



## Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

## Laboratorios de computación salas A y B

*Profesor:* TISTA GARCÍA EDGAR

*Asignatura:* ESTRUCTURA DE DATOS Y ALGORITMOS II

*Grupo:* 5

*Número de Práctica(s):* Guía práctica de estudio 1: Algoritmos de Ordenamiento de Datos

*Integrante(s):* MURRIETA VILLEGAS ALFONSO

*Semestre:* 2019 - 1

*Fecha de entrega:* 14 DE AGOSTO DE 2018

*Observaciones:*

**CALIFICACIÓN:** \_\_\_\_\_

# ALGORITMOS DE ORDENAMIENTO DE DATOS. PARTE 1

## INTRODUCCIÓN

En el mundo de la programación existen ciertas características que hacen a éste metódico, para poder resolver cualquier problema siempre se debe plantear un algoritmo el cual se encargará de dar ya sea a primera instancia una solución cualquiera, o la mejor solución, posteriormente este se lleva a través de un lenguaje de programación para poder representarlo en lo que conocemos como programa.

Pero antes, se le conoce como **algoritmo** a un conjunto finito de instrucciones libre de ambigüedades que sirven para realizar una tarea o acción específica. Para hacer un algoritmo debe tener algunas características particulares como son:

- 1) Debe ser exacta la descripción de las actividades a realizar.
- 2) Debe excluir cualquier ambigüedad
- 3) Debe ser invariante en el tiempo como en lugar.

Por otro lado, entendemos el concepto de “ordenar” como la acción de reagrupar o reorganizar un conjunto de datos u objetos en una secuencia específica, la cual nos permite manejar información de manera más eficiente.

Dicho lo anterior, podemos definir que un algoritmo de ordenamiento es aquel que cumple con factores como la cantidad de memoria o el entorno de software para reacomodar colecciones de elementos de tal forma que todos sus elementos cumplan con una lógica respecto al criterio global ya sea ascendente o descendente.

Por último, las condiciones que deben cumplirse para que se pueda considerar un algoritmo de ordenamiento de datos son las siguientes:

- 1) **Verificación:** El algoritmo siempre devuelve los elementos en orden correcto.
- 2) **Tiempo de ejecución:** Para calcular el tiempo es importante contabilizar comparaciones, intercambios, o acceso al arreglo – lista.
- 3) **Memoria:** Se debe considerar la cantidad de memoria extra necesaria para realizar operaciones.

### *Clasificación de algoritmos de ordenamientos*

Pese existen muchos criterios para clasificar los algoritmos de ordenamiento de datos, estos se pueden clasificar en 4 grupos:

#### *1) Algoritmos de inserción:*

Se consideran un elemento a la vez y cada elemento insertado en la posición apropiada con respecto a los demás elementos que ya han sido ordenados.

Ejemplos: Shell sort, inserción binaria y hashing.

## 2) Algoritmos de intercambio:

En estos algoritmos se trabaja con parejas de elementos que se van comparando e intercambiando si no están en el orden adecuado. El proceso se realiza hasta que se han revisado todos los elementos del conjunto a ordenar.

Ejemplos: Bubble Sort y Quicksort.

## 3) Algoritmos de selección:

En estos algoritmos se selecciona el elemento mínimo o el máximo de todo el conjunto a ordenar y se coloca en la posición apropiada. Esta selección se realiza con todos los elementos restantes del conjunto.

## 4) Algoritmos de enumeración:

Se compara cada elemento con todos los demás y se determina cuántos son menores que él. La información del conteo para cada elemento indicará su posición de ordenamiento.

Otras formas de clasificar el tipo este tipo de algoritmos es a través de la memoria usada como es el caso de: Ordenamiento interno y Ordenamiento externo. A su vez también se pueden clasificar por su estructura: Directo (Sobre el mismo elemento) e Indirecto (A través de otros elementos).

En el caso del ordenamiento interno podemos destacar que se caracteriza por el rápido acceso aleatorio a la memoria, mientras que en el ordenamiento externo se necesita recursos de la memoria secundaria.

## OBJETIVOS DE LA PRÁCTICA

- El estudiante identificará la estructura de algoritmos de ordenamiento *Bubble Sort* y *Quick Sort*

## DESARROLLO

### Actividad 1. Análisis del manual de prácticas

En este caso los 2 métodos que se llevarán a cabo serán 2 algoritmos por intercambio como son Bubble Sort y Quick sort.

#### 1.1 Bubble Sort

##### Generalización

Bubble Sort es uno de los algoritmos de ordenamiento de datos más conocidos, básicos y fáciles de entender sin embargo este carece de ser eficiente y esto se debe a que dentro del algoritmo se revisa elemento por elemento de la estructura empleada, donde en caso de que se cumpla la condición principal del algoritmo se intercambian de posición los elementos de la estructura.

