el tiempo medio de reloj en segundos, para SelectSort. Como podemos comprobar, además de coincidir con el tiempo en OBs, también se ajusta al valor teórico esperado.



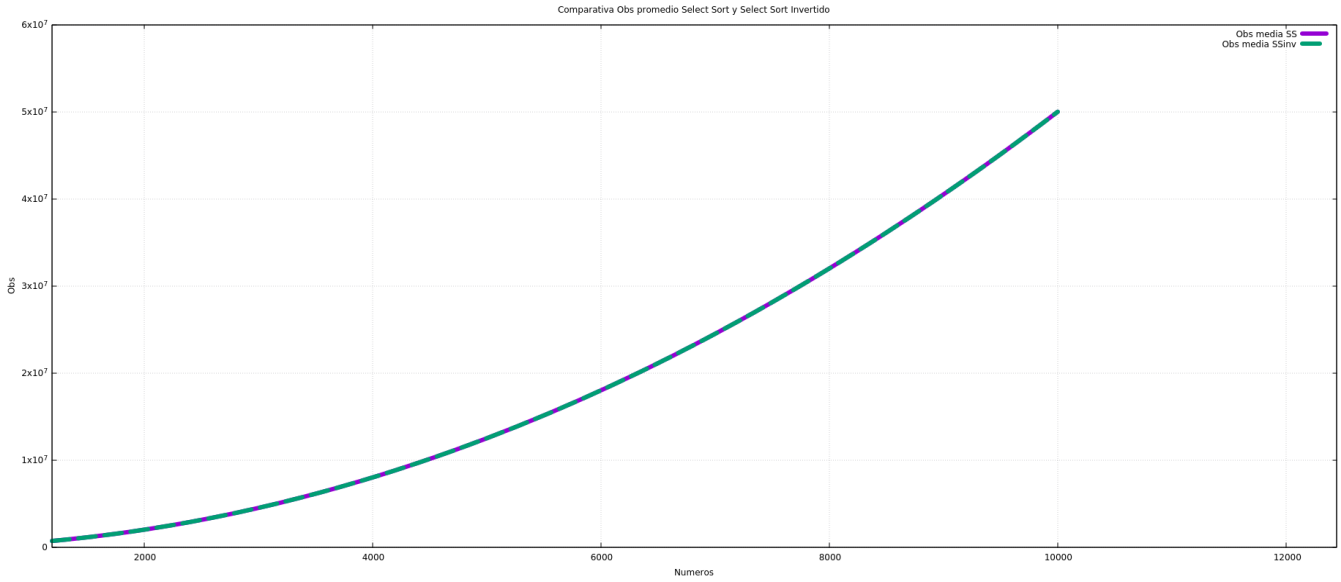
5.6 Apartado

```
Tabla original:
10      3       7       5       8       6       4       1       2       9
Tabla ordenada:
1       2       3       4       5       6       7       8       9       10
Tabla ordenada inv:
10      9       8       7       6       5       4       3       2       1
```

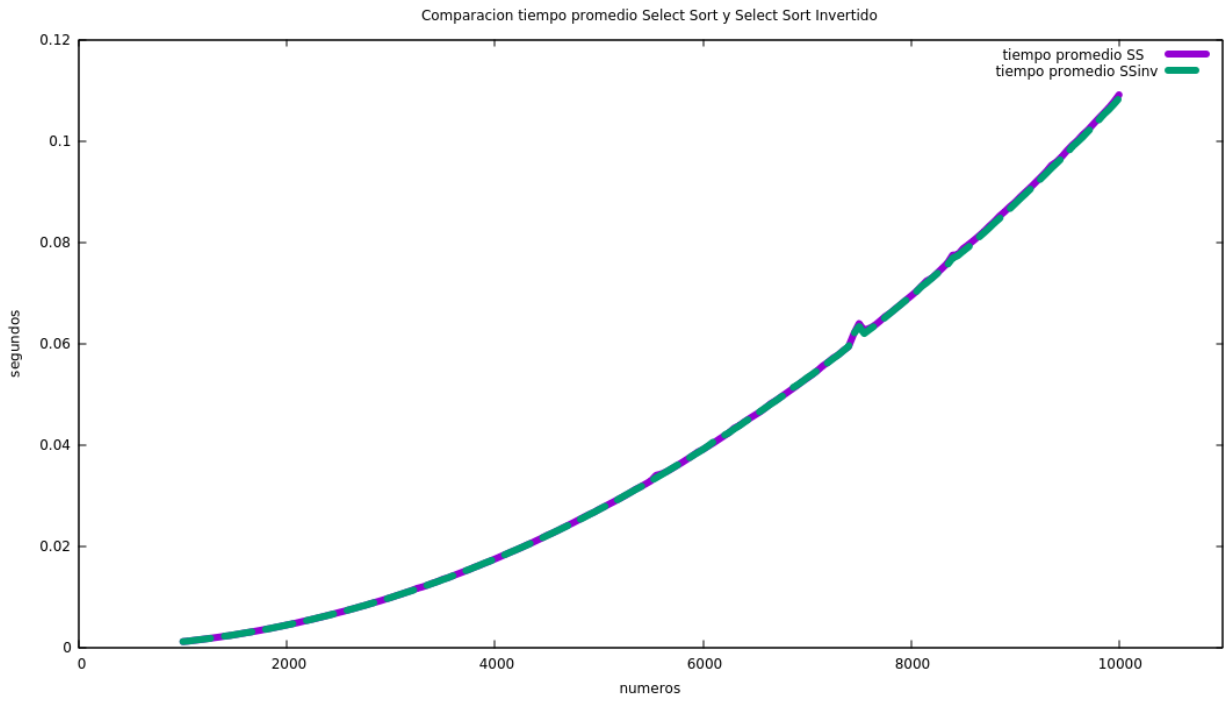
En este apartado se ha ordenado una tabla de 10 elementos del 1 al 10 de mayor a menor. Para ello hemos usado el algoritmo SelectSortInv, que se diferencia del que conocemos en que la comparación de claves comprueba si T[j] > T[min], en vez de menor.

