Análisis de Algoritmos 2021/2022

Práctica 2

Hugo Torres y Luis Rodríguez, Grupo 1261.

Código	Gráficas	Memoria	Total

1. Introducción.

En esta práctica trataremos los fundamentos de otros algoritmos de ordenación, en específico los algoritmos de ordenación recursivos. Como el Merge Sort o el Quick Sort en lenguaje C.

2. Objetivos

Aquí indicáis el trabajo que vais a realizar en cada apartado.

2.1 Algoritmo Merge Sort apartado 1

Implementamos en Sorting.c el primer algoritmo de esta práctica. El Merge Sort.

Se implementan dos funciones nuevas que se necesitan para que funcione el Merge Sort; estas son: int mergesort(int* tabla, int ip, int iu) y la función de combinación int merge(int* tabla, int ip, int iu, int imedio).

También modificamos el ejercicio 4 para comprobar el algoritmo nuevo creado.

2.2 Algoritmo Merge Sort Apartado 2

Modificamos el Ejercicio 5 de la práctica anterior pero esta vez usando el algoritmo MergeSort para obtener una tabla de tiempos correspondiente con tiempos medios, mejores y peores dentro de este algoritmo. Los representamos y mostramos a continuación lo que nos da en el ejercicio de las tablas. Dichos valores serán también comparados con los valores teóricos que sean necesarios para ese algoritmo.

2.3 Algoritmo Quick Sort apartado 1

En este apartado haremos el algoritmo de ordenación Quick Sort. Los valores que devuelve son o ERR o el número de veces que se ejecuta la OB del algoritmo. Contiene otras rutinas de apoyo para el correcto funcionamiento como son las funciones de *partition* y *median*.

Luego se ha de modificar el ejercicio 4 para comprobar el correcto funcionamiento de este algoritmo.

2.4 Algoritmo Quick Sort apartado 2

En este ejercicio el objetivo es la modificación del ejercicio 5. para obtener la tabla de tiempo promedio de ejecución, máximo, mínimo y promedio de operaciones básicas del algoritmo QuickSort en función del tamaño de la permutación.

2.5 Algoritmo Quick Sort apartado 3

En este ejercicio hay que modificar la recursión de cola de este algoritmo para ver si se elimina la sobrecarga creada por el alto número de veces que se llama a esa función. Creando la rutina de QuickSort_src .

3. Herramientas y metodología

Aquí ponéis qué entorno de desarrollo (Windows, Linux, MacOS) y herramientas habéis utilizado (Netbeans, Eclipse, gcc, Valgrind, Gnuplot, Sort, uniq, etc) y qué metodologías de desarrollo y soluciones al problema planteado habéis empleado en cada apartado. Así como las pruebas que habéis realizado a los programas desarrollados.

3.1 Apartado 1

Hugo - Linux, Visual Studio Code, Gcc, valgrind, make, Sort, Gnuplot, uniq.
Luis - Unix, Visual Studio Code, Gcc, make, Gnuplot, uniq, Sort.

En este primer ejercicio no nos encontramos con ningún problema que no supiesemos resolver. La parte más compleja fue entender el algoritmo a la perfección para poder pasarlo a código bien.

3.2 Apartado 2

Hugo - Linux, Visual Studio Code, Gcc, valgrind, make, Sort, Gnuplot, uniq.
Luis - Unix, Visual Studio Code, Gcc, make, Gnuplot, uniq, Sort

No fue complejo modificar el ejercicio 5. Las tablas que salieron están adjuntas en esta memoria en el apartado 5 de resultados y gráficas.

3.3 Apartado 3

Hugo - Linux, Visual Studio Code, Gcc, valgrind, make, Sort, Gnuplot, uniq.
Luis - Unix, Visual Studio Code, Gcc, make, Gnuplot, uniq, Sort

En este ejercicio lo que mas nos costo fue entender el hecho del pivote y la forma de devolverlo y encontrarlo en el algoritmo.

3.4 Apartado 4

Hugo - Linux, Visual Studio Code, Gcc, valgrind, make, Sort, Gnuplot, uniq.
Luis - Unix, Visual Studio Code, Gcc, make, Gnuplot, uniq, Sort

No fue complicado la modificación del ejercicio 5 para obtener los tiempos.

3.5 Apartado 5

Hugo - Linux, Visual Studio Code, Gcc, valgrind, make, Sort, Gnuplot, uniq.
Luis - Unix, Visual Studio Code, Gcc, make, Gnuplot, uniq, Sort

En este ejercicio tuvimos la complicación de entender como quitar la recursión de este algoritmo. Una vez entendido eso conseguimos crear el nuevo algoritmo y comparar los tiempos con el que si tiene recursión de cola.

