



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: M.I Edgar Tista García

Asignatura: Estructura de Datos y Algoritmos II

Grupo: 9

No de Práctica(s): 2

Integrante(s): Díaz Hernández Marcos Bryan

*No. de Equipo de
cómputo empleado:* Equipo personal

No. de Lista o Brigada: 9

Semestre: 2021-1

Fecha de entrega: 11 de octubre de 2020

Observaciones:

CALIFICACIÓN: _____

Objetivo de la practica

El estudiante identificará la estructura de los algoritmos de ordenamiento BubbleSort, QuickSort y HeapSort.

Introducción

En el reporte siguiente se podrá encontrar una explicación concreta de las dificultades, del aprendizaje adquirido del breve análisis realizado a los algoritmos y de la obtención o cumplimiento del objetivo planteado al comienzo de este. Todo realizado con el fin de poder comprender a futuro como realice esta práctica y para tener mas claro cada concepto.

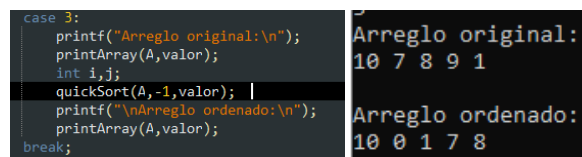
Ejercicios de la practica:

- Ejercicio 1:

El primero consistía en colocar los algoritmos de ordenamiento: BubbleSort, QuickSort, y HeapSort, dentro de la biblioteca de ordenamientos y una vez dentro se tenía que evidenciar si el QuickSort fue visto en clase, por lo que se hizo un análisis del algoritmo. El segundo punto para hacer era el detener el BubbleSort porque este se itera n veces sin importar si ya se ordenó la lista, por lo que es necesario detenerlo. Y por último se tenía que analizar la función Heapify que tiene el HeapSort y describir su funcionamiento.

- Dificultades en el código

El primer problema que tuve fue el QuickSort debido a que al ejecutarlo, eliminaba valores o agregaba negativos, lo que termino siendo un problema para poder analizar la forma en que este operaba y distinguir si ya lo habíamos visto en clase (Imagen 1).



```
case 3:
    printf("Arreglo original:\n");
    printArray(A,valor);
    int i,j;
    quickSort(A,-1,valor);
    printf("\nArreglo ordenado:\n");
    printArray(A,valor);
    break;
```

Arreglo original:
10 7 8 9 1

Arreglo ordenado:
10 0 1 7 8

Imagen 1

Solución: Para poder resolverlo, tuve que checar el algoritmo a mano y eso me genero la evidencia incontrovertible de que no habíamos visto esa variable del algoritmo, además que pude observar que hace iteraciones innecesarias o que se vuelve poco eficiente una vez ordenada la lista, en el momento que devuelve el pivote, ya que pide la recursividad. Al final tuve que colocar en el valor de low, un -1 para que este funcionara, debido a que los valores en los for se manejan uno menos si se deja en el valor original.

```

22     printf("Arreglo original:\n");
23     printArray(A,valor);
24     selectionSort(A,valor);
25     printf("\nArreglo ordenado:\n");
26     printArray(A,valor);
27     break;
28
29     case 2:
30     printf("Arreglo original:\n");
31     printArray(A,valor);
32     insertionSort(A,valor);
33     printf("\nArreglo ordenado:\n");
34     printArray(A,valor);
35     break;
36
37     case 3:
38     printf("Arreglo original:\n");
39     printArray(A,valor);
40     int i,j;
41     quickSort(A,0,valor);
42     printf("\nArreglo ordenado:\n");
43     printArray(A,valor);
44     break;
45
46     case 4:
47     printf("Arreglo original:\n");
48     printArray(A,valor);
49     bubbleSort(A,valor);

```

Cambio en el arreglo:
 -2020288422 1 2 7 6 5 4 3
 Cambio en el arreglo:
 -2020288422 1 2 3 6 5 4
 Cambio en el arreglo:
 -2020288422 1 2 3 6 5 4
 Cambio en el arreglo:
 -2020288422 1 2 3 6 5 4
 Cambio en el arreglo:
 -2020288422 1 2 3 6 5 4
 Cambio en el arreglo:
 -2020288422 1 2 3 4 5
 Cambio en el arreglo:
 -2020288422 1 2 3 4 5
 Cambio en el arreglo:
 -2020288422 1 2 3 4 5
 Arreglo ordenado:
 -2020288422 1 2 3 4 5 6 7 8 9
 Selecciona el tipo de ordenamiento:
 1)Selection

Evidencia de Quick

El segundo problema que encontré fueron las bibliotecas, ya que al implementarlas y querer llevar a cabo buenas practicas que saltaba el undefined de las funciones que tenían dentro, así que incluso definiendo las bibliotecas esto no funcionaba, puede ser por la versión de mi Dev, pero igual lo solucione por medio de colocar los prototipos y las instrucciones en la misma biblioteca (Imagen 2).

```

1  #ifndef _ORDENAMIENTOS_H_
2  #define _ORDENAMIENTOS_H_
3
4
5  #include <stdio.h>
6  #include "utilidades.h"
7
8  void selectionSort(int arreglo[], int n);
9  void insertionSort(int a[], int n);
10 int partition(int arr[],int low, int high);
11 void quickSort(int arr[],int low, int high);
12 void bubbleSort(int a[],int size);
13 void Heapify(int* A, int i, int size);
14 void BuildHeap(int* A, int size);
15 void HeapSort(int* A, int size);
16
17 #endif
18
19 /*int heapSize;
20
21 void Heapify(int* A, int i, int size)
22 {

```

Archivo: C:\Users\Brain\AppData\Local\Temp\cciEmhLQ.o
 Mensaje: SegundoEjercicio.c:(.text+0x1f4): undefined reference to `bubbleSort'
 SegundoEjercicio.c:(.text+0x20f): undefined reference to `printArray'
 SegundoEjercicio.c:(.text+0x22c): undefined reference to `printArray'
 SegundoEjercicio.c:(.text+0x23b): undefined reference to `HeapSort'
 SegundoEjercicio.c:(.text+0x256): undefined reference to `printArray'
 [Error] ld returned 1 exit status

Imagen 2

- Análisis del QuickSort.

El QuickSort o la variable de este, no la habíamos visto en clase, por varias razones, ya que si bien comparte una estructura de recursividad con los demás vistos, eso es porque el algoritmo funciona de esa forma creando sublistas para ordenar y dividir en más.

Mi evidencia no es una ejecución porque al final de cuentas los algoritmos no se pondrían determinar en cuanto a las operaciones que hacen, a menos que se tengan todas las versiones y se ejecuten para el mismo arreglo, pero eso no es posible debido a que no se tiene el algoritmo de uno de los vistos, o por lo menos no se sabe si está bien hecho, porque no pude comprobarlo, de cualquier forma al revisar apuntes y realizar el algoritmo a mano pude comprobar que no funciona igual que los demás.

- En el primero el ordenamiento se escogía como pivote al primero de la lista y después se comparaba con el último para buscar menores que el pivote y una vez que los encontraba, se ponía a buscar mayores que el y eventualmente tras repetirse, quedaban los menores a la izquierda y los mayores a la derecha, para separar en dos listas y ordenar de nuevo.
- El segundo tomaba como pivote al elemento de en medio y después comparaba desde los extremos para encontrar los mayores y menores del pivote, pero este permanecía en su posición los que se intercambiaban eran los valores que resultaban ser mayores o menores, una vez ordenado se dividía, en otras sublistas y se repetía.
- Este algoritmo toma como pivote al elemento del final, y comienza a comprobar todos los valores que sean menores, en caso de encontrarlos pues solo intercambia entre los valores de i y j , lo que permite encontrar el valor más grande y colocarlo a la derecha del valor y posteriormente se comienza a dividir en otras listas, y se repite el proceso, por lo que no son iguales las versiones.

Ahora sabiendo como funciona cada uno pues, ya no hay duda de que es una nueva versión y que hace operaciones distintas para poder llevar a cabo el ordenamiento de la lista y que se cumplan las condiciones del algoritmo, como que los menores estén a la izquierda y los mayores a la derecha.

La evidencia igual muestra el problema de colocar el índice 1 o 0, ya que eso genera una reiteración innecesaria, lo que puede estar causando los problemas que se muestran en los problemas que tuve en la codificación (Imagen 1).

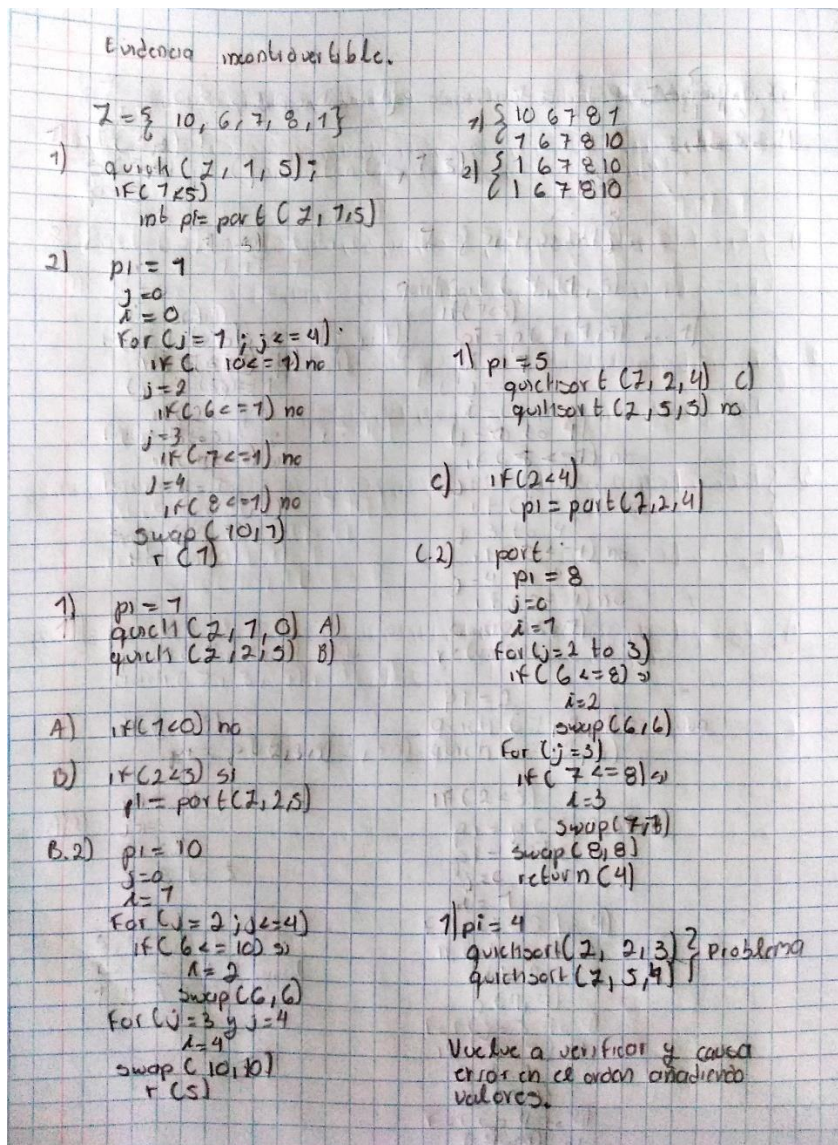


Imagen 1

- Detener el BubbleSort cuando la lista esta ordenada.

Para hacer lo anterior únicamente tuve que ver como funciona y haciendo las pruebas a mano, me di cuenta de que pues comprueba todo pero el for de afuera es quien dicta las iteraciones innecesarias en caso de que ya se ordeno la lista y se sigan las iteraciones, por lo que simplemente coloque una comprobación dentro del segundo for, el cual recorre la lista y si se cumple que ya no hay intercambios en toda esas comprobaciones mi valor de comprobación no va a cambiar por lo que la lista estará ordenada y se tiene que romper el primer for para que no haya más comparaciones (Imagen 2).

```

iteracion=1;
verifica=1;
for(j=0; j<i; j++){
    if(a[j]>a[j+1]){
        swap(&a[j], &a[j+1]);
        verifica=0;
    }
}
printf("Iteracion:%d\n", iteracion);
printArray(a, size);
if(verifica==1){
    i=0;
}

```

Imagen 2

- Describe la función Heapify.

Esta función lo que hace es utilizar el arreglo que le envían y poder verificar los valores de los padres, por medio de comparaciones simples, lo importante de la función es verificar los argumentos y valores que recibe porque con eso se puede determinar el tipo de organización que lleva a cabo si es Top o Bottom.

Lo que recibe es una lista, un índice, y el tamaño de la lista, por lo que hay que describirlos:

- La lista que recibe es un paso directo de la lista original, por lo que se trabaja sobre una lista ya hecha (Imagen 1).

```
void BuildHeap(int* A, int size){
    heapSize = size - 1;
    int i;
    for(i = (size - 1) / 2; i >= 0; i--){
        Heapify(A, i, size);
    }
}
```

Imagen 1

- El índice busca una posición, es decir que se divide sobre dos y se le resta un uno, esto significa que se buscan el ultimo nodo que tuvo hijos y por medio de este se comienza a verificar los valores de los padres con los hijos (Imagen 2).

```
for(i = (size - 1) / 2; i >= 0; i--){
    Heapify(A, i, size);
}
```

```
int l = 2 * i + 1;
int r = 2 * i + 2;
int largest;
```

Imagen 2

Una vez revisado esto nos podemos dar cuenta que es Bottom Up, por las características mencionadas, y para concluir se verifica que los valores de cada hijo no sean mayores al del padre, y en caso de que suceda se intercambian posiciones y se vuelve a comprobar que el nuevo padre no sea menor a otro hijo y al final ya se comienza a eliminar las raíces para comenzar a ordenar los valores.

- Relación con la teoría:

En esta ocasión fue bastante interesante el analizar los algoritmos y fue más fácil intentar realizar el algoritmo a mano con una lista simple, de esta forma se puede distinguir el funcionamiento de cada algoritmo y al mismo tiempo saber donde implementar las cosas, de cualquier forma el peso de clase es bastante ya que es necesario el poder identificar las características de cada algoritmo para poder emitir un juicio acertado.