## Ejemplificación

Imaginemos una lista con los siguientes elementos [9,21,4,40,10,35] lo que se hace dentro de este algoritmo de manera visual es lo siguiente:

### Primera Pasada

{9,21,4,40,10,35} --> {9,21,4,40,10,35} No se realiza intercambio  
{9,21,4,40,10,35} --> {9,4,21,40,10,35} Intercambio entre el 21 y el 4  
{9,4,21,40,10,35} --> {9,4,21,40,10,35} No se realiza intercambio  
{9,4,21,40,10,35} --> {9,4,21,10,40,35} Intercambio entre el 40 y el 10  
{9,4,21,10,40,35} --> {9,4,21,10,35,40} Intercambio entre el 40 y el 35

### Segunda Pasada

{9,4,21,10,35,40} --> {4,9,21,10,35,40} Intercambio entre el 9 y el 4  
{4,9,21,10,35,40} --> {4,9,21,10,35,40} No se realiza intercambio  
{4,9,21,10,35,40} --> {4,9,10,21,35,40} Intercambio entre el 21 y el 10  
{4,9,10,21,35,40} --> {4,9,10,21,35,40} No se realiza intercambio  
{4,9,10,21,35,40} --> {4,9,10,21,35,40} No se realiza intercambio

*Imagen 1: Esquema de como bubble sort va ordenando elemento tras elemento dentro de la lista previamente planteada.*

Como se puede ver en el esquema anterior, este algoritmo necesariamente revisa cada uno de los elementos de la lista en cada una de las pasadas o recorridos que realiza, para determinar la cantidad de recorridos lo único que se necesita es determinar la longitud de la estructura utilizada.

Sin embargo, como bien se sabe, conforme más se estudia algo, es posible determinar o buscar una mejora a los algoritmos, como es el caso de este algoritmo.

Lo primero que podemos notar es que en la versión anterior del algoritmo (Ver imagen 2) aunque la lista esté ya ordenada aun así verificará elemento tras elemento, para ello lo único que se debe hacer es agregar alguna variable auxiliar o bandera que nos pueda ayudar en este tipo de casos.

```
bubbleSort(a, n)
  Para i=1 hasta n -1
    Para j=0 hasta i < n -2
      Si (a[j]>a[j+1]) entonces
        tmp=a[j]
        a[j]=a[j+1]
        a[j+1]=tmp
      Fin Si
    Fin Para
  Fin Para
Fin bubbleSort
```

```
bubbleSort2 (a, n)
Inicio
bandera = 1;
pasada=0
Mientras pasada < n-1 y bandera es igual a 1
  bandera = 0
  Para j = 0 hasta j < n-pasada-1
    Si a[j] > a[j+1] entonces
      bandera = 1
      tmp= a[j];
      a[j] = a[j+1];
      a[j+1] = tmp;
    Fin Si
  Fin Para
  pasada=pasada+1
Fin Mientras
Fin
```

*Imagen 2: Del lado izquierdo el pseudocódigo de la burbuja tradicional, en la derecha el pseudocódigo de la burbuja mejorada*

Lo primero que notamos es que dentro de la versión 2 de la burbuja utilizamos una variable auxiliar declarada como “bandera” para poder determinar o ayudar cuando ya no sea necesario recorrer completamente vez tras vez la estructura.

## Análisis de complejidad

En el caso de la burbuja simple o sin auxiliar tenemos una complejidad  $O(n^2)$  en el caso de la Burbuja simple mientras que en el caso en la “Burbuja 2 o mejorada” tenemos que su peor caso tiene el mismo grado de complejidad, sin embargo, en el mejor caso puede resultar  $O(n)$  donde realmente notamos una gran diferencia.

## Actividad (Python)

A través de un código de “Bubble Sort” previamente programado en Python se pide completar agregar el código para imprimir el número de pasadas o iteraciones que realiza cada función:

```
while pasada < len(A)-1 and bandera:
    bandera=False
    for j in range (len(A)-1):
        if A[j]>A[j+1]:
            bandera=True
            temp=A[j]
            A[j]=A[j+1]
            A[j+1]=temp
    print(pasada) #Línea agregada para checar las pasadas
    pasada=pasada+1
```

Imagen 3: Parte de código de Bubble Sort, en gris la línea agregada para imprimir el número de pasadas

Por otro lado, se pide que se ordene la lista en forma inversa, para ello lo que se realizó fue lo siguiente:

Primero fue colocar de manera conveniente al pivote en la última posición (También podríamos invertir la lista pero sería menos eficiente), esto con el propósito de que posteriormente al cambiar las comparaciones dentro del código (Pasamos de “>” a “<”) se pudiera dar de manera inversa la lista. A continuación, se muestra la parte de código que fue modificada para poder obtener en orden inverso la lista o arreglo:

```
while bandera:
    bandera=False
    for j in range (size-1):
        if A[j] < A[j+1]: #Invertimos los signos de comparación
            bandera=True
            temp=A[j]
            A[j]=A[j+1]
            A[j+1]=temp
    pasada=pasada+1
```

Imagen 4: Parte de código de Bubble Sort inverso, se puede ver como el signo de comparación en el if fue invertido, además de que en el ciclo for se empieza desde la última posición.

