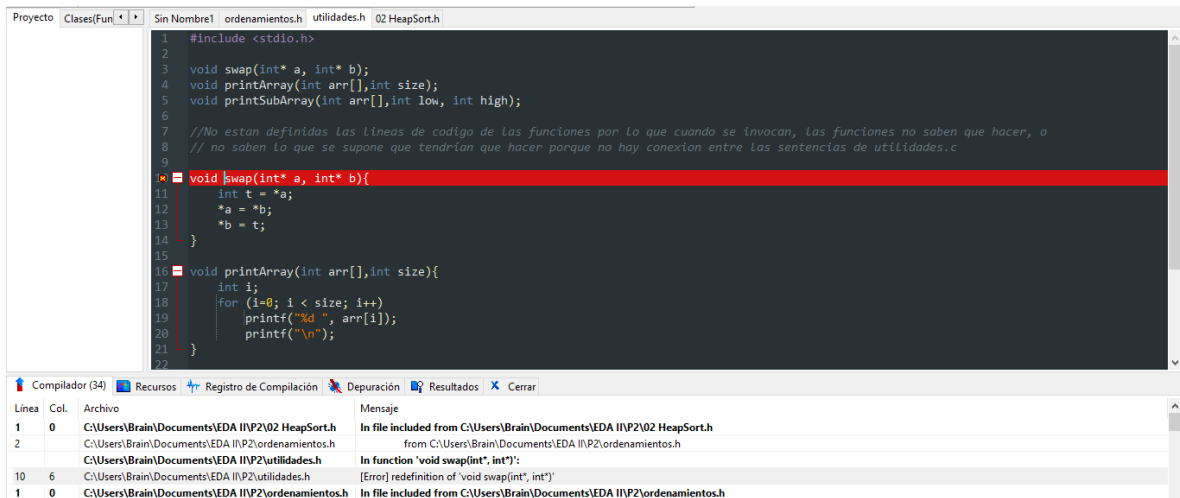


## Objetivo:

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

## Avances:

- Ejercicio 1:

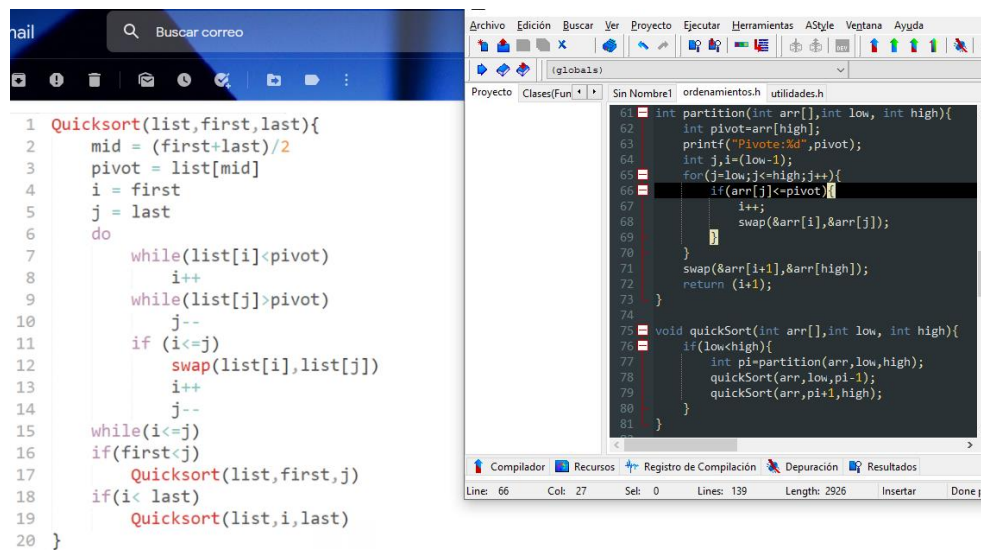


```
1 #include <stdio.h>
2
3 void swap(int* a, int* b);
4 void printArray(int arr[],int size);
5 void printSubArray(int arr[],int low, int high);
6
7 //No estan definidas las lineas de codigo de las funciones por lo que cuando se invocan, las funciones no saben que hacer, o
8 // no saben lo que se supone que tendrian que hacer porque no hay conexion entre las sentencias de utilidades.c
9
10 void swap(int* a, int* b){
11     int t = *a;
12     *a = *b;
13     *b = t;
14 }
15
16 void printArray(int arr[],int size){
17     int i;
18     for (i=0; i < size; i++){
19         printf("%d ", arr[i]);
20     }
21     printf("\n");
22 }
```

Compilador (34) Recursos Registro de Compilación Depuración Resultados X Cerrar

Línea	Col.	Archivo	Mensaje
1	0	C:\Users\Brain\Documents\EDA I\VP2\02 HeapSort.h	In file included from C:\Users\Brain\Documents\EDA I\VP2\02 HeapSort.h
2		C:\Users\Brain\Documents\EDA I\VP2\ordenamientos.h	from C:\Users\Brain\Documents\EDA I\VP2\ordenamientos.h
		C:\Users\Brain\Documents\EDA I\VP2\utilidades.h	In function 'void swap(int*, int*)':
10	6	C:\Users\Brain\Documents\EDA I\VP2\utilidades.h	[Error] redefinition of 'void swap(int*, int*)'
1	0	C:\Users\Brain\Documents\EDA I\VP2\ordenamientos.h	In file included from C:\Users\Brain\Documents\EDA I\VP2\ordenamientos.h