- Evidencia de implementación

The screenshot shows a C++ IDE with the file `SegundoEjercicio.c` open. The code implements a `selectionSort` function and a `main` function that tests it with an array `1 2 8 7 6 5 4 3`. The output window shows the following text:

```

1 2 8 7 6 5 4 3
Cambio en el arreglo:
1 2 8 7 6 5 4 3
Cambio en el arreglo:
1 2 8 7 6 5 4 3
Cambio en el arreglo:
1 2 8 7 6 5 4 3
Cambio en el arreglo:
1 2 8 7 6 5 4 3
Cambio en el arreglo:
1 2 8 7 6 5 4 3
Cambio en el arreglo:
1 2 3 7 6 5 4
Cambio en el arreglo:
1 2 3 7 6 5 4
Cambio en el arreglo:
1 2 3 7 6 5 4
Cambio en el arreglo:
1 2 3 4 6 5
Cambio en el arreglo:
1 2 3 4 6 5
Cambio en el arreglo:
1 2 3 4 6 5
Cambio en el arreglo:
1 2 3 4 5
Arreglo ordenado:
1 2 3 4 5 6 7 8 9 10
Selecciona el tipo de ordenamiento:

```

The compilation results show 0 errors and 0 warnings, with a compilation time of 3.47s.

Quicksort

The screenshot shows a C++ IDE with the file `ordenamientos.h` open. The code implements a `quickSort` function and a `bubbleSort` function. The output window shows the following text:

```

3)Quick
4)Bubble
5)Heap
Arreglo original:
43523 5634 342 4556 7845 4 2134 462 24523 75
Iteracion:1
5634 342 4556 7845 4 2134 462 24523 75 43523
Iteracion:2
342 4556 5634 4 2134 462 7845 75 24523 43523
Iteracion:3
342 4556 4 2134 462 5634 75 7845 24523 43523
Iteracion:4
342 4 2134 462 4556 75 5634 7845 24523 43523
Iteracion:5
4 342 462 2134 75 4556 5634 7845 24523 43523
Iteracion:6
4 342 462 75 2134 4556 5634 7845 24523 43523
Iteracion:7
4 342 75 462 2134 4556 5634 7845 24523 43523
Iteracion:8
4 75 342 462 2134 4556 5634 7845 24523 43523
Iteracion:9
4 75 342 462 2134 4556 5634 7845 24523 43523
Arreglo ordenado:
4 75 342 462 2134 4556 5634 7845 24523 43523
Selecciona el tipo de ordenamiento:
1)Selection

```

The compilation results show 0 errors and 0 warnings, with a compilation time of 0.50s.

BubbleSort


```
25
26 void Heapify(int* A, int i, int size)
27 {
28     int l = 2 * i + 1;
29     int r = 2 * i + 2;
30     int largest;
31
32     if(l <= heapSize && A[l] > A[i])
33         largest = l;
34     else
35         largest = i;
36     if(r <= heapSize && A[r] > A[largest]) //Comprueba el valor de Los hijos con el padre
37         largest = r;
38     if(largest != i){ //Busca el posible caso que Los hijos sean mayores al padre
39         swap(&A[i], &A[largest]);
40         printf("Intercambio:\n");
41         printArray(A, size);
42         Heapify(A, largest, size); //En caso de que un hijo sea mayor al padre se llama a la funcion del padre
43     }
44 }
45
46 void BuildHeap(int* A, int size){
47     heapSize = size - 1;
48     int i;
49     for(i = (size - 1) / 2; i >= 0; i--){ //Busca Los nodos finales, es decir los que no tienen hijos
50         Heapify(A, i, size);
51     }
52     printf("Terminó de construir el HEAP \n", 162);
53 }
```

3)Quick
4)Bubble
5)Heap

Arreglo original:
10 6 7 8 1
Intercambio:
10 8 7 6 1
Terminó de construir el HEAP
Iteración HS:
1 8 7 6 10
Intercambio:
8 1 7 6 10
Iteración HS:
8 6 7 1 10
Intercambio:
1 6 7 8 10
Intercambio:
7 6 1 8 10
Iteración HS:
1 6 7 8 10
Intercambio:
6 1 7 8 10
Iteración HS:
1 6 7 8 10
Arreglo ordenado:
1 6 7 8 10
Selecciona el tipo de ordenamiento:

HeapSort

- Ejercicio 2:

El segundo ejercicio consistía en poder implementar cada uno de los algoritmos y el colocar las operaciones correspondientes para verificar los cambios de los valores en el arreglo, después de cada interacción.

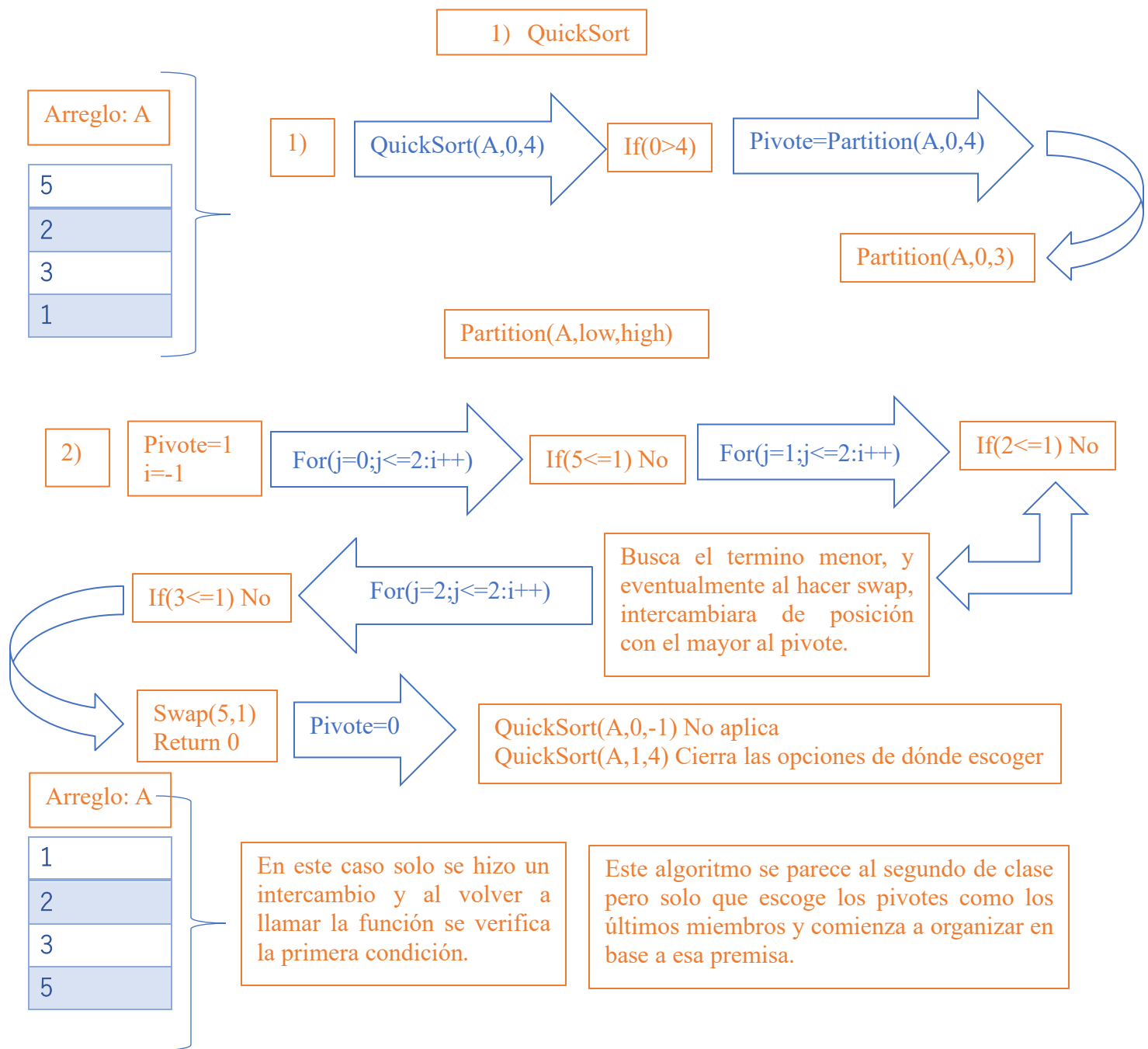
- Dificultades en el código

En esta ocasión no tuve dificultades que resolver ya que las había resuelto en el anterior y así que esta esta parte de la practica la puedo llamar felicidad.

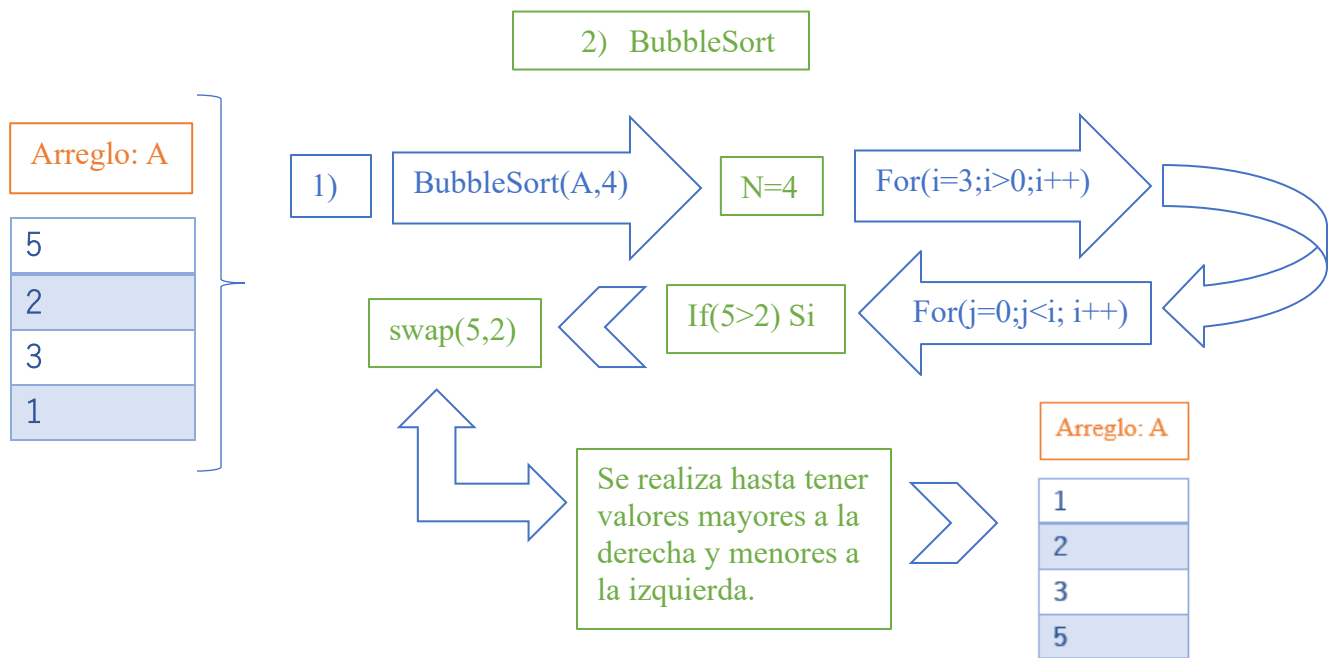
- Diagrama de funcionamiento:

Sin embargo es necesario el conocer la forma de funcionamiento de cada algoritmo, por lo que aplicare mis diagramas en forma de explicación de lo que sucede y de igual forma esto pues será la parte principal de este ejercicio que consiste en comprender las operaciones que realizan los algoritmos y como se repiten.

- o En cuanto a los algoritmos se puede decir que el único mas predecible es el BubbleSort porque si el termino menor se encuentra al final este tendrá que llevar a cabo cada iteración y en cada una solo se moverá una posición.

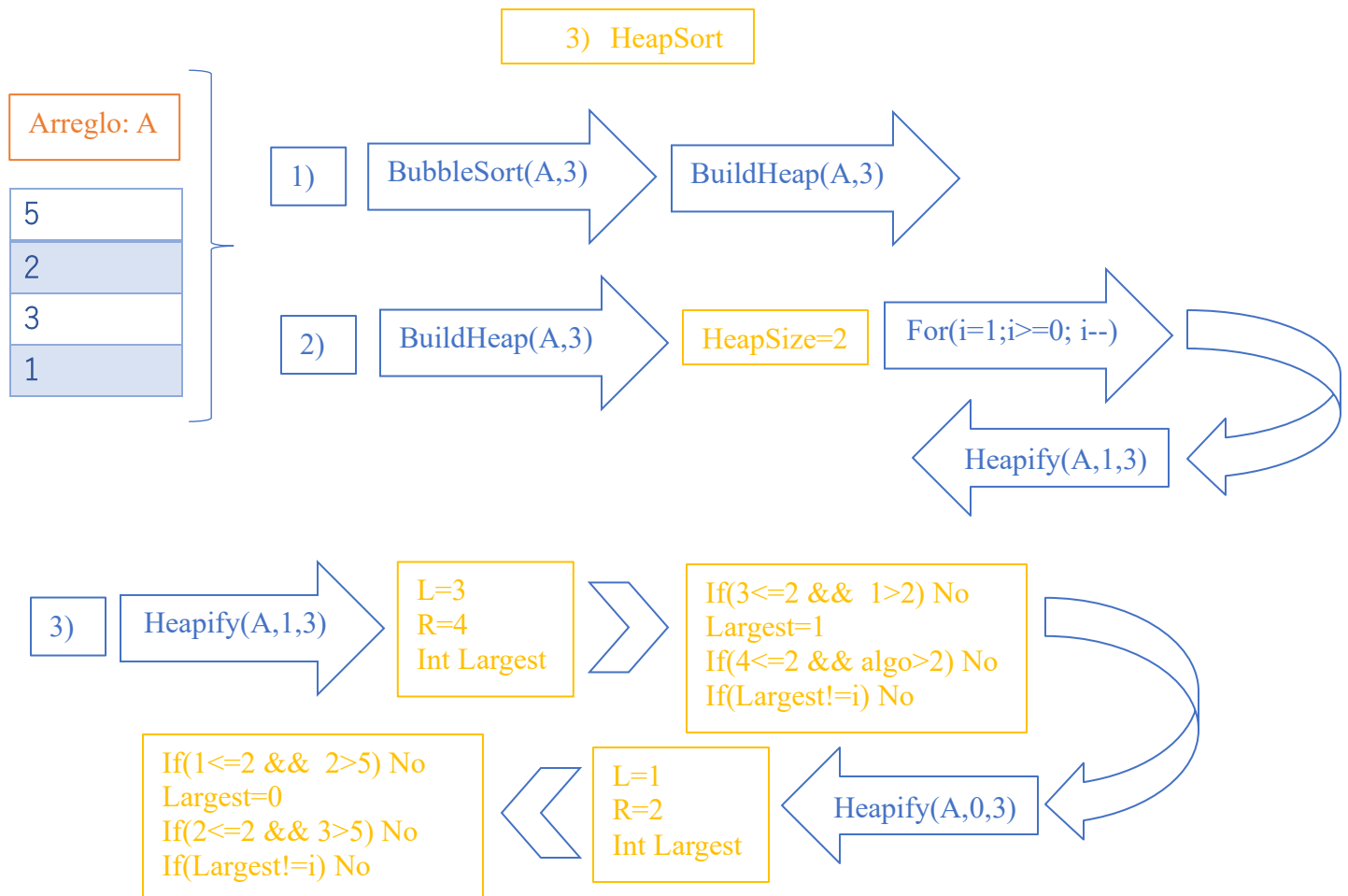


Algo importante que se tiene que considerar con el QuickSort, son los índices ya que si no se conocen, no es posible implementar el algoritmo, y en ese caso el algoritmo no es conveniente para su uso, de otro lado es bastante eficiente.