4. Código fuente

Aquí ponéis el código fuente **exclusivamente de las rutinas que habéis desarrollado vosotros** en cada apartado.

4.1 Apartado 1

```
int MergeSort(int* tabla, int ip, int iu){
int m,ob;

 assert(ip>=0||iu>=0||ip<iu);

 if(ip==iu) return 0;

 m=(ip+iu)/2;
 ob=0;
 ob+=MergeSort(tabla,ip,m);

 ob+=MergeSort(tabla,m+1,iu);

 ob+= merge(tabla,ip,iu,m);

 return ob;
}

void copy(int* aux, int* t, int p, int u){
 int i=p;
 aux-=p;
 while(i<=u){
   t[i]=aux[i];
   i++;
 }
}

int merge(int* tabla, int ip, int iu, int imedio){
 int* tablaaux = NULL;
 int i, j, k, ob=0;

 assert(ip>=0||iu>=0||ip<iu||imedio>=0);

 i = ip;
 j = imedio+1;

 tablaaux=(int*)malloc((iu-ip+1)*sizeof(int));
```

```
    k = 0;

     while(i<=imedio && j<=iu){
      if(tabla[i]<tabla[j]){
        tablaaux[k]=tabla[i];
        i++;
        ob++;
      }
      else{
       tablaaux[k]=tabla[j];
       j++;
       ob++;
      }
      k++;
    }
    if(i>imedio){
      while(j<=iu){
        tablaaux[k]=tabla[j];
        j++;
        k++;
      }
    }
    else if(j>iu){
      while(i<=imedio){
        tablaaux[k]=tabla[i];
        i++;
        k++;
      }
    }
    copy(tablaaux, tabla, ip, iu);
    free(tablaaux);
    return ob;
   }
```

4.3 Apartado 3

```
int quicksort(int* tabla, int ip, int iu){
  int M, pos, ob=0;
  assert(ip>=0||iu>=0);
  if(ip == iu) return OK;
  else{
   M = partition(tabla, ip, iu, &pos);
   ob+=M;
   if(ip<pos-1){
     M = quicksort(tabla, ip, pos-1);
     ob+=M;
   }
   if(pos+1<iu){
     M = quicksort(tabla, pos+1, iu);
     ob+=M;
   }
 }
 return ob;
}

int partition(int* tabla, int ip, int iu, int *pos){
 int M, k, i, ob=0;

 assert(ip>=0||iu>=0);

 M= median(tabla, ip, iu, pos);

 ob+=M;

 k = tabla[*pos];

 swap(&tabla[ip], &tabla[*pos]);
 *pos = ip;
 for(i=ip+1; i<=iu; i++){
   ob++;
   if(tabla[i]<k){
     (*pos)++;
     swap(&tabla[i],  &tabla[*pos]);
   }
 }
 swap(&tabla[ip], &tabla[*pos]);
 return ob;
}
```

```c
int median(int *tabla, int ip, int iu,int *pos){

 assert(ip>=0||iu>=0);

 *pos=(ip+iu)/2;
 return 0;
}
```

4.5 Apartado 5

```c
int quicksort_src(int* tabla, int ip, int iu){
 int* p=NULL;
 int pos, a, ob;
  a = 0;
 pos = 0;
 ob = 0;
 assert(ip>=0||iu>=0||ip<iu);
 p = (int*)malloc((iu+2)*sizeof(int));
 if(p==NULL) return ERR;
 p[a]=ip;
 a++;
 p[a]=iu;
 while(a>=0){
   iu=p[a];
   a--;
   ip=p[a];
   a--;
   ob+=partition(tabla,ip,iu,&pos);
   if(ip<pos-1){
     a++;
     p[a]=ip;
     a++;
     p[a]=pos-1;
   }
   if(pos+1<iu){
     a++;
     p[a]=pos+1;
     a++;
     p[a]=iu;
   }
 }
 free(p);
 return ob;
```