Este fue el primer problema que encontré porque “02 HeapSort” utilizaba las funciones de utilidades por lo que se solicitaban estas funciones en dos bibliotecas y eso creaba una doble solicitud lo que generaba un error.



```
1 Quicksort(list,first,last){
2     mid = (first+last)/2;
3     pivot = list[mid];
4     i = first;
5     j = last;
6     do
7     while(list[i]<pivot)
8         i++;
9     while(list[j]>pivot)
10        j--;
11    if (i<=j)
12        swap(list[i],list[j]);
13        i++;
14        j--;
15    while(i<=j)
16    if(first<j)
17        QuickSort(list,first,j);
18    if(i< last)
19        QuickSort(list,i,last);
20 }
```

```
61 int partition(int arr[],int low, int high){
62     int pivot=arr[high];
63     printf("Pivot:%d",pivot);
64     int j,i=(low-1);
65     for(j=low;j<=high;j++){
66         if(arr[j]<=pivot){
67             i++;
68             swap(&arr[i],&arr[j]);
69         }
70     }
71     swap(&arr[i+1],&arr[high]);
72     return (i+1);
73 }
74
75 void quickSort(int arr[],int low, int high){
76     if(low<high){
77         int pi=partition(arr,low,high);
78         quickSort(arr,low,pi-1);
79         quickSort(arr,pi+1,high);
80     }
81 }
```

Compilador Recursos Registro de Compilación Depuración Resultados

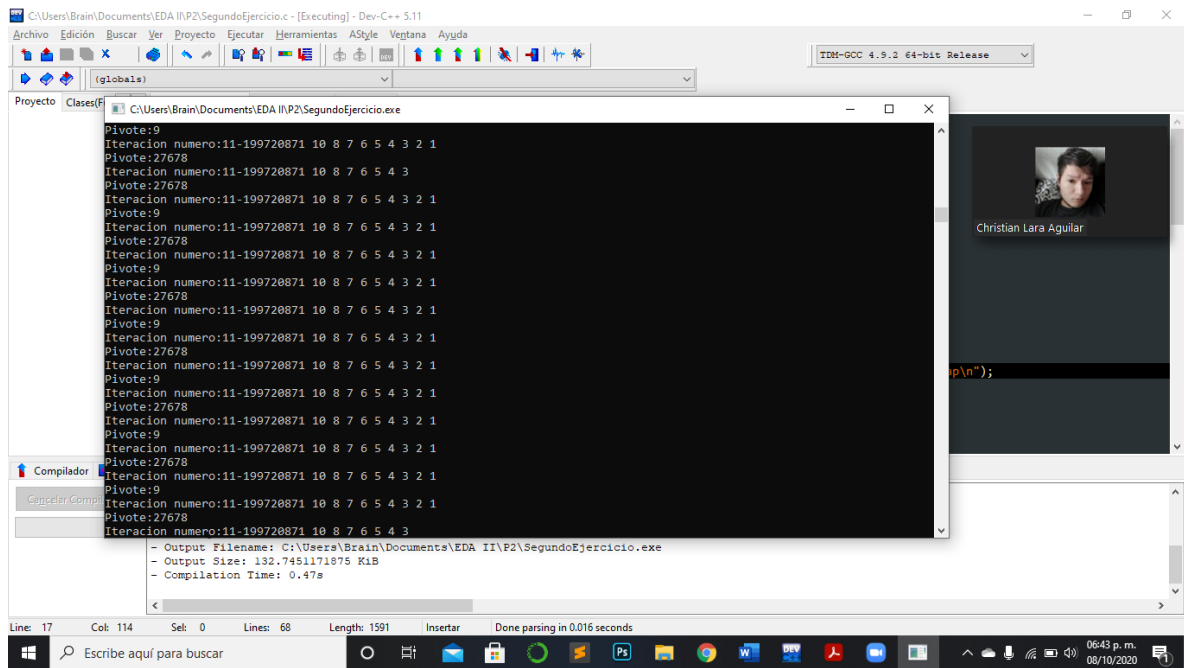
Línea: 66 Col: 27 Sel: 0 Línea: 139 Length: 2926 Insertar Done

Al comparar los elementos del Quicksort que se nos dio en clase, lo que hace el código es la asignación del pivote inicial al final de la lista que se entrega, después se asignan los índices  $i$ ,  $j$ , donde  $j > i$ , y se crea una condición de intercambio entre los dos, para cuando se encuentran los menores del pivote se va moviendo los valores mayores a la izquierda y al final se coloca el mayor de los valores al lado derecho del pivote, y se regresa el valor de la posición que tenía este valor mayor, y después se parte el arreglo a partir de esa posición en  $i$ .

Por lo que no es precisamente igual a los que hemos visto debido a que los otros, partían la lista a la mitad o se comenzaba con el primer elemento de la lista, lo que hace este es encontrar el mas grande de los menores del pivote y después intercambiar con el pivote para poder partir la lista a partir de ahí.

```
swap(&a[j], &a[j+1]);
for(verifica=0; verifica<n; i++){
    if(a[verifica]>a[verifica+1]){
        verifica=n+1;
    }
    if(verifica==n){
        j=i;
        i=0;
    }
}
```

Para romper el Bubble y no continúe ordenando cuando la lista esta ordenada, es necesario decirle que esta ordenada y por ello se hace otro for que recorre la lista, pero empeora la complejidad del algoritmo, por otro lado se busca que todos los elementos estén ordenados y que si se cumple la condición se finalicen todos los ciclos.



En esta parte pues tuve problemas en el Quicksort por lo que tuve que dejarlo de esa manera, para no perder tiempo en la elaboración/arreglo del programa.