En el caso de que se ordene al primer ciclo del segundo For, se tiene que esperar a que el algoritmo termine todos los for, de modo que es bastante innecesario el tener que esperar cuando ya está listo el arreglo.

Como es posible ver el arreglo quedo listo en las iteraciones del segundo for, pero todavía faltan que se realicen las demás iteraciones hasta que acabe el for principal



The screenshot shows a C++ IDE with the following components:

- Source Code (ordenamientos.h):**

```

85 void quickSort(int arr[], int low, int high){
86     if(low < high){
87         int pi = partition(arr, low, high);
88         quickSort(arr, low, pi-1);
89         quickSort(arr, pi+1, high);
90     }
91 }
92
93 void bubbleSort(int a[], int size){
94     int i, j, n, verifica=0, iteracion=0;
95     n = size;
96     for(i=n-1; i>0; i--){
97         iteracion++;
98         for(j=0; j<i; j++){
99             //En caso de que el valor cambie si
100             if(a[j]>a[j+1]){
101                 swap(&a[j], &a[j+1]);
102                 verifica=1; //Cuando cambio se tiene que se hic
103             }
104         }
105         printf("Iteracion:%d\n", iteracion);
106         printArray(a, size);
107         if(verifica==0){
108             i=0;
109         }
110     }
111 }

```
- Execution Output:**

```

2) Insertion
3) Quick
4) Bubble
5) Heap
Arreglo original:
345 5745 89 654 364 2345 24576 567 564 78
Iteracion:1
345 89 654 364 2345 5745 567 564 78 24576
Iteracion:2
89 345 364 654 2345 567 564 78 5745 24576
Iteracion:3
89 345 364 654 567 564 78 2345 5745 24576
Iteracion:4
89 345 364 567 564 78 654 2345 5745 24576
Iteracion:5
89 345 364 564 78 567 654 2345 5745 24576
Iteracion:6
89 345 364 78 564 567 654 2345 5745 24576
Iteracion:7
89 345 78 364 564 567 654 2345 5745 24576
Iteracion:8
89 78 345 364 564 567 654 2345 5745 24576
Iteracion:9
78 89 345 364 564 567 654 2345 5745 24576
Arreglo ordenado:
78 89 345 364 564 567 654 2345 5745 24576
Selecciona el tipo de ordenamiento:

```
- Compiler Output:**

```

Compilation results...
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\Brain\Documents\EDA II\P2\Diaz Hernández Marcos G9 P2 V1\SegundoEjercicio.exe
- Output Size: 132.7451171875 KiB
- Compilation Time: 0.45s

```

BubbleSort

The screenshot shows a C++ IDE with the following components:

- Source Code (ordenamientos.h):**

```

25
26 void Heapify(int* A, int i, int size)
27 {
28     int l = 2 * i + 1;
29     int r = 2 * i + 2;
30     int largest;
31
32     if(l <= heapSize && A[l] > A[i])
33         largest = l;
34     else
35         largest = i;
36     if(r <= heapSize && A[r] > A[largest]) //Comprueba el valor de los hijos con el
37         largest = r;
38     if(largest != i){ //Busca el posible caso que los hijos sean mayores al padre
39         swap(&A[i], &A[largest]);
40         printf("Intercambio:\n");
41         printArray(A, size);
42         Heapify(A, largest, size); //En caso de que un hijo sea mayor al padre se di
43     }
44 }
45
46 void BuildHeap(int* A, int size){
47     heapSize = size - 1;
48     int i;
49     for(i = (size - 1) / 2; i >= 0; i--){ //Busca los nodos finales, es decir los
50         Heapify(A, i, size);
51     }
52     printf("Terminó de construir el HEAP\n", 162);
53 }

```
- Execution Output:**

```

3) Quick
4) Bubble
5) Heap
Arreglo original:
10 6 7 8 1
Intercambio:
10 8 7 6 1
Terminó de construir el HEAP
Iteracion HS:
1 8 7 6 10
Intercambio:
8 1 7 6 10
Intercambio:
8 6 7 1 10
Iteracion HS:
1 6 7 8 10
Intercambio:
7 6 1 8 10
Iteracion HS:
1 6 7 8 10
Intercambio:
6 1 7 8 10
Iteracion HS:
1 6 7 8 10
Arreglo ordenado:
1 6 7 8 10
Selecciona el tipo de ordenamiento:

```
- Compiler Output:**

```

Compilation results...
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\Brain\Documents\EDA II\P2\Diaz Hernández Marcos G9 P2 V1\SegundoEjercicio.exe
- Output Size: 132.7451171875 KiB
- Compilation Time: 0.45s

```

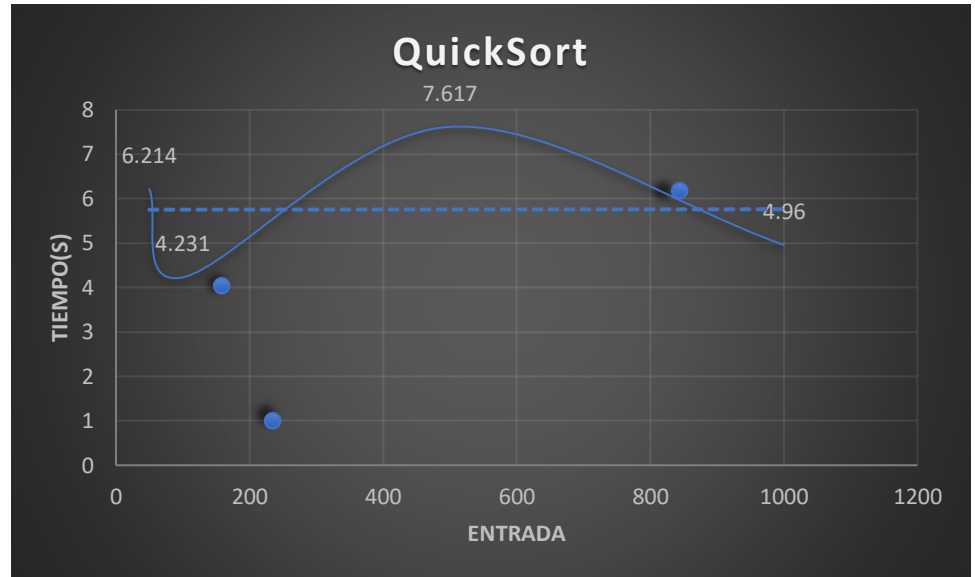
HeapSort

• Ejercicio 3:

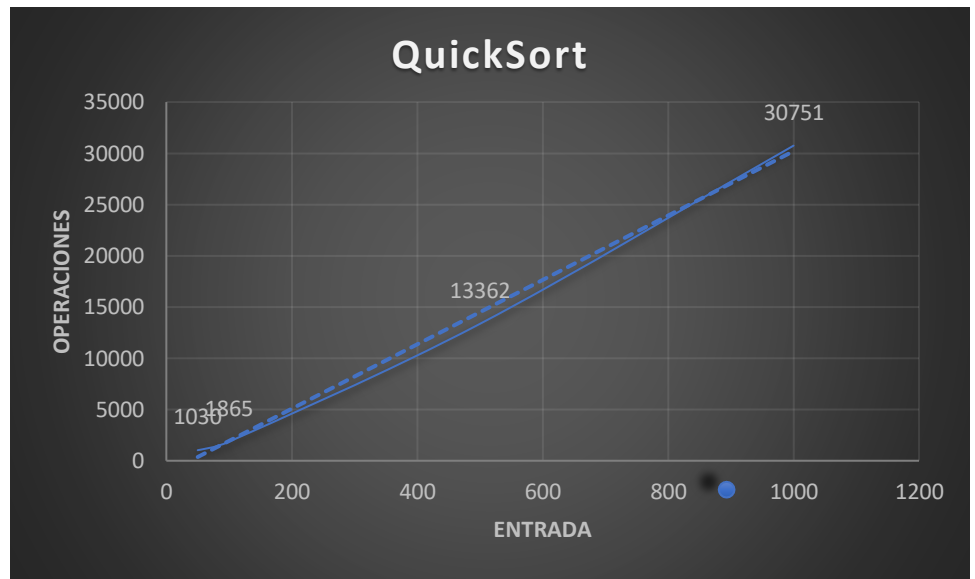
Este ejercicio busca el análisis de la complejidad de cada algoritmo, por lo que es necesario el evaluar y realizar pruebas de estrés en cada algoritmo, y a partir de graficas el poder evaluar el tipo de complejidad que tienen.

- Análisis de complejidad.
 - QuickSort

Entrada	Tiempo
50	6.214
100	4.231
500	7.617
1000	4.96



Entrada	Operaciones
50	1030
100	1865
500	13362
1000	30751



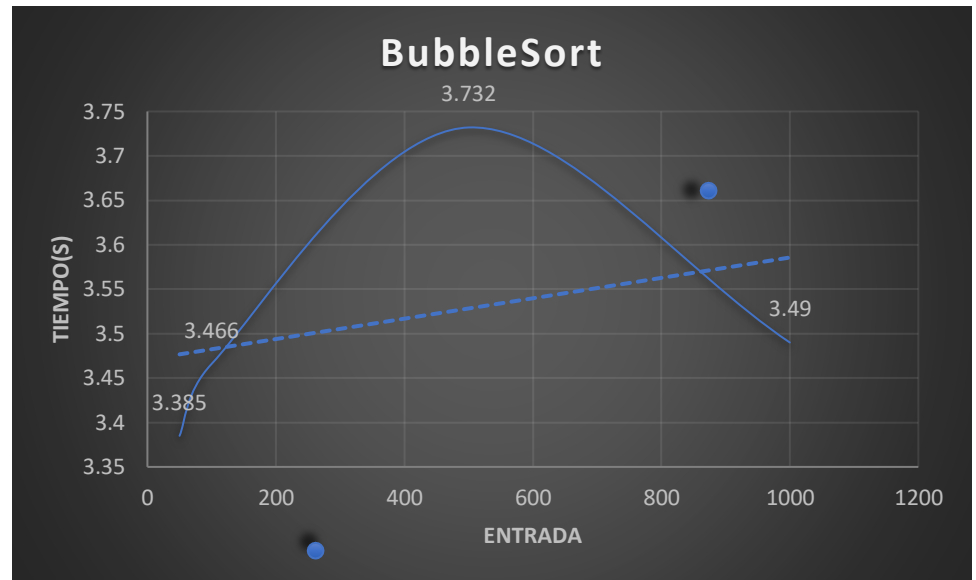
En el caso de Quicksort para la complejidad se puede ver que el tiempo de ejecución no fue proporcional a la cantidad de la entrada, esto significa que intervinieron varios factores, como las ventanas abiertas, el sistema operativo y el procesador, sin embargo se tiene una subida en base a las tres primeras entradas, lo que podría afectar la grafica en general que tiene un tendencia lineal que no corresponde con la gráfica, y los primeros tres puntos nos muestran que es una gráfica de $n \log(n)$, es decir que tiene la tendencia de ser de complejidad $O(n \log n)$, checar comparación de la graficas.

Como segunda muestra se utiliza la grafica de entrada y las operaciones correspondientes, esta segunda grafica muestra la tendencia que existe entre el tiempo y las operaciones, con esta segunda grafica podemos concretar la hipótesis del párrafo anterior.