5. Resultados, Gráficas

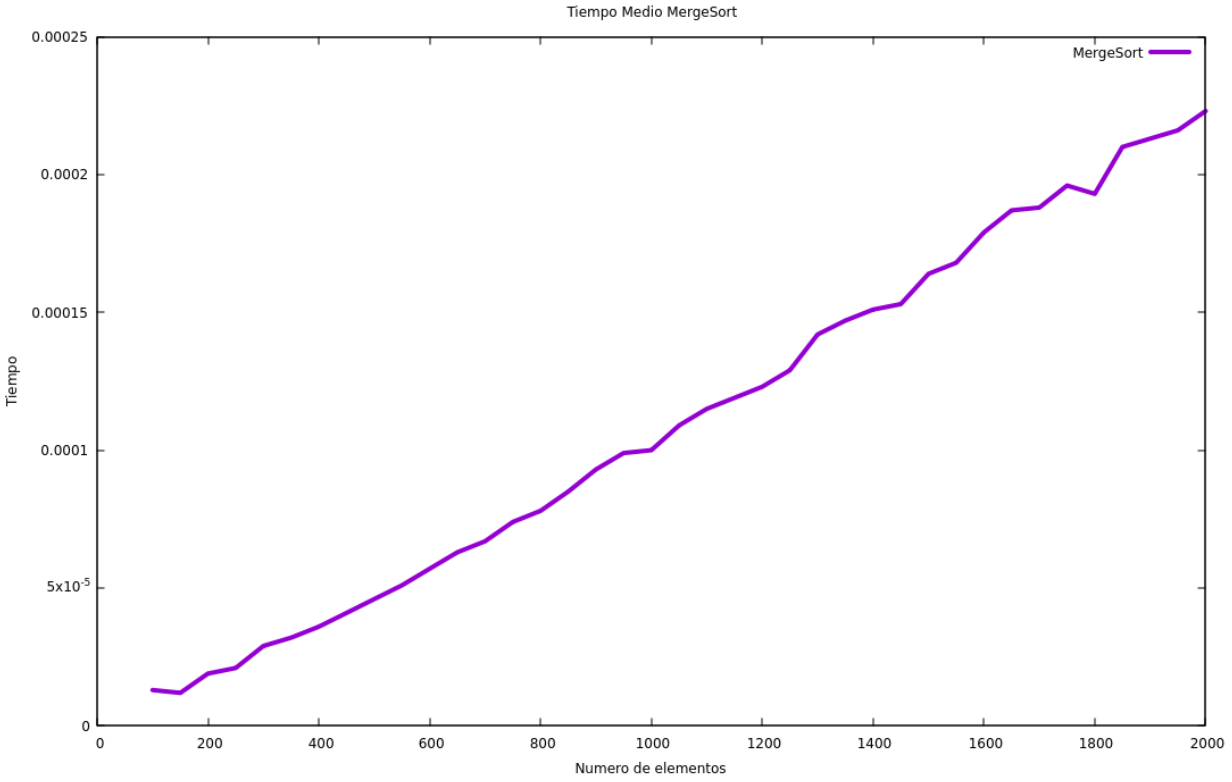
Aquí ponéis los resultados obtenidos en cada apartado, incluyendo las posibles gráficas.

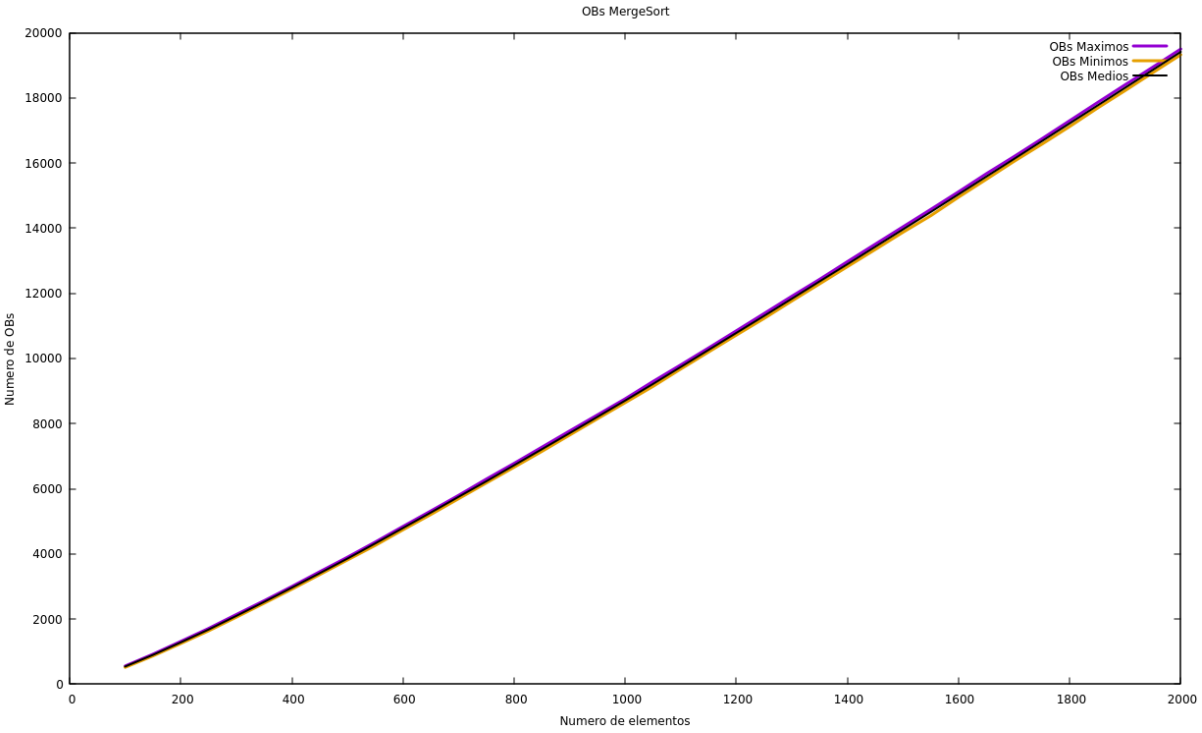
5.1 Apartado 1

```
exercise4_test:
	@echo Running exercise4
	@./exercise4 -size 15
```