NOTA: size es una variable que contiene el tamaño de la lista.

## 1.2 Quick Sort

## Generalización

Es el algoritmo de ordenamiento más eficiente de los métodos de almacenamiento interno (En tiempo), fue propuesto por Charles Hoore.

Este Algoritmo de ordenamiento al igual que bubble sort es del tipo “Algoritmos por intercambio”, sin embargo, algo que caracteriza a este algoritmo es el paradigma que sigue “Divide y vencerás”, donde se caracteriza por tener 3 procesos involucrados:

- 1) **Divide:** Se divide en arreglo A en 2 Sub-arreglos utilizando un elemento pivote x de manera que de un lado queden todos los elementos menores o iguales a él y del otro los mayores.
- 2) **Conquistar o resolver sub-problemas:** Una vez que ordenada los sub-arreglos o sub-listas en las 2 partes anteriormente mencionados a través de la recursividad es como vuelve a evaluar cada uno de los sub-arreglos o sub-listas.
- 3) **Combina:** Por último, una vez que ya ha realizado todas las sub-listas necesarias para ordenas los elementos de la lista principal procede a conjuntar todas las sub-listas para de esa forma dar como resultado final la lista principal y final.

NOTA: Cabe destacar que una característica de este tipo de algoritmo no importa donde tomemos el pivote de nuestra lista (Cuando mencionamos al pivote nos referimos al elemento que se utilizará para evaluar) ya que es indiferente si empezamos desde el primer elemento o desde el último.

### Ejemplificación:

Para poder demostrar o ejemplificar el algoritmo es necesario pensar en una lista o arreglo de cierta cantidad de elementos

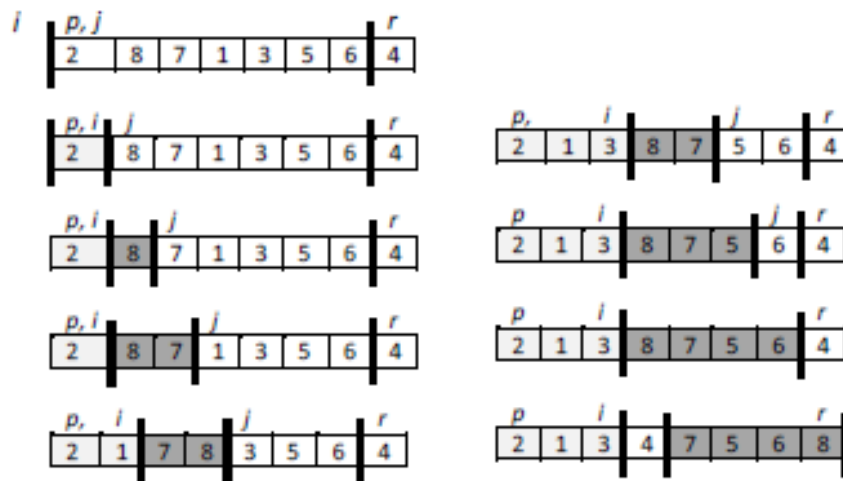


Imagen 4: Imagen tomada del manual de prácticas donde a través de una lista inicial se muestra cómo se realiza la división de un arreglo de 8 elementos, además se puede observar cada iteración, por último, también como queda el pivote marcando la división entre los elementos menores o iguales y mayores al pivote.

## Análisis de Complejidad:

El tiempo de ejecución del algoritmo depende de los particionamientos que se realizan si están balanceados o no, lo cual depende el número de elementos involucrados dentro de la lista.

Dicho lo anterior, podemos destacar que Quicksort es el algoritmo más rápido ofreciéndonos un tiempo de ejecución promedio  $O(n \log(n))$ , mientras que en su peor caso nos arrojaría  $O(n^2)$  el cual cabe destacar que es muy poco probable que suceda.

## Actividad (Python)

En este apartado se nos pidió que completáramos el código de manera que pudiéramos checar la salida del código, para ello lo único que fue necesario fue agregar una función alterna que nos pudiera ayudar a poder llamar a la función principal con el motivo de poder pasar los datos que requería.