- o BubbleSort

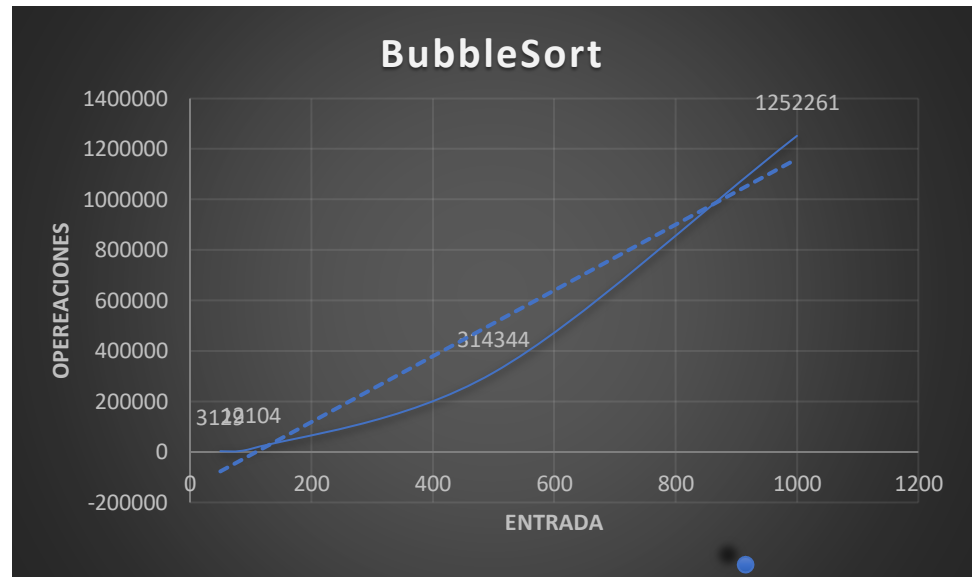
Entrada Tiempo

50	3.385
100	3.466
500	3.732
1000	3.49



Entrada Operaciones

50	3129
100	12104
500	314344
1000	1252261



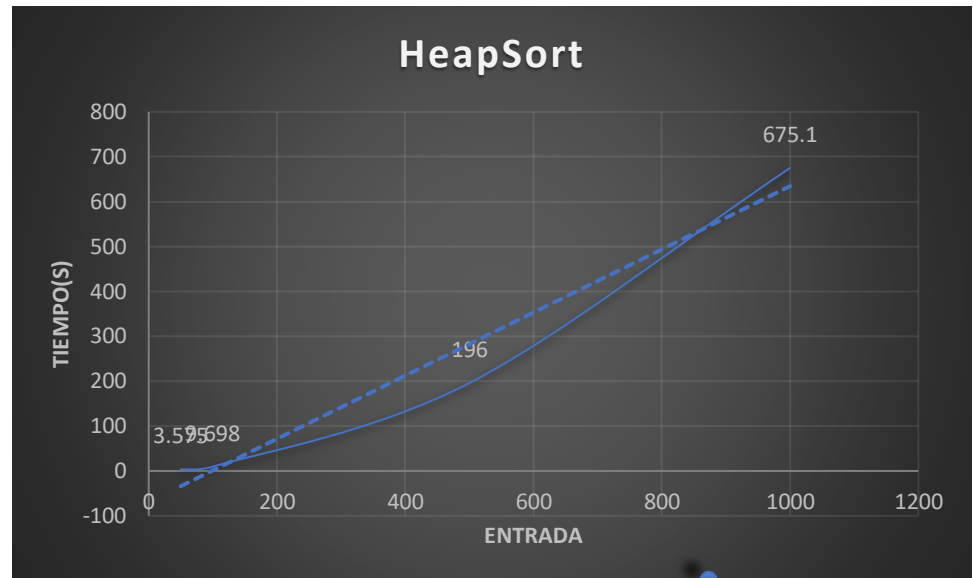
En este algoritmo vuelve a suceder algo similar con el tiempo de ejecución y el tamaño de la entrada lo que significa que de nuevo existen factores que pueden modificar la velocidad del procesamiento de ciertas actividades y de los algoritmos a la par, entonces analizando la grafica podemos ver que existe una tendencia de crecimiento en el tiempo, pero de pronto baja.

Para comprobar esto tenemos la segunda grafica que claramente nos muestra la forma de esta, siendo cuadrática, lo que significa que los tres primeros tiempos de la primer grafica tenían una tendencia de crecimiento y que a cada entrada las operaciones que se realizan son cada vez mayores al grado de ser el cuadrado del valor.

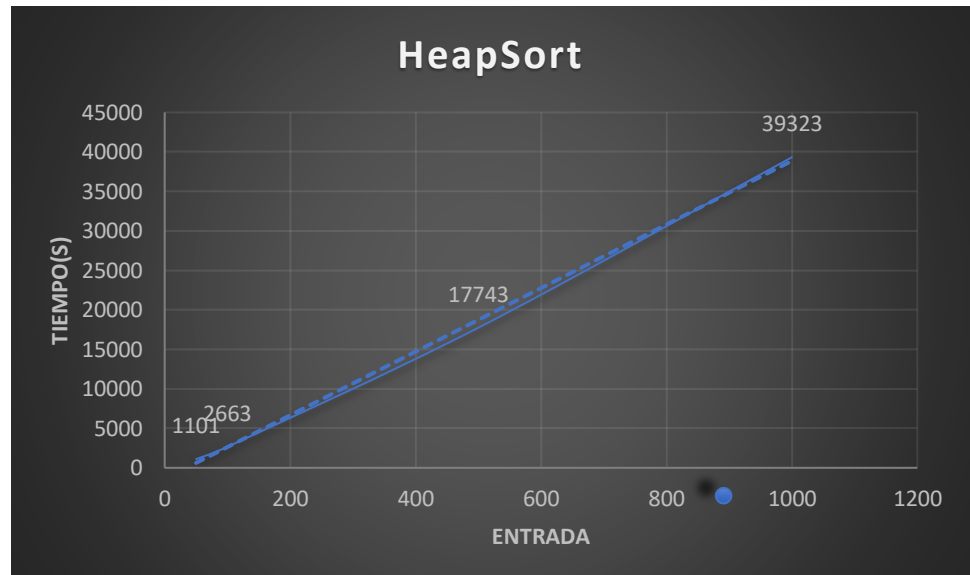
Por lo tanto la complejidad es de $O(n^2)$.

- HeapSort

Entrada	Tiempo
50	3.575
100	9.698
500	196
1000	675.1



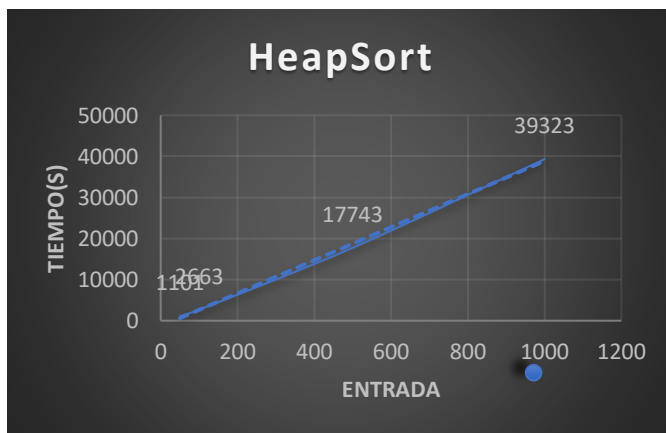
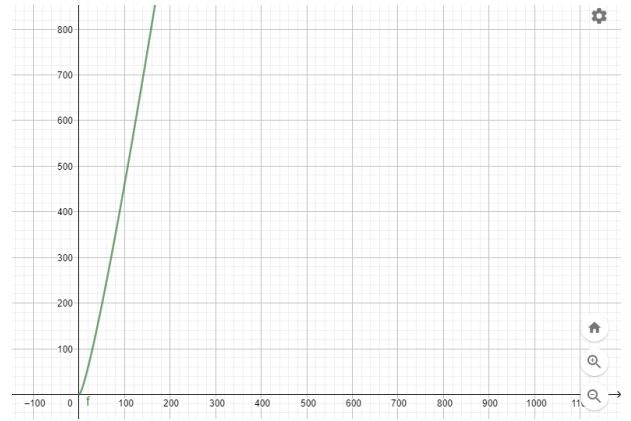
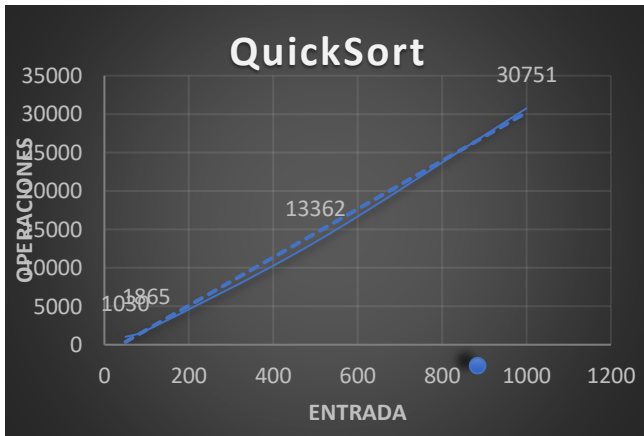
Entrada	Operaciones
50	1101
100	2663
500	17743
1000	39323



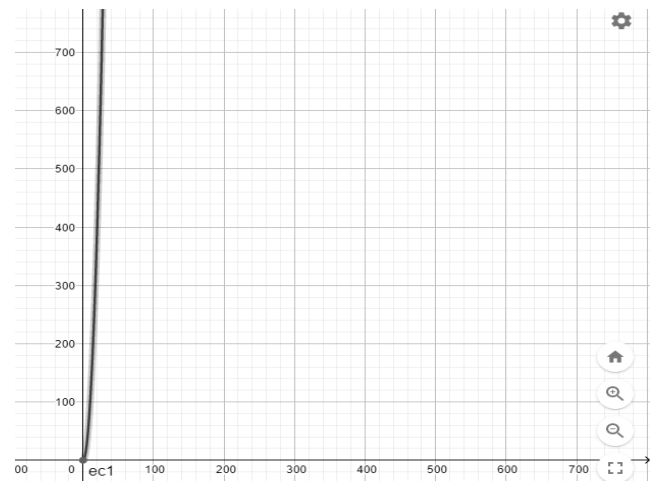
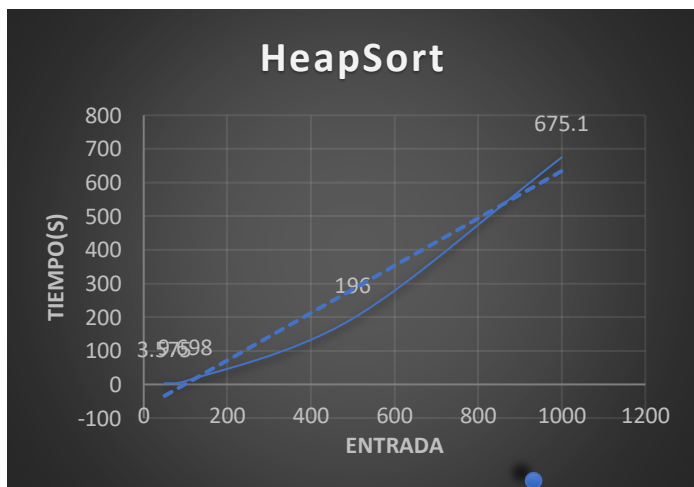
El caso de HeapSort me pareció bastante curioso ya que el tiempo que toma para realizar las operaciones en base a la entrada es mucho, y eso toma sentido al ver el código y al analizarlo, ya que tiene que evaluar valores e intercambiarlos, eliminar, por lo que se vuelve bastante amplia la operación, sin embargo están no crecen demasiado, se mantienen pocas si las comparamos con las del Bubble, esto da como resultado que al igual que el QuickSort, se tenga un crecimiento en tiempo y en operaciones realizadas de acuerdo a la entrada, pero el crecimiento de las operaciones no es mucho es proporcional y similar al del Quick.

Incluso la primera gráfica muestra la tendencia semi-cuadrática pero en la segunda se ve la linealidad que compone al algoritmo, por lo que se tiene una complejidad de $O(n \log(n))$.

- Comparación de graficas.



Al ver las graficas podemos ver que existe la tendencia de cumplir con la gráfica $n \log n$, igual es necesario el tener un comparado de graficas y la semejanza pero de este modo podemos darnos cuenta de la forma inicial de la grafica y como coinciden con las de tiempo y entrada.



De este sencillo modo es posible comparar las graficar de complejidad y ver que gran parte de dictamen es dado por la apertura de nuestras graficas porque la n^2 , esta mas pegada al eje y, y la grafica de $n \log n$, está en un punto medio entre la $y=x$, y entre la n^2 , me parece un modo bastante bueno y sencillo.

- Relación con la teoría:

Este ejercicio esta bastante cargado de análisis, y mas que nada de buen ojo, porque si bien es posible tener factores que determinen la complejidad dentro de la formalidad, la mayoría de estos métodos son matemáticas específicas para la solución de los problemas, tanto inducción como identidades que en este caso no se utilizaron pero no obstante, es posible llegar a una decisión de la complejidad.

- Evidencia de implementación

The screenshot shows a C++ IDE with the following code in `OrdenamientosModificados.h`:

```

68 int ContadorQFor=0;
69 int partition(int arr[],int low, int high){
70     int pivot=arr[high];
71     int j,i=(low-1);
72     for(j=low;j<=(high-1);j++){
73         ContadorQFor++;
74         ContadorQComparaciones++;
75         if(arr[j]<=pivot){
76             i++;
77             swap(&arr[i],&arr[j]);
78             ContadorQIntercambios++;
79         }
80     }
81     swap(&arr[i+1],&arr[high]);
82     ContadorQIntercambios++;
83     return (i+1);
84 }
85
86 void quickSort(int arr[],int low, int high){
87     ContadorQComparaciones++;
88     if(low<high){
89         int pi=partition(arr,low,high);
90         quickSort(arr,low,pi-1);
91         quickSort(arr,pi,high);
92     }
93     printf("Numero de comparaciones:%d\n Numero de intercambios:%d\n Numero de entradas al for:%d\n",ContadorQComparaciones,ContadorQIntercambios,ContadorQFor);
94 }
95
96 void bubbleSort(int a[],int size){
97     int i,j,n,verifica,ContadorBComparaciones=0, ContadorBFor=0, ContadorBIntercambios=0;
98     n=size;
99     for(i=n-1;i>0;i--){
100         ContadorBFor++;
101         for(j=0;j<i;j++){
102             ContadorBFor++;
103             ContadorBComparaciones++;
104             if(a[j]>a[j+1]){
105                 swap(&a[j],&a[j+1]);
106                 ContadorBIntercambios++;
107             }
108         }
109         verifica=0;
110         for(j=0;j<i;j++){
111             ContadorBFor++;
112             if(a[j]<a[j+1]){
113                 verifica=1;
114             }
115         }
116         if(verifica==0) break;
117     }
118     printf("Numero de comparaciones:%d\n Numero de intercambios:%d\n Numero de entradas al for:%d\n",ContadorBComparaciones,ContadorBIntercambios,ContadorBFor);
119 }

```

The execution output shows the following statistics:

```

Numero de entradas al for:365
Numero de comparaciones:437
Numero de intercambios:227
Numero de entradas al for:365
Numero de comparaciones:437
Numero de intercambios:227
Numero de entradas al for:365
Numero de comparaciones:437
Numero de intercambios:227
Numero de entradas al for:365
Numero de comparaciones:438
Numero de intercambios:227
Numero de entradas al for:365
Numero de comparaciones:438
Numero de intercambios:227
Numero de entradas al for:365
Numero de comparaciones:438
Numero de intercambios:227
Arreglo ordenado:
384 2499 3244 3543 3960 4779 7076 7671 7852 7955 81
34 8922 9050 10174 10643 13050 13586 14086 14774 14
996 17439 18074 18188 19400 19479 19799 22072 22099
22374 22388 22628 23022 23269 24004 24271 24428 25
312 25995 26138 26209 26608 28084 28629 28916 29253
29347 29904 31840 31877 31977
-----
Process exited after 6.214 seconds with return value 3221226356
Presione una tecla para continuar . . .