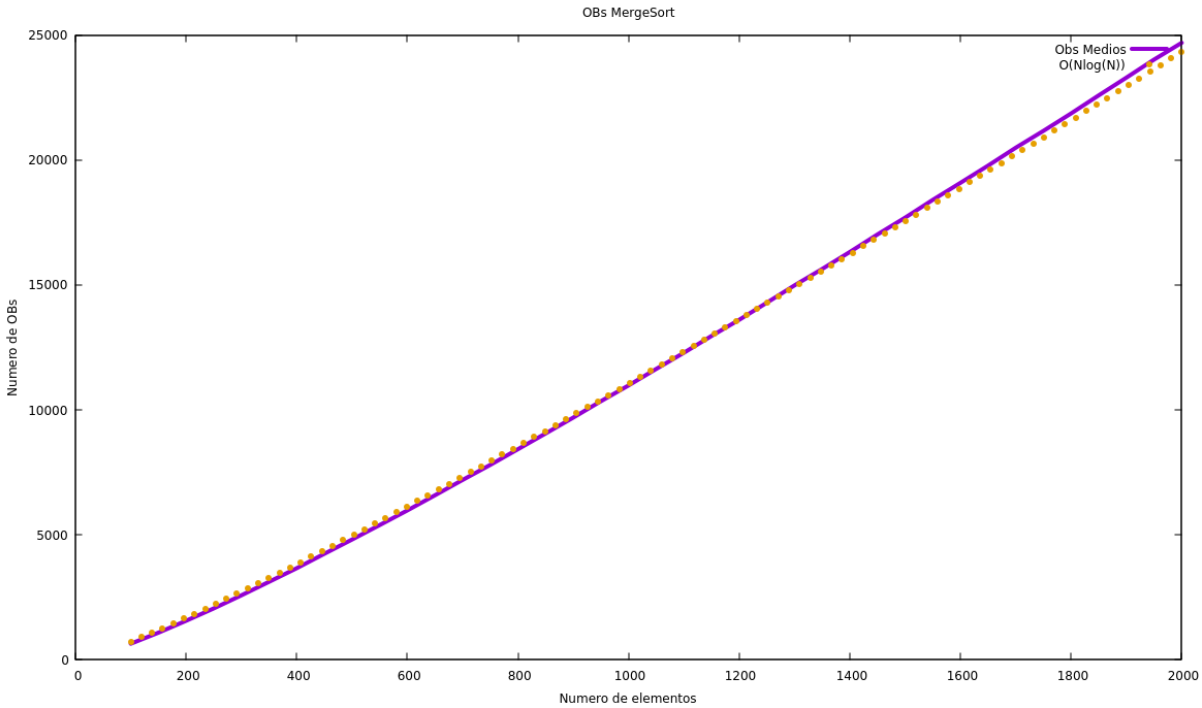
```
Running exercise4
Practice number 2, section 4
Done by: Hugo Torres y Luis Rodríguez
Group: 1261
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
```

5.2 Apartado 2





Gráfica con el tiempo medio de reloj para MergeSort. Se puede ver que el tiempo medio de Merge Sort es casi igual a el peor y mejor. Aquí se ve bien representado el hecho de que el algoritmo Merge Sort es mucho más eficiente que el Quick Sort.