A continuación, hemos medido los tiempos de ejecución del nuevo algoritmo para tamaños de 1000 a 10.000 en incrementos de 50. Hemos generado el fichero ejercicio6.log en el que se recoge el tiempo medio de reloj, el tiempo medio en Obs, el tiempo máximo en Obs y el tiempo mínimo en Obs para cada N.

En la siguiente gráfica se comparan el tiempo medio en OBs para ambos algoritmos (SelectSort en morado y SelectSortInv en verde). Podemos observar que las gráficas coinciden, según lo que hemos explicado previamente.



En la siguiente gráfica se muestra la comparación entre los tiempos medios de reloj en segundos para ambos algoritmos. Como podemos observar, las gráficas son parecidas. Se pueden apreciar picos entre ambas curvas a partir de iteraciones de más de 7000 números. No estamos seguros de si esto se debe al tiempo de ejecución del ordenador en ese momento, o a diferencias de los algoritmos. Creemos que se trata de lo primero, ya que ambos algoritmos utilizan la misma OB salvo un cambio de signo.



5. Respuesta a las preguntas teóricas.

5.1 Pregunta 1

Ya que la función rand() nos proporciona números aleatorios dentro del rango [0, RAND_MAX], teníamos que modificarla para ajustarla al intervalo deseado. Para ello, analizamos los pasos a seguir para pasar del intervalo original al deseado. Llegamos a la conclusión de que teníamos que dividir el número de elementos en el intervalo original entre el número de elementos del intervalo deseado. De esta forma, generamos una constante por la que hay que dividir el número aleatorio generado por rand(). A partir de esta operación, el nuevo número se encuentra entre 0 y el total de elementos del intervalo deseado. Para solucionar esto, le sumamos el límite inferior del intervalo deseado, obteniendo así que el número aleatorio proporcionado por rand() se encuentre dentro de nuestro intervalo.

Otra forma de generar números aleatorios dentro de un intervalo podría ser usando rand()% (lim_sup + 1 – lim_inf) + lim_inf. El problema de usar el módulo es que los resultados ya no son realmente aleatorios, y además si generamos números cercanos a RAND_MAX tendríamos el mismo problema.

5.2 Pregunta 2

El algoritmo recorre cada elemento de la tabla a ordenar. El bucle externo recorre los elementos desde el índice 0 hasta el N-1. Antes de entrar al segundo bucle,

se asume que el índice del elmento mínimo es i.

El bucle interno recorre el resto de elementos, es decir, desde i+1 hasta N, y busca el menor de todos. Si el elemento en la posición j es mayor o igual que el que había en la posición min, entonces no realiza nada. Si es menor, min tomará el valor de j.

Al salir del bucle interno, tendremos en min el índice del elemento más pequeño de la tabla que va desde i+1 hasta N. Ahora podemos colocar ese elemento en la primera posición de nuestra tabla, es decir, la posición i, y repetir el proceso para el resto de elementos. Este análisis demuestra la corrección del algoritmo de ordenación SelectSort.

5.3 Pregunta 3

SelectSort ordena una tabla de menor a mayor. Para ello, se basa en la idea de que hay dos subtablas: una, la subtabla ordenada. Otra, la subtabla desordenada. La subtabla desordenada corresponde a los elementos i+1,…, N. Encuentra el menor entre ellos y lo agrega a la subtabla ordenada, es decir, la tabla 0,…,i, donde i llega hasta el penúltimo elemento. Si el bucle exterior llegase hasta el último elemento, no tendríamos subtabla desordenada sobre la que buscar el mínimo, por lo tanto el criterio de ordenación no funcionaría.

5.4 Pregunta 4

La comparación de claves: if(tabla[j] < tabla[min])

5.5 Pregunta 5

El caso peor del algoritmo es $n^2/2 + O(n)$. El caso mejor del algoritmo es también $n^2/2 + O(n)$. Esto se debe a que el algoritmo no tiene forma de comprobar que una entrada está ordenada, por lo tanto, el tiempo de ejecución no depende de la entrada.

5.5 Pregunta 6

Los tiempos en OBs coinciden para ambos algoritmos. Esto se debe a que, aunque uno ordene de mayor a menor y otro de menor a mayor, la OB sigue siendo la misma, salvo por un signo (en un caso ‘<’ y en otro ’>’). Por lo tanto, ambos algoritmos tardan lo mismo. En cambio, observamos diferencias en los tiempos en segundos. Creemos que se debe al estado del ordenador en el momento en el que se realizó la prueba.