```

QuickSort – 50

The screenshot shows a C++ IDE with the following code in `OrdenamientosModificados.h`:

```

15 A[i] = rand()%10000000;
16 }
17
18 printf("\nSeleccione el tipo de ordenamiento:\n 1)QuickSort\n 2)BubbleSort\n 3)HeapSort\n");
19 scanf("%d",&opcion);
20
21 switch(opcion){
22     case 1:
23         printf("Arreglo original:\n");
24         printArray(A,valor);
25         int i,j;
26         quickSort(A,1,valor);
27         printf("Arreglo ordenado:\n");
28         printArray(A,valor);
29         break;
30     case 2:
31         printf("Arreglo original:\n");
32         printArray(A,valor);
33         bubbleSort(A,valor);
34         printf("Arreglo ordenado:\n");
35         printArray(A,valor);
36         break;
37     case 3:
38         printf("Arreglo original:\n");
39         printArray(A,valor);
40         HeapSort(A,valor);
41         printf("Arreglo ordenado:\n");
42         printArray(A,valor);
43         break;
44     default:
45         opcion=4;
46         break;
47 }
48 free(A);

```

The execution output shows the following statistics:

```

Numero de entradas al for:646
Numero de comparaciones:777
Numero de intercambios:442
Numero de entradas al for:646
Numero de comparaciones:777
Numero de intercambios:442
Numero de entradas al for:646
Numero de comparaciones:777
Numero de intercambios:442
Numero de entradas al for:646
Numero de comparaciones:777
Numero de intercambios:442
Numero de entradas al for:646
Numero de comparaciones:777
Numero de intercambios:442
Numero de entradas al for:646
Numero de comparaciones:777
Numero de intercambios:442
Arreglo ordenado:
167 328 581 721 760 1045 1647 2390 2748 2810 3634 3688 3795 4781
5386 6423 6610 7637 7962 8085 8174 8233 8487 8570 8763 9828 10625
10695 10802 11028 11140 11332 11488 11580 11626 11716 12070 1230
9 12436 12594 13431 13529 13661 13852 13868 13975 14022 14318 147
35 14750 15443 18331 18864 19014 19541 19717 19762 20581 20741 20
759 21104 21152 21158 21576 21604 21747 21858 22114 22293 22641 2
2918 22979 23088 24068 24078 24776 24836 25463 25563 26477 26487
26808 26896 27278 27641 27833 27880 27897 28566 29636 29902 29990
30703 31034 31332 32300 32354 32383 32468 32473
-----
Process exited after 4.231 seconds with return value 3221226356
Presione una tecla para continuar . . .

```

QuickSort - 100

C:\Users\Brain\Documents\EDA II\P2\Díaz Hernández Marcos G9 P2 V1\TercerEjercicio.c - [Executing] - Dev-C++ 5.11

Archivo Edición Buscar Ver Proyecto Ejecutar Herramientas AStyle Ventana Ayuda

(globals)

Proyecto Clases(Funciones) TercerEjercicio.c OrdenamientosModificados.h

```
15 }
16 A[i] = rand()%1000000;
17 }
18 printf("\nSelecciona el tipo de ordenamiento:");
19 scanf("%d",&opcion);
20
21 switch(opcion){
22     case 1:
23         printf("Arreglo original:\n");
24         printArray(A,valor);
25         int i,j;
26         quickSort(A,1,valor);
27         printf("\nArreglo ordenado:\n");
28         printArray(A,valor);
29         break;
30     case 2:
31         printf("Arreglo original:\n");
32         printArray(A,valor);
33         bubbleSort(A,valor);
34         printf("\nArreglo ordenado:\n");
35         printArray(A,valor);
36         break;
37     case 3:
38         printf("Arreglo original:\n");
39         printArray(A,valor);
40         break;
41 }
```

Numero de entradas al for:4994
Numero de comparaciones:5667
Numero de intercambios:2701
Numero de entradas al for:4994

Arreglo ordenado:

72 72 121 150 161 175 343 498 615 620 706 759 829 916 962 1184 1344 1474 1534 1595 1620 1721 1746 1896
2033 2038 2136 2292 2339 2359 2386 2534 2575 2783 2818 3008 3059 3083 3224 3249 3357 3363 3483 3510 3
538 3582 3751 3803 3838 4001 4041 4252 4277 4290 4327 4349 4445 4543 4812 4880 4895 4967 4978 5018 503
3 5037 5044 5048 5096 5136 5138 5167 5229 5256 5330 5341 5350 5424 5463 5563 5668 5670 5689 5743 5753
5835 5907 5909 6068 6160 6180 6184 6195 6398 6422 6457 6499 6516 6663 6668 6674 6718 6722 6808 6885 68
88 6901 6998 7033 7036 7058 7114 7118 7154 7177 7317 7318 7355 7493 7426 7512 7522 7598 7645 7672 7696
7776 7820 7844 8150 8180 8206 8317 8321 8325 8357 8445 8533 8558 8609 8648 8751 8762 8880 9060 9061 9
118 9266 9293 9433 9470 9716 9756 9794 9882 9917 10028 10089 10191 10230 10270 10392 10510 10671 10688
10731 10806 10965 11031 11081 11195 11236 11332 11344 11464 11534 11693 11705 11721 11747 11761 11844
11934 12103 12174 12248 12383 12408 12466 12577 12602 12732 12756 13230 13743 13751 13773 13827 13927
13954 14010 14063 14094 14107 14272 14466 14540 14622 14632 14685 14793 14793 14857 14891 14936 14956
14989 15054 15196 15272 15295 15464 15517 15790 15873 15882 15970 15985 15994 16053 16151 16219 16382
16411 16461 16475 16479 16729 16806 16816 16872 16886 16940 16960 16974 17092 17108 17189 17196 17179
17220 17270 17314 17316 17371 17422 17496 17552 17590 17672 17695 17819 17837 17871 17876 17948 17967
18047 18101 18153 18158 18184 18246 18252 18421 18422 18465 18477 18508 18719 18757 18843 19084 19156
19231 19542 19546 19555 19561 19586 19613 19672 19698 19788 19804 19871 19874 19890 19942 20025 20065
20077 20132 20259 21023 20442 20523 20583 20719 20800 20904 20911 20969 21082 21227 21295 21331 21346
21413 21425 21438 21495 21587 21628 21763 22072 22084 22100 22100 22159 22190 22203 22388 22391
22433 22451 22562 22673 22740 22802 22882 22893 22945 23061 23095 23115 23171 23324 23376 23450 23546
23608 23629 23682 23683 23746 23839 23841 23904 23911 23915 23927 23962 24019 24048 24109 24114 24118
24489 24640 24716 24743 24900 24915 24916 24934 25084 25119 25123 25179 25221 25264 25272 25298 25376
25391 25404 25447 25518 25768 25843 25844 25893 25957 26053 26259 26433 26435 26572 26689 26726 26766
26789 26895 26898 26903 26903 26953 26969 27073 27336 27346 27393 27498 27505 27516 27544 27596 27611
27676 27710 27716 27795 27818 27857 27913 27926 27954 28025 28040 28061 28106 28211 28277 28370 28406
28527 28530 28536 28564 28728 28875 28879 28894 28915 28947 29100 29144 29200 29327 29439 29571 29618
29639 29715 29742 29797 29879 29924 29940 29946 29988 30002 30029 30038 30089 30187 30368 30476 30551
30582 30610 30743 30745 30912 30923 30934 30939 30939 30967 31007 31017 31113 31172 31222 31227 31270
31326 31393 31475 31538 31651 31799 32062 32193 32255 32296 32471 32554

Process exited after 7.617 seconds with return value 3221226356
Presione una tecla para continuar . . .

Compilador Recursos Registro de Compilación Depuración Resultados

Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\Brain\Documents\EDA II\P2\Díaz Hernández Marcos G9 P2 V1\TercerEjercicio.exe
- Output Size: 132.9091796875 KiB
- Compilation Time: 0.53s

Line: 1 Col: 1 Sel: 0 Lines: 53 Length: 1107 Insertar Done parsing in 0.015 seconds