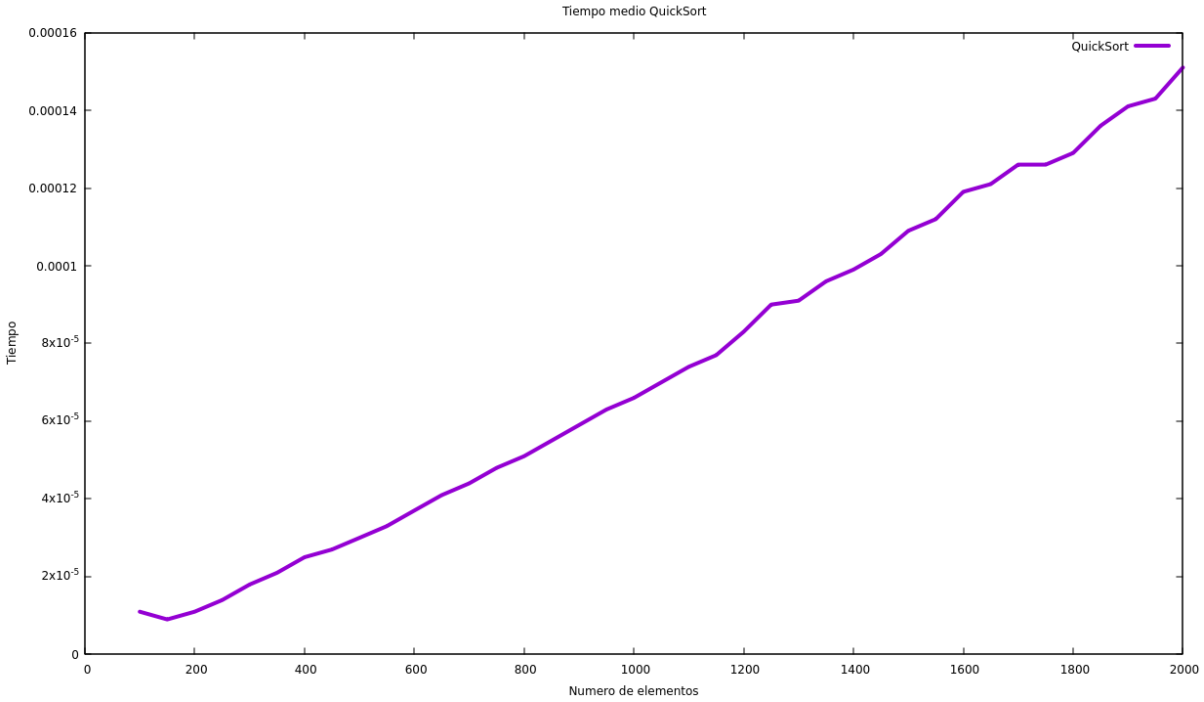


5.3 Apartado 3

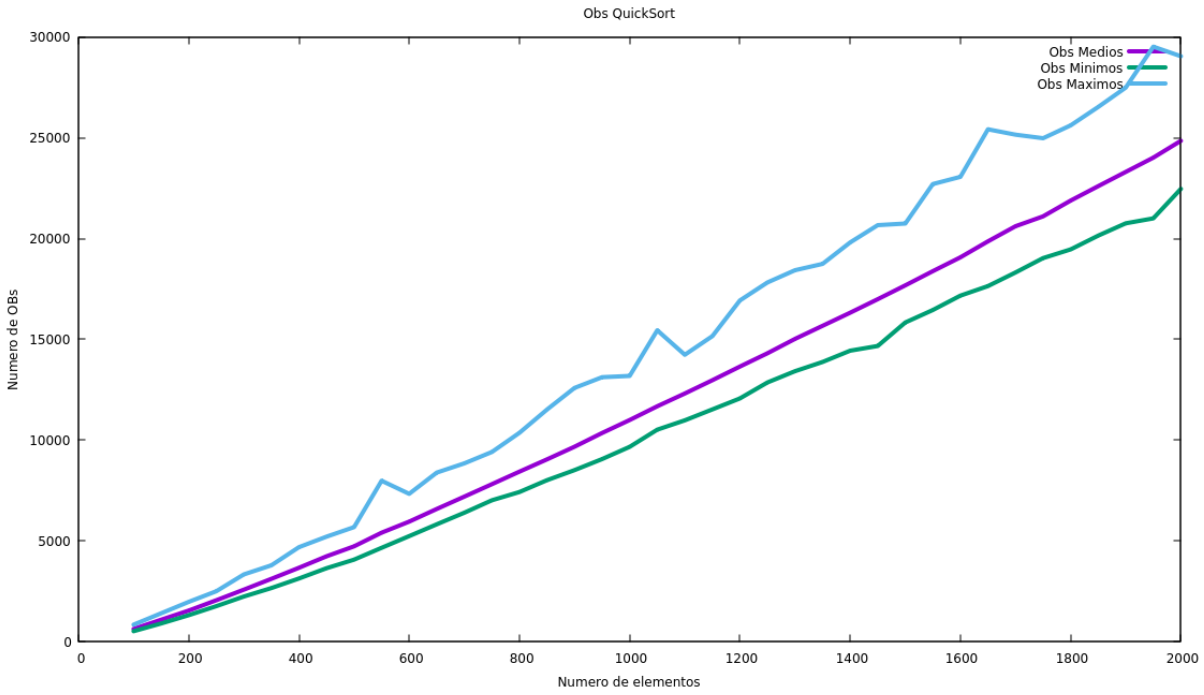
```
exercise4_test:
	@echo Running exercise4
	@./exercise4 -size 10
```

```
Running exercise4
Practice number 2, section 4
Done by: Hugo Torres y Luis Rodríguez
Group: 1261
1       2       3       4       5       6       7       8       9       10
```