Escribe aquí para buscar

QuickSort – 500

```
14962 14964 14967 14970 14973 14976 14979 14982 14985 14988 14991 14994 14997 15000 15003 15006 15009 15012 15015 15018 15021 15024 15027 15030 15033 15036 15039 15042 15045 15048 15051 15054 15057 15060 15063 15066 15069 15072 15075 15078 15081 15084 15087 15090 15093 15096 15099 15102 15105 15108 15111 15114 15117 15120 15123 15126 15129 15132 15135 15138 15141 15144 15147 15150 15153 15156 15159 15162 15165 15168 15171 15174 15177 15180 15183 15186 15189 15192 15195 15198 15201 15204 15207 15210 15213 15216 15219 15222 15225 15228 15231 15234 15237 15240 15243 15246 15249 15252 15255 15258 15261 15264 15267 15270 15273 15276 15279 15282 15285 15288 15291 15294 15297 15300 15303 15306 15309 15312 15315 15318 15321 15324 15327 15330 15333 15336 15339 15342 15345 15348 15351 15354 15357 15360 15363 15366 15369 15372 15375 15378 15381 15384 15387 15390 15393 15396 15399 15402 15405 15408 15411 15414 15417 15420 15423 15426 15429 15432 15435 15438 15441 15444 15447 15450 15453 15456 15459 15462 15465 15468 15471 15474 15477 15480 15483 15486 15489 15492 15495 15498 15501 15504 15507 15510 15513 15516 15519 15522 15525 15528 15531 15534 15537 15540 15543 15546 15549 15552 15555 15558 15561 15564 15567 15570 15573 15576 15579 15582 15585 15588 15591 15594 15597 15600 15603 15606 15609 15612 15615 15618 15621 15624 15627 15630 15633 15636 15639 15642 15645 15648 15651 15654 15657 15660 15663 15666 15669 15672 15675 15678 15681 15684 15687 15690 15693 15696 15699 15702 15705 15708 15711 15714 15717 15720 15723 15726 15729 15732 15735 15738 15741 15744 15747 15750 15753 15756 15759 15762 15765 15768 15771 15774 15777 15780 15783 15786 15789 15792 15795 15798 15801 15804 15807 15810 15813 15816 15819 15822 15825 15828 15831 15834 15837 15840 15843 15846 15849 15852 15855 15858 15861 15864 15867 15870 15873 15876 15879 15882 15885 15888 15891 15894 15897 15900 15903 15906 15909 15912 15915 15918 15921 15924 15927 15930 15933 15936 15939 15942 15945 15948 15951 15954 15957 15960 15963 15966 15969 15972 15975 15978 15981 15984 15987 15990 15993 15996 15999 16002 16005 16008 16011 16014 16017 16020 16023 16026 16029 16032 16035 16038 16041 16044 16047 16050 16053 16056 16059 16062 16065 16068 16071 16074 16077 16080 16083 16086 16089 16092 16095 16098 16101 16104 16107 16110 16113 16116 16119 16122 16125 16128 16131 16134 16137 16140 16143 16146 16149 16152 16155 16158 16161 16164 16167 16170 16173 16176 16179 16182 16185 16188 16191 16194 16197 16200 16203 16206 16209 16212 16215 16218 16221 16224 16227 16230 16233 16236 16239 16242 16245 16248 16251 16254 16257 16260 16263 16266 16269 16272 16275 16278 16281 16284 16287 16290 16293 16296 16299 16302 16305 16308 16311 16314 16317 16320 16323 16326 16329 16332 16335 16338 16341 16344 16347 16350 16353 16356 16359 16362 16365 16368 16371 16374 16377 16380 16383 16386 16389 16392 16395 16398 16401 16404 16407 16410 16413 16416 16419 16422 16425 16428 16431 16434 16437 16440 16443 16446 16449 16452 16455 16458 16461 16464 16467 16470 16473 16476 16479 16482 16485 16488 16491 16494 16497 16500 16503 16506 16509 16512 16515 16518 16521 16524 16527 16530 16533 16536 16539 16542 16545 16548 16551 16554 16557 16560 16563 16566 16569 16572 16575 16578 16581 16584 16587 16590 16593 16596 16599 16602 16605 16608 16611 16614 16617 16620 16623 16626 16629 16632 16635 16638 16641 16644 16647 16650 16653 16656 16659 16662 16665 16668 16671 16674 16677 16680 16683 16686 16689 16692 16695 16698 16701 16704 16707 16710 16713 16716 16719 16722 16725 16728 16731 16734 16737 16740 16743 16746 16749 16752 16755 16758 16761 16764 16767 16770 16773 16776 16779 16782 16785 16788 16791 16794 16797 16800 16803 16806 16809 16812 16815 16818 16821 16824 16827 16830 16833 16836 16839 16842 16845 16848 16851 16854 16857 16860 16863 16866 16869 16872 16875 16878 16881 16884 16887 16890 16893 16896 16899 16902 16905 16908 16911 16914 16917 16920 16923 16926 16929 16932 16935 16938 16941 16944 16947 16950 16953 16956 16959 16962 16965 16968 16971 16974 16977 16980 16983 16986 16989 16992 16995 16998 17001 17004 17007 17010 17013 17016 17019 17022 17025 17028 17031 17034 17037 17040 17043 17046 17049 17052 17055 17058 17061 17064 17067 17070 17073 17076 17079 17082 17085 17088 17091 17094 17097 17100 17103 17106 17109 17112 17115 17118 17121 17124 17127 17130 17133 17136 17139 17142 17145 17148 17151 17154 17157 17160 17163 17166 17169 17172 17175 17178 17181 17184 17187 17190 17193 17196 17199 17202 17205 17208 17211 17214 17217 17220 17223 17226 17229 17232 17235 17238 17241 17244 17247 17250 17253 17256 17259 17262 17265 17268 17271 17274 17277 17280 17283 17286 17289 17292 17295 17298 17301 17304 17307 17310 17313 17316 17319 17322 17325 17328 17331 17334 17337 17340 17343 17346 17349 17352 17355 17358 17361 17364 17367 17370 17373 17376 17379 17382 17385 17388 17391 17394 17397 17400 17403 17406 17409 17412 17415 17418 17421 17424 17427 17430 17433 17436 17439 17442 17445 17448 17451 17454 17457 17460 17463 17466 17469 17472 17475 17478 17481 17484 17487 17490 17493 17496 17499 17502 17505 17508 17511 17514 17517 17520 17523 17526 17529 17532 17535 17538 17541 17544 17547 17550 17553 17556 17559 17562 17565 17568 17571 17574 17577 17580 17583 17586 17589 17592 17595 17598 17601 17604 17607 17610 17613 17616 17619 17622 17625 17628 17631 17634 17637 17640 17643 17646 17649 17652 17655 17658 17661 17664 17667 17670 17673 17676 17679 17682 17685 17688 17691 17694 17697 17700 17703 17706 17709 17712 17715 17718 17721 17724 17727 17730 17733 17736 17739 17742 17745 17748 17751 17754 17757 17760 17763 17766 17769 17772 17775 17778 17781 17784 17787 17790 17793 17796 17799 17802 17805 17808 17811 17814 17817 17820 17823 17826 17829 17832 17835 17838 17841 17844 17847 17850 17853 17856 17859 17862 17865 17868 17871 17874 17877 17880 17883 17886 17889 17892 17895 17898 17901 17904 17907 17910 17913 17916 17919 17922 17925 17928 17931 17934 17937 17940 17943 17946 17949 17952 17955 17958 17961 17964 17967 17970 17973 17976 17979 17982 17985 17988 17991 17994 17997 18000 18003 18006 18009 18012 18015 18018 18021 18024 18027 18030 18033 18036 18039 18042 18045 18048 18051 18054 18057 18060 18063 18066 18069 18072 18075 18078 18081 18084 18087 18090 18093 18096 18099 18102 18105 18108 18111 18114 18117 18120 18123 18126 18129 18132 18135 18138 18141 18144 18147 18150 18153 18156 18159 18162 18165 18168 18171 18174 18177 18180 18183 18186 18189 18192 18195 18198 18201 18204 18207 18210 18213 18216 18219 18222 18225 18228 18231 18234 18237 18240 18243 18246 18249 18252 18255 18258 18261 18264 18267 18270 18273 18276 18279 18282 18285 18288 18291 18294 18297 18300 18303 18306 18309 18312 18315 18318 18321 18324 18327 18330 18333 18336 18339 18342 18345 18348 18351 18354 18357 18360 18363 18366 18369 18372 18375 18378 18381 18384 18387 18390 18393 18396 18399 18402 18405 18408 18411 18414 18417 18420 18423 18426 18429 18432 18435 18438 18441 18444 18447 18450 18453 18456 18459 18462 18465 18468 18471 18474 18477 18480 18483 18486 18489 18492 18495 18498 18501 18504 18507 18510 18513 18516 18519 18522 18525 18528 18531 18534 18537 18540 18543 18546 18549 18552 18555 18558 18561 18564 18567 18570 18573 18576 18579 18582 18585 18588 18591 18594 18597 18600 18603 18606 18609 18612 18615 18618 18621 18624 18627 18630 18633 18636 18639 18642 18645 18648 18651 18654 18657 18660 18663 18666 18669 18672 18675 18678 18681 18684 18687 18690 18693 18696 18699 18702 18705 18708 18711 18714 18717 18720 18723 18726 18729 18732 18735 18738 18741 18744 18747 18750 18753 18756 18759 18762 18765 18768 18771 18774 18777 18780 18783 18786 18789 18792 18795 18798 18801 18804 18807 18810 18813 18816 18819 18822 18825 18828 18831 18834 18837 18840 18843 18846 18849 18852 18855 18858 18861 18864 18867 18870 18873 18876 18879 18882 18885 18888 18891 18894 18897 18900 18903 18906 18909 18912 18915 18918 18921 18924 18927 18930 18933 18936 18939 18942 18945 18948 18951 18954 18957 18960 18963 18966 18969 18972 18975 18978 18981 18984 18987 18990 18993 18996 18999 19002 19005 19008 19011 19014 19017 19020 19023 19026 19029 19032 19035 19038 19041 19044 19047 19050 19053 19056 19059 19062 19065 19068 19071 19074 19077 19080 19083 19086 19089 19092 19095 19098 19101 19104 19107 19110 19113 19116 19119 19122 19125 19128 19131 19134 19137 19140 19143 19146 19149 19152 19155 19158 19161 19164 19167 19170 19173 19176 19179 19182 19185 19188 19191 19194 19197 19200 19203 19206 19209 19212 19215 19218 19221 19224 19227 19230 19233 19236 19239 19242 19245 19248 19251 19254 19257 19260 19263 19266 19269 19272 19275 19278 19281 19284 19287 19290 19293 19296 19299 19302 19305 19308 19311 19314 19317 19320 19323 19326 19329 19332 19335 19338 19341 19344 19347 19350 19353 19356 19359 1
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include "OrdenamientosModificados.h"
5
6 void main(){
7     int i, opcion, valor;
8     int *A;
9     printf("Inserta la cantidad de valores de tu arreglo:");
10    scanf("%d", &valor);
11    A = (int*)calloc(valor, sizeof(int));
12    srand(time(NULL));
13    for (i=0; i<valor; i++){
14        A[i] = rand() % 1000;
15    }
16
17    printf("\nArreglo original:");
18    printArray(A, valor);
19    printf("\nSelecciona el tipo de ordenamiento:");
20    scanf("%d", &opcion);
21
22    switch(opcion){
23        case 1:
24            printf("Arreglo ordenado:");
25            printArray(A, valor);
26            quickSort(A, 0, valor-1);
27            printf("\nArreglo ordenado:");
28            printArray(A, valor);
29    }
30
31    printf("\n-----\n");
32    printf("Process exited after 3.385 seconds with return value 1\n");
33    printf("Presione una tecla para continuar . . .");
34    getch();
35}
```

Arreglo original:
3028 13625 13616 27461 26572 5074 28957 8856 10696 6723 27615 11860 8140 17612 15814 24154 20424 24814 8286 24533 25801 5584 25924 10432 31857 19425 20281 15529 26789 14271 373 11956 14725 3160 31037 1250 9121 6223 17848 24723 5737 21153 29183

Arreglo ordenado:
373 1250 3028 3160 4170 5074 5584 5737 5862 6223 6723 6792 8140 8286 8856 9121 10432 10696 11860 11935 11956 12968 13616 13625 14271 14725 15529 15814 17612 17848 19425 20281 20424 21153 21875 24154 24533 24723 24814 25801 25924 26572 26789 27461 27615 28337 28957 29183 31037 31857

Numero de comparaciones:1225
Numero de intercambios:630
Numero de entradas al for:1274

Compilation results...
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\Brain\Documents\EDA II\P2\Díaz Hernández Marcos G9 P2 V1\TercerEjercicio.exe
- Output Size: 132.9091796875 KiB
- Compilation Time: 0.64s

BubbleSort - 50

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include "OrdenamientosModificados.h"
5
6 void main(){
7     int i, opcion, valor;
8     int *A;
9     printf("Inserta la cantidad de valores de tu arreglo:");
10    scanf("%d", &valor);
11    A = (int*)calloc(valor, sizeof(int));
12    srand(time(NULL));
13    for (i=0; i<valor; i++){
14        A[i] = rand() % 10000000;
15    }
16
17    printf("\nArreglo original:");
18    printArray(A, valor);
19    printf("\nSelecciona el tipo de ordenamiento:");
20    scanf("%d", &opcion);
21
22    switch(opcion){
23        case 1:
24            printf("Arreglo ordenado:");
25            printArray(A, valor);
26            quickSort(A, 0, valor-1);
27            printf("\nArreglo ordenado:");
28            printArray(A, valor);
29    }
30
31    printf("\n-----\n");
32    printf("Process exited after 3.466 seconds with return value 1\n");
33    printf("Presione una tecla para continuar . . .");
34    getch();
35}
```

Arreglo original:
3116 8920 4428 21810 10154 12792 14689 28123 30525 4565 23621 13049 16065 5611 25178 763 23614 30775 2494 53 5341 3218 153 23100 5788 1620 1687 7612 15862 25592 30415 9447 18230 31027 31858 15167 10410 6309 1611 2556 23140 2032 2668 19128 14121 31502 8228 31039 1220 3511 24480 5480 23590 20195 25708 21390 19682 1917 96 10513 13583 15011 10541 11246 5116 14483 27514 3054 28041 29832 15883 24623 28458 19891 3225 2452 1895 1529 7939 14824 31036 16995 27635 20710 27415 14236 24116 28072 32387 29565 11873 16106

Arreglo ordenado:
153 763 1229 1620 1687 2452 2487 2668 2932 3054 3116 3218 3225 3511 4339 4428 4565 5116 5341 5489 5611 57 40 7612 7939 8228 8920 9447 10154 10410 10513 10541 11246 11873 12556 12792 13049 13583 13869 14121 14236 689 14824 15011 15167 15862 15883 16065 16106 16114 16995 18230 18953 19128 19177 19583 19682 19891 20195 529 21810 22996 23100 23140 23599 23614 23621 24116 24489 24623 24947 25178 25592 25708 27415 27514 27635 123 28458 29565 29832 29997 30353 30415 30525 30775 31027 31036 31039 31502 31858 32387

Numero de comparaciones:4950
Numero de intercambios:2105
Numero de entradas al for:5049

Compilation results...
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\Brain\Documents\EDA II\P2\Díaz Hernández Marcos G9 P2 V1\TercerEjercicio.exe
- Output Size: 132.9091796875 KiB
- Compilation Time: 0.53s

BubbleSort - 100


```
C:\Users\Brain\Documents\EDA\INP2\Diaz Hernández Marcos G9 P2 V1\TercerEjercicio.c - [Executing] - Dev-C++ 5.11
C:\Users\Brain\Documents\EDA\INP2\Diaz Hernández Marcos G9 P2 V1\TercerEjercicio.exe
12471 24833 11553 12723 10583 12753 3043 7703 21870 14340 16162 791 14718 9446 13961 18172 17084 4261 18974 16395 4967 21870
5943 2349 6061 26969 31550 30081 26829 21212 19019 32474 7554 9662 21730
Numero de comparaciones:124750
Proy Numero de intercambios:64345
Numero de entradas al for:125249
Arreglo ordenado:
37 244 267 375 412 791 875 924 974 907 1079 1179 1301 1306 1312 1377 1395 1449 1453 1512 1596 1619 1656 1720 1807 1820 1
927 1949 1951 2002 2242 2349 2437 2484 2515 2608 2669 2710 2767 2791 2875 2925 2927 2930 2978 3007 3043 3107 3157 3157 3
191 3236 3263 3299 3426 3518 3547 3716 3717 3832 3943 4065 4175 4197 4261 4303 4509 4578 4617 4715 4967 4980 5074 5078 5
161 5175 5261 5314 5355 5471 5503 5528 5630 5682 5866 5902 5953 5959 5962 6061 6099 6099 6135 6412 6414 6483 6532 6545 6
570 6601 6641 6645 6679 6692 6945 6945 7309 7336 7422 7422 7436 7448 7528 7541 7554 7603 7675 7703 7729 7753 7896 7942 8
017 8138 8230 8234 8277 8528 8589 8704 8732 8735 8829 8830 8865 8888 8889 8924 8990 9026 9078 9169 9311 9425 9438 9445 9
446 9467 9470 9498 9638 9662 9721 9775 9844 9945 10028 10036 10126 10151 10316 10366 10428 10445 10458 10486 10540 10564 1
10573 10583 10615 10659 10669 10737 10749 10768 10780 11031 11067 11115 11134 11140 11191 11344 11553 11562 11888 11890
12026 12042 12106 12114 12174 12186 12221 12298 12470 12471 12501 12579 12584 12674 12723 12734 12753 12758 12942 12962
13041 13176 13212 13280 13324 13357 13393 13450 13629 13649 13664 13700 13772 13818 13823 13863 13961 13967 14027 14219
14340 14373 14400 14522 14525 14684 14702 14718 14727 14741 14765 14875 14971 15031 15041 15248 15254 15257 15259 15376
15441 15478 15558 15560 15610 15614 15823 15854 15864 15898 15989 16011 16025 16065 16121 16150 16152 16162 16258 16351
16352 16367 16395 16512 16799 16844 16868 16909 16965 17024 17084 17093 17112 17165 17204 17336 17378 17418 17627 17708
17783 17857 17862 17965 18007 18022 18104 18172 18445 18503 18663 18763 18777 18857 18860 18872 18874 18974 18979 19019
19168 19243 19256 19310 19320 19746 19833 19843 19857 19882 19927 19951 20089 20097 20192 20622 20641 20656 20705 20727
20919 21167 21212 21450 21489 21513 21559 21567 21582 21607 21623 21730 21777 21870 21884 21912 21940 22003 22037 22077
22085 22181 22130 22133 22312 22369 22428 22579 22599 22747 22756 22814 22847 22848 22946 22993 23087 23099 23315 23441
23484 23564 23677 23693 23755 23769 23963 24045 24093 24133 24196 24212 24241 24375 24383 24689 24693 24734 24833 24869
24893 24900 25034 25050 25120 25199 25226 25248 25303 25328 25386 25482 25648 25655 25681 25744 25749 25873 25943 26063
26141 26160 26253 26269 26326 26473 26635 26746 26829 26964 26969 27101 27172 27266 27313 27411 27468 27595 27621 27651
27751 27890 27904 28117 28383 28426 28438 28518 28535 28649 28802 28863 28880 28923 29003 29006 29145 29157 29172 29225
29275 29299 29558 29651 29717 29722 29733 29915 29931 29947 30030 30035 30048 30050 30055 30081 30114 30284 30354 30360
30417 30837 31018 31198 31208 31210 31346 31465 31482 31521 31550 31648 31891 31948 32024 32065 32087 32092 32096 32107
32140 32220 32240 32279 32345 32353 32433 32474 32518 32634 32648 32708
-----
Process exited after 3.732 seconds with return value 1
Presione una tecla para continuar . . .
Shorten compiler paths
- Output Size: 132.9091796875 KiB
- Compilation Time: 0.72s
Line: 1 Col: 1 Sel: 0 Lines: 53 Length: 1107 Insertar Done parsing in 0.015 seconds
Escribe aquí para buscar
```

BubbleSort – 500

```
20129 1748 30880 22873 4135 11874 23210 21925 23022 15340 10406 234 28213 94
19865 5145 10245 28780 7009 7154 11297 17249 28135 1654 31130 6696 31034 196
3940 8011 4929 3880 18699 13262 1712 32570 510 13500 10966 7995 30950 22150
62 14544 6591 2237 25239 3069 10400 3879 20847 26479 5012 30376 16654 18536
32 4132 21531
Numero de comparaciones:499500
Numero de intercambios:252262
Numero de entradas al for:500499
Arreglo ordenado:
16 56 130 154 174 197 234 235 260 274 283 291 314 391 423 463 495 509 510 567
4 834 842 845 910 916 921 939 947 1064 1081 1083 1088 1154 1194 1200 1275 127
502 1521 1640 1654 1712 1748 1848 1858 1882 1924 1950 1975 2066 2101 2174 21
408 2591 2679 2700 2700 2759 2763 2771 2819 2826 2870 2970 2997 3003 3069 31
400 3467 3506 3518 3527 3566 3613 3638 3727 3744 3751 3817 3879 3880 3910 39
132 4136 4140 4143 4153 4173 4191 4221 4233 4235 4246 4248 4325 4340 4446 44
733 4807 4800 4848 4876 4929 4956 5004 5012 5029 5104 5134 5145 5165 5175 52
677 5687 5818 5848 5900 5960 6032 6062 6168 6215 6286 6322 6334 6343 6364 63
612 6623 6637 6694 6696 6704 6704 6769 6849 6855 6860 6963 6999 7009 7035 70
220 7225 7227 7240 7247 7254 7359 7363 7383 7386 7410 7528 7573 7576 7578 76
890 7903 7918 7953 7954 7980 7995 8005 8010 8011 8023 8025 8069 8115 8258 82
569 8612 8694 8723 8762 8766 8804 8805 8836 8858 8900 8978 8997 9013 9088 91
498 9507 9521 9579 9602 9719 9786 9798 9911 10086 10163 10178 10245 10247 10
7 10468 10482 10491 10552 10603 10614 10652 10703 10742 10753 10773 10816 10
9 11010 11059 11085 11093 11192 11196 11254 11264 11273 11297 11403 11426 11
1 11640 11713 11714 11805 11833 11845 11845 11874 11880 11945 12075 12085 12
9 12429 12430 12445 12454 12458 12486 12520 12563 12595 12625 12626 12740 12
8 13074 13108 13134 13151 13164 13193 13211 13251 13262 13264 13304 13332 13
32747 32766
-----
Process exited after 3.49 seconds with return value 1
Presione una tecla para continuar . . .
```

BubbleSort – 1000

C:\Users\Brain\Documents\EDA II\P2\Díaz Hernández Marcos G9 P2 V1\TercerEjercicio.c - [Executing] - Dev-C++ 5.11

Archivo Edición Buscar Ver Proyecto Ejecutar Herramientas AStyle Ventana Ayuda

Proyecto Clases(Funciones) OrdenamientosModificados.h TercerEjercicio.c

```
2223 13396 13593 13908 15546 16899 18310 18959 20436 20676 22183 22900 23546 23795 23898 23920 24971 26063 26547 28461 29
574 29632 30241 31254 31282 32031 32561
case 1:
printf("Arreglo Iteracion HS:
printArray(A,1519 2027 1848 2628 2905 2945 3081 3670 3744 3925 4345 5153 5471 6215 6291 7092 7766 8598 9416 9731 10551 11375 11837 12
int i,j; 223 13396 13593 13908 15546 16899 18310 18959 20436 20676 22183 22900 23546 23795 23898 23920 24971 26063 26547 28461 29
quickSort(A,574 29632 30241 31254 31282 32031 32561
printf("\nArreglo Iteracion HS:
printArray(A,223 13396 13593 13908 15546 16899 18310 18959 20436 20676 22183 22900 23546 23795 23898 23920 24971 26063 26547 28461 29
break; 574 29632 30241 31254 31282 32031 32561
case 2:
printf("Arreglo Iteracion HS:
printArray(A,1519 2027 1848 2628 2905 2945 3081 3670 3744 3925 4345 5153 5471 6215 6291 7092 7766 8598 9416 9731 10551 11375 11837 12
bubbleSort(A,574 29632 30241 31254 31282 32031 32561
printf("\nArreglo Iteracion HS:
printArray(A,223 13396 13593 13908 15546 16899 18310 18959 20436 20676 22183 22900 23546 23795 23898 23920 24971 26063 26547 28461 29
break; 574 29632 30241 31254 31282 32031 32561
case 3:
printf("Arreglo Numero de comparaciones:789
printArray(A,223 13396 13593 13908 15546 16899 18310 18959 20436 20676 22183 22900 23546 23795 23898 23920 24971 26063 26547 28461 29
HeapSort(A,574 29632 30241 31254 31282 32031 32561
printf("\nArreglo Numero de intercambios:238
printArray(A,223 13396 13593 13908 15546 16899 18310 18959 20436 20676 22183 22900 23546 23795 23898 23920 24971 26063 26547 28461 29
break; 574 29632 30241 31254 31282 32031 32561
printf("Arreglo ordenado:
1519 1848 2027 2628 2905 2945 3081 3670 3744 3925 4345 5153 5471 6215 6291 7092 7766 8598 9416 9731 10551 11375 11837 12
223 13396 13593 13908 15546 16899 18310 18959 20436 20676 22183 22900 23546 23795 23898 23920 24971 26063 26547 28461 29
default:
574 29632 30241 31254 31282 32031 32561
opcion=4;
```

Compilador Recursos Registro de Compilación Derivados

Cancelar Compilación

Shorten compiler paths

Compilation results...

Process exited after 3.575 seconds with return value 1

Presione una tecla para continuar...

Errors: 0

Warnings: 0

Output Filename: C:\Users\Brain\Documents\EDA II\P2\Díaz Hernández Marcos G9 P2 V1\TercerEjercicio.exe

Output Size: 132.9091796875 KiB

Compilation Time: 0.55s

Line: 1 Col: 1 Sel: 0 Lines: 53 Length: 1107 Insertar Done parsing in 0.016 seconds

Escribe aquí para buscar

HeapSort - 50

C:\Users\Brain\Documents\EDA II\P2\Díaz Hernández Marcos G9 P2 V1\TercerEjercicio.c - [Executing] - Dev-C++ 5.11

Archivo Edición Buscar Ver Proyecto Ejecutar Herramientas AStyle Ventana Ayuda

Proyecto Clases(Funciones) OrdenamientosModificados.h TercerEjercicio.c

```
26 45 733 1059 1683 1865 1934 2069 2283 2840 3385 3882 3990 4731 5222 5251 5484 5649 5808 6161 6168 6257 7422 7807 8006
8422 8492 8691 8792 9310 9320 9943 9996 10340 10616 10630 10645 11150 11604 12523 12590 12637 14302 14328 14655 14755 15
106 15953 16011 16400 16476 17481 17566 17661 18099 18160 19507 19715 20009 20260 20825 20866 21318 21341 22397 23344 23
372 23598 23599 23840 24445 24619 24695 24927 24955 25245 25251 25422 25913 26113 26467 26526 26954 27304 27594 28391 28
750 28809 29309 29364 29426 29895 30300 30314 30353 30900 31165 31982 32150 32294
Iteracion HS:
26 45 733 1059 1683 1865 1934 2069 2283 2840 3385 3882 3990 4731 5222 5251 5484 5649 5808 6161 6168 6257 7422 7807 8006
8422 8492 8691 8792 9310 9320 9943 9996 10340 10616 10630 10645 11150 11604 12523 12590 12637 14302 14328 14655 14755 15
106 15953 16011 16400 16476 17481 17566 17661 18099 18160 19507 19715 20009 20260 20825 20866 21318 21341 22397 23344 23
372 23598 23599 23840 24445 24619 24695 24927 24955 25245 25251 25422 25913 26113 26467 26526 26954 27304 27594 28391 28
750 28809 29309 29364 29426 29895 30300 30314 30353 30900 31165 31982 32150 32294
Numero de comparaciones:1923
Numero de Intercambios:591
Numero de entradas al for:149
Arreglo ordenado:
26 45 733 1059 1683 1865 1934 2069 2283 2840 3385 3882 3990 4731 5222 5251 5484 5649 5808 6161 6168 6257 7422 7807 8006
8422 8492 8691 8792 9310 9320 9943 9996 10340 10616 10630 10645 11150 11604 12523 12590 12637 14302 14328 14655 14755 15
106 15953 16011 16400 16476 17481 17566 17661 18099 18160 19507 19715 20009 20260 20825 20866 21318 21341 22397 23344 23
372 23598 23599 23840 24445 24619 24695 24927 24955 25245 25251 25422 25913 26113 26467 26526 26954 27304 27594 28391 28
750 28809 29309 29364 29426 29895 30300 30314 30353 30900 31165 31982 32150 32294
```

Process exited after 0.698 seconds with return value 1

Presione una tecla para continuar...

Cancelar Compilación

Shorten compiler paths

Compilation results...

Errors: 0

Warnings: 0

Output Filename: C:\Users\Brain\Documents\EDA II\P2\Díaz Hernández Marcos G9 P2 V1\TercerEjercicio.exe

Output Size: 132.9091796875 KiB

Compilation Time: 0.53s

Line: 1 Col: 1 Sel: 0 Lines: 53 Length: 1107 Insertar Done parsing in 0.016 seconds

Escribe aquí para buscar

HeapSort - 100

C:\Users\Brain\Documents\EDA\INP2\Díaz Hernández Marcos G9 P2 V1\TercerEjercicio.c - [Executing] - Dev-C++ 5.11

Archivo Edición Buscar Ver Proyecto Ejecutar Herramientas AStyle Ventana Ayuda

ITM-GCC 4.9.2 64-bit Release

Proyecto Clases(Func) TercerEjercicio.c

```
2116 32181 32196 32286 32290 32432 32452 32518 32678 32746
Numero de comparaciones:12933
Numero de Intercambios:4061
Numero de entradas al for:749

case 1:
printf("Arreglo ordenado:
int i,j;
quickSort(A,
printf("\nArreglo ordenado:
break;
471 4512 4628 4628 4642 4835 4863 4927 4063 4960 5160 5200 5220 5317 5326 5380 5562 5615 5615 5696 6001 6128 6206
210 6243 6280 6310 6347 6362 6458 6514 6550 6593 6608 6697 6730 6805 6833 6883 6922 6935 7025 7226 7515 7560 7643
696 7708 7720 7762 7845 7874 7900 7914 7060 7983 8026 8102 8115 8235 8278 8326 8326 8408 8413 8430 8658 8664 8696
846 8892 8900 8937 8957 9034 9125 9181 9259 9266 9282 9294 9309 9326 9486 9551 9671 9765 9784 9790 9851 9904 9969
982 10227 10255 10256 10267 10269 10272 10302 10324 10408 10595 10608 10619 10621 10667 10814 10880 10904 10982 11
95 11026 11098 11137 11146 11262 11292 11345 11490 11513 11523 11618 11625 11666 11670 11800 11814 11890 11898 121
12332 12373 12439 12465 12530 12581 12673 12750 12842 12892 13017 13027 13081 13103 13148 13244 13305 13305 134
13449 13456 13526 13587 13668 13733 13804 13944 13993 14201 14302 14376 14386 14797 14873 14961 15095 15220 15220
15255 15329 15361 15361 15365 15424 15478 15505 15515 15578 15606 15636 15695 15697 15715 15825 15852 15860 15878
5876 15936 15969 16032 16091 16334 16431 16474 16544 16648 16714 16992 16998 17148 17285 17354 17429 17507 17654 1
677 17762 17805 17980 18217 18296 18445 18447 18448 18454 18576 18662 18785 18805 18894 18899 18939 19014 19024 19
75 19086 19332 19376 19402 19446 19495 19503 19566 19606 19648 19866 20000 20015 20277 20324 20336 20447 20469 205
20584 20651 20666 20670 20672 20979 21093 21105 21145 21220 21228 21253 21333 21419 21553 21741 21814 21851 2190
21993 22172 22207 22236 22257 22264 22304 22332 22382 22505 22565 22646 22691 22712 22797 22807 22841 23110 23124
23127 23129 23150 23167 23254 23319 23358 23642 23658 23663 23687 23795 23803 23910 23929 24021 24021 24253 24365
4483 24494 24513 24622 24645 24803 24996 25005 25054 25142 25165 25295 25329 25350 25410 25486 25508 25596 25629 2
737 25762 25765 25785 25820 25983 26007 26063 26112 26182 26188 26391 26434 26444 26506 26540 26652 26682 26700 26
32 26752 26772 26826 26829 26877 26932 26944 27089 27242 27283 27303 27328 27414 27602 27636 27743 27836 27879 278
228051 28058 28061 28128 28190 28216 28228 28245 28259 28327 28396 28449 28494 28516 28560 28944 28954 28994 2904
29173 29228 29508 29627 29637 29699 29914 30003 30029 30052 30161 30267 30304 30577 30585 30589 30590 30604 30636
30647 30649 30802 30844 30853 30869 30876 30942 31078 31161 31177 31279 31570 31583 31683 31746 31811 31840 32113
2116 32181 32196 32286 32290 32432 32452 32518 32678 32746

-----
Process exited after 196 seconds with return value 1
Presione una tecla para continuar . . .
```

Compilador Recursos Registro de Compilación

Cancelar Compilación

Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\Brain\Documents\EDA\INP2\Díaz Hernández Marcos G9 P2 V1\TercerEjercicio.exe
- Output Size: 132.9091796 KB
- Compilation Time: 0.55s

Line: 1 Col: 1 Sek: 0 Lines: 53 Length: 1107 Insertar Done parsing in 0.016 seconds

Escribe aquí para buscar

12:57 p.m.
11/10/2020

HeapSort – 500

C:\Users\Brain\Documents\EDA\INP2\Díaz Hernández Marcos G9 P2 V1\TercerEjercicio.exe

```
Numero de comparaciones:28743
Numero de Intercambios:9081
Numero de entradas al for:1499