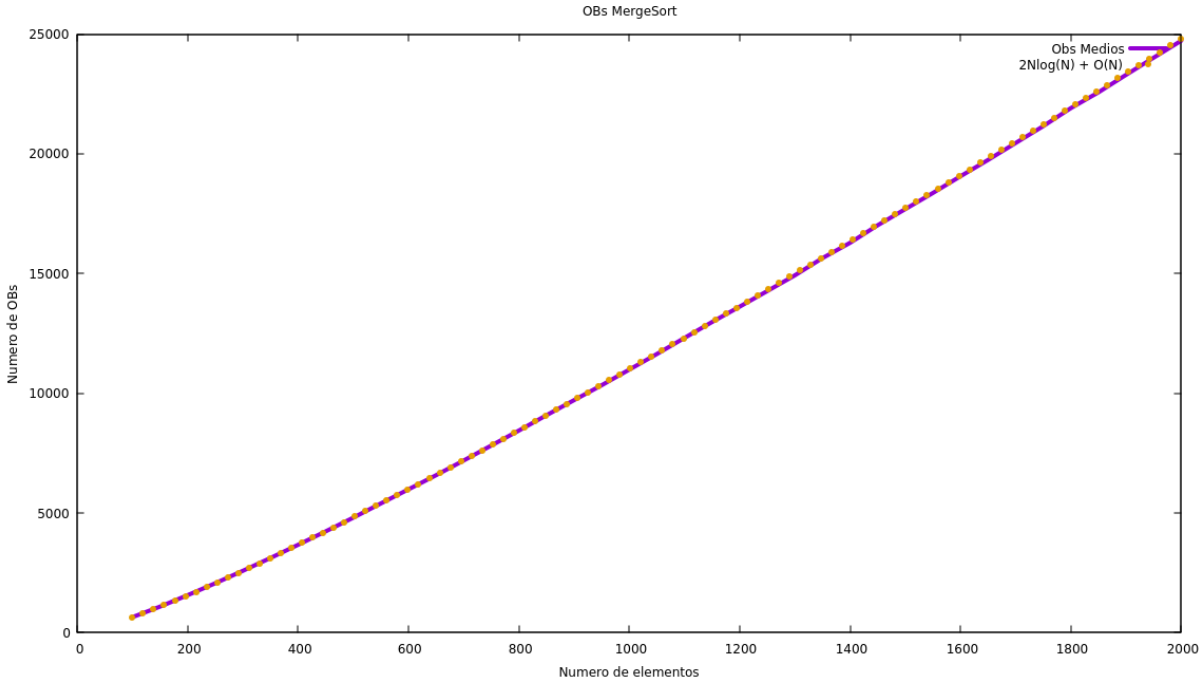
5.4 Apartado 4



Gráfica con el tiempo medio de reloj para QuickSort.

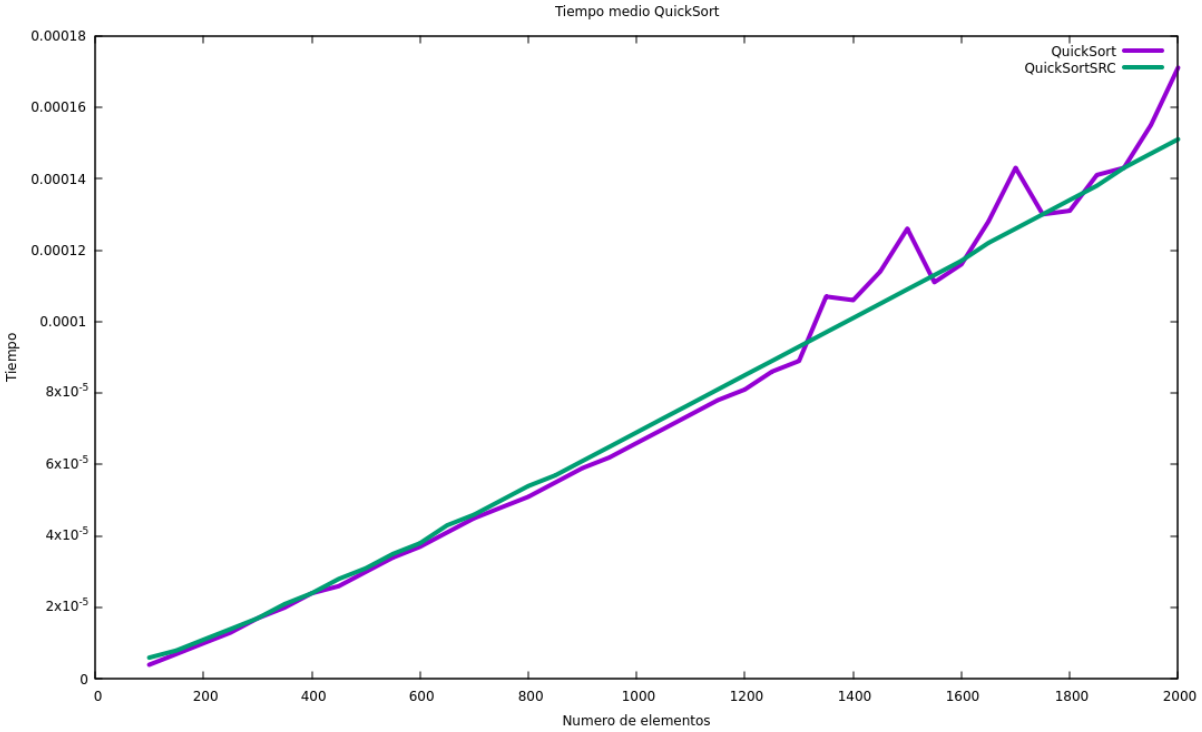


Gráfica comparando las OBs ,mejor peor y medio en OBs para QuickSort.



Gráfica comparando las OBs medias con la función teórica del algoritmo.

5.5 Apartado 5



Gráfica con el tiempo medio de reloj comparando las versiones de Quicksort con y sin recursión de cola.

5. Respuesta a las preguntas teóricas.

Aquí respondéis a las preguntas teóricas que se os han planteado en la práctica.

5.1 Pregunta 1: Compara el rendimiento empírico de los algoritmos con el caso medio teórico en cada caso. Si las trazas de las gráficas del rendimiento son muy picudas razonad por qué ocurre esto.

El rendimiento que hemos tenido nosotros en el caso medio de Merge Sort es de 1.6*NLog(N) el rendimiento es equivalente a el de el caso medio teórico ya que es O(NLog(N)). Como hemos visto en clase, estos dos son equivalentes por lo que consideramos que está bien hecho el ejercicio.

Nuestro rendimiento en el caso de QuickSort es de 2NLog(N) + (-2.8)N. Que este también es equivalente a el caso teorico de el caso medio en quicksort dado en clase el cual es: 2NLog(N) + O(N). Estos dos son equivalentes.

5.2 Razonad el resultado obtenido al comparar las versiones de quicksort con quicksort src tanto si se obtienen diferencias apreciables como si no.

100	0.000004	646.953333	519	946
150	0.000007	1087.323333	916	1645
200	0.000010	1564.585000	1321	2294
250	0.000013	2065.754333	1707	2830
300	0.000017	2583.873000	2195	3649
350	0.000020	3116.658333	2633	4173
400	0.000024	3668.477333	3161	4868
450	0.000026	4231.105667	3624	5686
500	0.000030	4804.503667	4132	6432
550	0.000034	5399.419333	4649	7245
600	0.000037	5986.330667	5212	8066
650	0.000041	6577.127333	5688	9042
700	0.000045	7188.231333	6256	10330
750	0.000048	7815.436333	6742	10132
800	0.000051	8442.300000	7326	10768
850	0.000055	9080.693667	7852	11929
900	0.000059	9710.878667	8448	13338
950	0.000062	10332.862000	8906	12900
1000	0.000066	10983.886000	9603	14132
1050	0.000070	11639.409333	10224	14816
1100	0.000074	12310.326333	10774	16734
1150	0.000078	12973.924667	11280	16408
1200	0.000081	13612.256000	11856	17079
1250	0.000086	14267.833000	12247	19234
1300	0.000089	14924.672667	13237	18933
1350	0.000107	15640.574333	13515	19759
1400	0.000106	16281.548667	14112	21693
1450	0.000114	17014.195000	14838	23355
1500	0.000126	17689.343333	15572	23749
1550	0.000111	18363.129333	16227	23819
1600	0.000116	19058.060000	16761	25797
1650	0.000128	19745.298000	17531	25069
1700	0.000143	20447.299000	17961	26855
1750	0.000130	21153.359333	18597	26424
1800	0.000131	21910.408333	19214	31351
1850	0.000141	22564.063333	19781	27674
1900	0.000143	23299.817333	20524	29206
1950	0.000155	24011.559333	21371	29863
2000	0.000171	24721.866333	21964	31685