Arreglo ordenado:
8 17 30 42 74 122 233 252 285 306 317 318 320 431 471 532 592 627 679 718 791 862 899 924 924 943 956 981 1004 1005 1021 1162 1195 1199 1211 1255 1256 1350 1357 1381 14
37 1465 1488 1490 1501 1517 1555 1564 1577 1585 1587 1601 1605 1620 1621 1627 1647 1650 1770 1779 1784 1792 1800 1822 1866 1946 1957 2062 2064 2087 2138 2201 2242 2251
2308 2330 2339 2403 2416 2421 2439 2458 2481 2520 2564 2604 2823 2934 3013 3032 3041 3132 3190 3217 3221 3279 3297 3329 3331 3374 3479 3498 3521 3529 3562 3566 3596 360
8 3676 3683 3706 3708 3736 3738 3812 3870 3949 3963 4028 4032 4076 4090 4093 4127 4143 4147 4169 4211 4215 4296 4362 4364 4436 4531 4535 4540 4558 4580 4593 4601 4671 4
683 4735 4737 4779 4780 4796 4808 4824 4862 4881 4929 4958 5002 5004 5028 5050 5056 5058 5154 5174 5187 5292 5296 5316 5339 5345 5371 5411 5414 5479 5496 5536 5554 5566
5586 5709 5721 5786 5843 5852 5871 5879 5910 5954 5963 5970 5974 6034 6064 6084 6110 6121 6127 6142 6203 6248 6264 6274 6278 6335 6339 6367 6372 6451 6484 6500 6565 66
07 6729 6749 6777 6786 6802 6817 6992 7084 7033 7059 7067 7119 7140 7141 7203 7226 7241 7252 7279 7333 7343 7367 7386 7389 7427 7501 7531 7556 7563 7619 7718 7749 7752
7761 7765 7822 7830 7834 7868 7929 7930 7975 7983 8021 8056 8061 8110 8121 8121 8121 8151 8257 8261 8306 8324 8335 8350 8357 8386 8441 8444 8445 8471 8484 8495 8499 853
1 8554 8565 8569 8571 8581 8679 8682 8709 8802 8813 8833 8852 8881 8948 9049 9076 9086 9105 9118 9141 9153 9206 9370 9381 9385 9406 9411 9427 9469 9472 9472 9494 9524 9
547 9564 9622 9673 9698 9713 9717 9773 9822 9833 9854 9861 9946 9951 9952 10115 10155 10159 10223 10246 10380 10380 10412 10417 10425 10440 10477 10484 10508 1053
4 10534 10577 10611 10650 10682 10687 10721 10809 10825 10860 10999 11037 11064 11093 11116 11125 11132 11160 11199 11202 11246 11273 11299 11341 11386 11412 11418 1143
6 11528 11538 11548 11576 11582 11593 11620 11628 11670 11682 11728 11734 11760 11785 11987 11924 12107 12201 12210 12237 12286 12322 12426 12440 12445 12499 12508 1252
7 12663 12668 12669 12677 12761 12766 12852 12871 12873 12912 12946 12967 12997 12997 13016 13044 13096 13153 13203 13276 13293 13322 13369 13378 13400 13439 13468 1351
5 13571 13702 13722 13741 13755 13785 13796 13840 14020 14026 14039 14054 14057 14151 14152 14158 14170 14179 14201 14253 14257 14274 14287 14292 14301 14325 14358 1440
0 14403 14446 14454 14507 14578 14625 14664 14718 14746 14763 14790 14816 14859 14874 14896 14917 14923 14987 14987 15001 15042 15055 15082 15132 15142 15202 15245 1525
4 15254 15271 15325 15343 15377 15449 15457 15643 15686 15687 15724 15732 15756 15773 15801 15838 15846 15917 15932 15932 15939 15947 15967 16005 16042 16113 16156 1624
6 16247 16316 16371 16377 16385 16399 16412 16423 16435 16451 16468 16486 16555 16564 16579 16656 16662 16666 16669 16684 16770 16801 16847 16864 16905 16938 16987 1703
1 17044 17053 17071 17088 17091 17136 17143 17351 17359 17372 17442 17465 17482 17518 17553 17599 17625 17631 17651 17682 17749 17750 17807 17864 17903 17910 17984 1799
9 18027 18125 18158 18161 18170 18196 18236 18283 18325 18338 18398 18402 18489 18578 18579 18621 18664 18707 18729 18758 18759 18780 18781 18912 18943 18946 18958 1896
1 19037 19085 19095 19097 19113 19125 19134 19139 19153 19165 19249 19252 19260 19264 19272 19298 19300 19356 19367 19374 19375 19427 19440 19529 19530 19549 19605 1970
4 19736 19769 19825 19831 19845 19881 19898 19938 19997 20019 20026 20047 20053 20229 20238 20246 20261 20277 20320 20374 20389 20400 20457 20457 20563 20617 20646 2066
20676 20685 20687 20721 20751 20763 20764 20780 20801 20817 20863 20871 21048 21073 21078 21141 21145 21173 21183 21222 21257 21278 21297 21321 21322 21339 21354 2136
7 21438 21460 21474 21564 21586 21633 21700 21719 21726 21734 21742 21744 21748 21757 21771 21777 21809 21814 21825 21850 21881 21894 21902 21961 21980 21986 21999 2204
7 22052 22058 22128 22141 22228 22264 22282 22295 22330 22394 22397 22398 22410 22459 22495 22524 22532 22721 22751 22759 22800 22838 22857 22865 22890 22967 23022 2302
4 23027 23063 23070 23111 23162 23198 23199 23257 23257 23275 23282 23439 23441 23529 23676 23679 23716 23719 23763 23790 23825 23829 23830 23961 23977 24061 24104 2414
2 24144 24163 24189 24276 24298 24303 24308 24404 24454 24462 24468 24475 24529 24580 24586 24590 24691 24859 24941 24942 24944 25033 25048 25059 25066 25107 25136 2527
6 25328 25355 25382 25391 25461 25489 25522 25536 25629 25633 25646 25657 25695 25742 25762 25791 25858 25859 25868 25892 25936 25999 26019 26025 26043 26141 26159 2619
2 26204 26219 26274 26340 26359 26372 26421 26422 26470 26482 26483 26584 26616 26713 26750 26778 26808 27003 27022 27035 27062 27122 27148 27178 27195 27218 27228 2722
9 27272 27280 27298 27368 27496 27501 27515 27583 27587 27590 27670 27677 27737 27748 27763 27772 27785 27816 27837 27855 27910 27914 27935 28051 28057 28151 28152 2822
2 28227 28247 28338 28346 28399 28408 28484 28532 28562 28597 28607 28619 28633 28688 28764 28765 28773 28810 28871 28911 28923 28995 29061 29105 29149 29360 29408 2949
2 29504 29514 29518 29554 29579 29605 29615 29626 29670 29675 29819 29834 29843 29872 29940 29993 30016 30099 30113 30170 30184 30190 30191 30212 30225 30234 30260 3042
2 30439 30479 30525 30531 30550 30617 30647 30671 30685 30701 30705 30715 30740 30745 30764 30846 30847 30849 30852 30854 30909 30911 31073 31093 31113 31131 31157 31162
0 31178 31182 31201 31224 31255 31285 31337 31376 31407 31456 31497 31534 31537 31555 31776 31795 31800 31816 31958 32007 32045 32050 32052 32088 32129 32187 32238 3228
4 32314 32328 32355 32384 32392 32447 32465 32533 32535 32558 32609 32645 32681 32722 32750 32757

-----
Process exited after 675.1 seconds with return value 1
Presione una tecla para continuar . . .
```

Escribe aquí para buscar

01:11 p.m.
11/10/2020

HeapSort - 1000

- Ejercicio 4:

El último ejercicio consistió en la implementación del QuickSort en Java, y pues hacer todo lo que conlleva el poder crear un algoritmo en Java.

- Dificultades en el código

La parte que mas se me dificulto fue la clase QuickSort, debido a que si bien la parte de Partition no tenia problemas, en la parte de que tenia que igualarse al pivote no sabía como, tal vez suene burdo pero pensé que tendría que ser de otra forma, y cuando lo intente poner de la forma común en c resulto que si funciona así (Imagen 1).

```
public void sort(int[] arr,int low, int hi
    if(low<high){
        int pi = partition(arr,low,high);
        sort(arr,low,pi-1); //La recursi
        sort(arr,pi+1,high); //forma difer
```

Imagen 1

Dentro del programa hice los respectivos comentarios pero algo que me causa duda o curiosidad es que se tenga que poner un el acceso y también que las clases con el acceso public, tengan que ser iguales al nombre del programa, porque sino causa problema en la compilación, eso no lo vie en este caso pero en al hacerlo con comandos, si pasa eso.

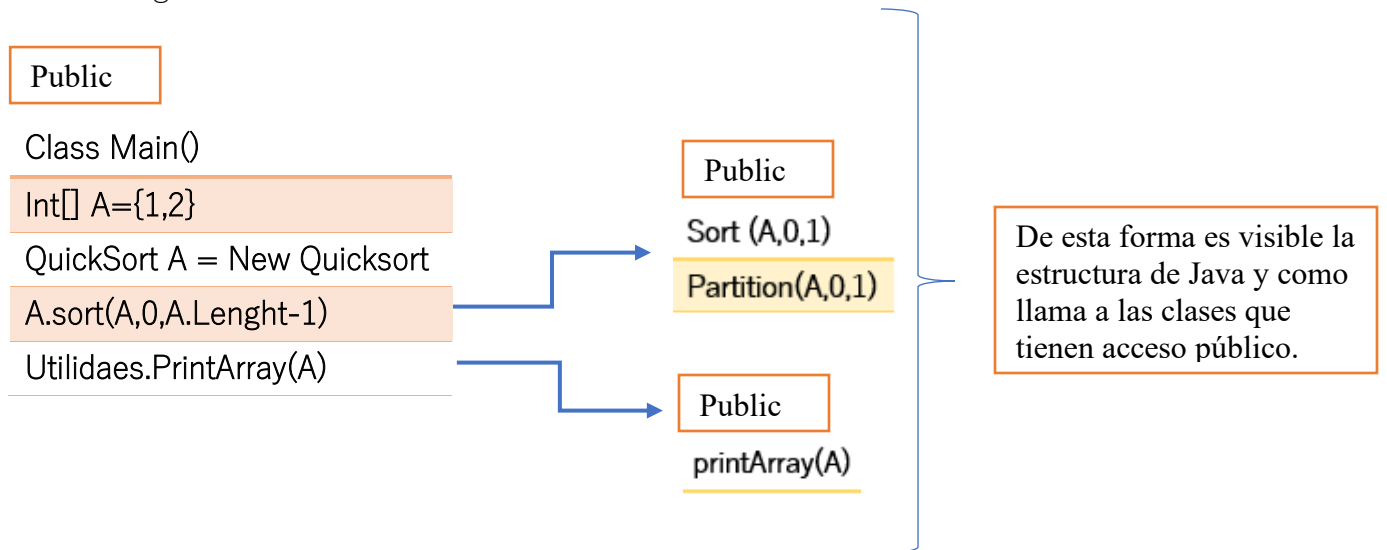
Otra cosa que se me hizo bastante curiosa fue que al tener una clase tienes métodos de esta y puede definir objetos que heredan los atributos y los métodos, y de la misma forma existen métodos que son pertenecientes a la clase, que son estáticos, entonces por esta condición decidí colocar los métodos de QuickSort como de instancia en caso de que se quisieran tener mas arreglos y arreglarlos, para que no se creara un problema por la modificación de los métodos.

En cuanto a la creación de los objetos y el uso de los métodos pues creía que para tener un objeto era necesario definir atributos pero no, con tener métodos basta, y para utilizar los métodos es necesario acceder a ellos por medio del objeto así que es bastante interesante la forma de trabajo en Java (Imagen 2), y por ultimo algo importante que vi es que se necesita tener los métodos en publico de lo contrario otras clases no pueden acceder.

```
public static void main(String[] args){
    int[] arr={87,4,32,15,8,12,10,30,22};
    Quicksort quicksort = new Quicksort();
    quicksort.sort(arr,0,arr.length-1);
    Utilidades.printArray(arr);
}
```

Imagen 2

- Diagrama de funcionamiento:



- Relación con la teoría:

De primera mano pues programar en Java es algo aplicado a EDA II, al igual que saber implementar los algoritmos en cualquier lenguaje, ya que eso permite ver la diferencia entre los lenguajes y los paradigmas que se manejan en cada lenguaje, en POO no llevo muchos programas por lo que se vuelve algo confuso a veces pero poco a poco voy entendiendo la forma de trabajo y mucho mejor si tengo que programar algo.

- Evidencia de implementación

```

public class Practica2DiazMarcos {
    public static void main(String[] args) {
        int[] arr={87,4,32,15,8,12,10,30,22};
        int[] arr1={1,345,657,234,678,23};
        Quicksort quicksort = new Quicksort(); //Se crea el objeto de tipo Quicksort
        quicksort.sort(arr,0,arr.length-1); //Se realiza el metodo sort, y se envian los datos.
        Utilidades.printArray(arr);
        quicksort.sort(arr1, 0,arr1.length-1);
        Utilidades.printArray(arr1);
    }
}

```

Output - Practica2[DiazMarcos] (run)

```

run:
4 8 10 12 15 22 30 32 87
1 23 234 345 657 678
BUILD SUCCESSFUL (total time: 0 seconds)

```

Conclusiones.

En esta práctica se llevo a cabo el análisis de los algoritmos de ordenamiento: QuickSort, HeapSort, BubbleSort, los cuales se ejecutaron, se analizo su complejidad, se implementaron en otros lenguajes de programación por medio de otro paradigma, además de ser modificados con la intención de poder comprender las diferencias entre las versiones que existen de algunos de ellos, hacerlos mas eficientes, o simplemente comprender lo que hacían.

Por último el objetivo de la práctica se cumplió, y siento que a la vez pasada mejore el análisis de los algoritmos, a la vez que me metí más al análisis de algunos de ellos ya que antes de realizar los cambios es mucho mejor saber lo que hace en ese momento el algoritmo y en base a eso buscar los puntos que se pueden cambiar o arreglar. Sin mas que decir me llevo conocimiento que sirve para concretar lo visto en las clases.