QUICKSORT

100	0.000006	647.740000	523	945
150	0.000008	1088.236667	896	1516
200	0.000011	1564.375000	1304	2178
250	0.000014	2066.043000	1734	2991
300	0.000017	2582.455000	2152	3874
350	0.000021	3115.773000	2660	4157
400	0.000024	3664.987000	3113	5225
450	0.000028	4242.273667	3597	5660
500	0.000031	4806.158000	4092	6565
550	0.000035	5376.993000	4593	7350
600	0.000038	5996.687333	5168	7942
650	0.000043	6585.593333	5682	8519
700	0.000046	7189.741667	6194	9416
750	0.000050	7807.070667	6821	10079
800	0.000054	8430.302000	7335	10853
850	0.000057	9043.969333	7847	11622
900	0.000061	9711.092667	8483	12675
950	0.000065	10325.881000	9073	13711
1000	0.000069	10981.843000	9653	15419
1050	0.000073	11626.020667	10074	14796
1100	0.000077	12306.272000	10576	15503
1150	0.000081	12947.963667	11366	17332
1200	0.000085	13620.829333	11850	17251
1250	0.000089	14258.106667	12741	18034
1300	0.000093	14949.909667	13195	19341
1350	0.000097	15635.170000	13617	20384
1400	0.000101	16318.880333	14144	20325
1450	0.000105	16955.622000	14972	21818
1500	0.000109	17697.999667	15327	22208
1550	0.000113	18349.223000	16311	24357
1600	0.000117	19057.509667	16811	24568
1650	0.000122	19733.977333	17398	25187
1700	0.000126	20455.324333	18032	27305
1750	0.000130	21158.126667	18730	27037
1800	0.000134	21880.367333	19391	27307
1850	0.000138	22573.161000	19882	27708
1900	0.000143	23303.542333	20570	30254
1950	0.000147	24029.366333	21549	30208
2000	0.000151	24716.178667	22015	31272

QUICKSORT_SRC

5.3 Pregunta 3: ¿Cuáles son los casos mejor y peor para cada uno de los algoritmos? ¿Qué habrá que modificar en la práctica para calcular estrictamente cada uno de los casos (también el caso medio)?

El caso mejor de Merge Sort es (1/2)Nlog(N) y el peor es NLog(N) + O(N).

En el caso de QuickSort se puede aproximar el mejor caso a 2NLog(N) + O(N) y el peor caso es (N^2 / 2) – (N/2).

5.4 Pregunta 4: ¿Cuál de los dos algoritmos estudiados es más eficiente empíricamente? Compara este resultado con la predicción teórica. ¿Cuál(es) de los algoritmos es/son más eficientes desde el punto de vista de la gestión de memoria? Razona este resultado.

De manera empírica se puede observar que MergeSort es más eficiente que el Quick Sort. Pero en lo que es manejo de memoria el QuickSort es mucho mejor ya que no necesita ninguna memoria extra y el Merge Sort necesita mucha memoria dinámica.

6. Conclusiones finales.

Como conclusión de estos ejercicios hemos aprendido dos algoritmos de ordenación nuevos (QuickSort y MergeSort) y visto el método divide y vencerás. También formas de ordenación de tipo recursivo como en el caso del algoritmo QuickSort.

Comparando los dos algoritmos vemos que hay puntos en los que cada uno es peor que el otro, como en el caso de la memoria que lo hace mucho mejor el QuickSort, o en el peor caso del QuickSort el cual es cuadrático por lo que es mucho peor que el de MergeSort que es logarítmico. Por lo que cada uno tiene su punto fuerte y punto débil.

Pero en comparación con las prácticas pasadas se puede confirmar que la eficiencia de estos algoritmos de tipo divide y vencerás es mucho mejor que cualquier otro algoritmo visto previamente en esta asignatura.