A continuación, se muestra la salida del programa lista = [54,26,93,17,77,31,44,55,20] :

```
In [15]: runfile('C:/Users/AlfonsoMV/OneDrive/Documentos/Universidad/3° Semestre/EDA II/Prácticas_LAB/1_practice/
Códigos_Manual_Python/2_quicksort.py', wdir='C:/Users/AlfonsoMV/OneDrive/Documentos/Universidad/3° Semestre/EDA II/
Prácticas_LAB/1_practice/Códigos_Manual_Python')
[17, 20, 26, 31, 44, 54, 55, 77, 93]
```

Imagen 5: Lista ordenada a través de Quicksort en Python

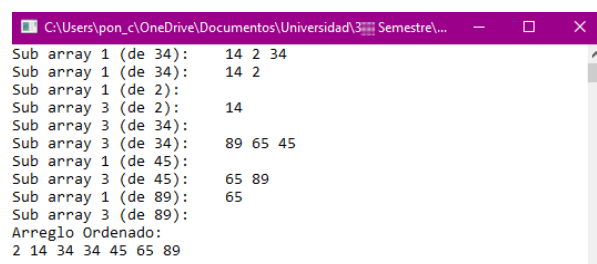
## Actividad 2. Ejercicios de laboratorio

### 2.1 QuickSort & BubbleSort

Para este problema se proporcionaron previamente los códigos de 3 bibliotecas, una genérica, una del algoritmo de Bubblesort y otra de Quicksort.

#### 1.1) Verificación el funcionamiento de las bibliotecas:

A continuación se muestra la salida empleando Quicksort de la lista arr[] = {14,2,45,34,89,65,34};



```
C:\Users\pon_c\OneDrive\Documentos\Universidad\3° Semestre\...
Sub array 1 (de 34): 14 2 34
Sub array 1 (de 34): 14 2
Sub array 1 (de 2):
Sub array 3 (de 2): 14
Sub array 3 (de 34):
Sub array 3 (de 34): 89 65 45
Sub array 1 (de 45):
Sub array 3 (de 45): 65 89
Sub array 1 (de 89): 65
Sub array 3 (de 89):
Arreglo Ordenado:
2 14 34 34 45 65 89
```

Imagen 6: Salida del programa que empleando Quicksort

A continuación se muestra la salida empleando Bubblesort de la lista arr[] = {14,2,45,34,89,65,34};

```

C:\Users\pon_c\OneDrive\Documentos\Universidad\3er Semestre\EDA II\Prácticas_LAB\1_practice\1_program.exe
2 14 45 34 89 65 34
2 14 45 34 89 65 34
2 14 34 45 89 65 34
2 14 34 45 89 65 34
2 14 34 45 65 89 34
2 14 34 45 65 34 89
End iteration
2 14 34 45 65 34 89
2 14 34 45 65 34 89
2 14 34 45 65 34 89
2 14 34 45 65 34 89
2 14 34 45 34 65 89
End iteration
2 14 34 45 34 65 89
2 14 34 45 34 65 89
2 14 34 45 34 65 89
2 14 34 34 45 65 89
End iteration
2 14 34 34 45 65 89
2 14 34 34 45 65 89
2 14 34 34 45 65 89
End iteration
Arreglo Ordenado:
2 14 34 34 45 65 89

```

Imagen 7: Salida del programa que empleando BubbleSort

## 1.2) Descripción de las funciones:

Para poder comprender a mayor profundidad los algoritmos y sobretodo las bibliotecas empleadas anteriormente, a continuación, se explicarán función tras función de cada biblioteca:

### **fgénérica:**

Esta biblioteca es la encargada de poder llevar a cabo los 2 puntos principales de y los 2 puntos que ambos algoritmos tienen en común, el intercambiar valores (Esto se debe a que son del mismo tipo de algoritmo) y la parte de la impresión.

#### *función “swap”:*

Es la función genérica encargada de intercambiar los elementos dentro del arreglo y esto a través de pasarle los apuntadores de los elementos. Además, es una función del tipo void debido a que no devuelve nada, solamente cambia la posición.

#### *función “printArray”:*

Esta función simplemente se encarga de imprimir el arreglo cada vez que sea llamada, para poder imprimir el arreglo entero solamente hace uso de un ciclo for condicionado con la dimensión de este.

### **QuickSort:**

Esta biblioteca es la encargada de llevar a cabo prácticamente toda la lógica del algoritmo de ordenamiento de datos “QuickSort” y para ello se usan 3 funciones principales:

#### *función “printSubArray”:*

Esta función se encarga de imprimir las sublistas generadas durante la ejecución del algoritmo, para ello lo que necesita es tanto el arreglo como el mayor como menor valor de esta.

NOTA: Los valores sirven para poder hacer las condiciones dentro del ciclo for que emplea en la impresión

#### *función “Quicksort”:*

Al igual que la anterior función, se le pasan los mismos argumentos, el arreglo, el valor mayor y el valor menor. Esta función es la encargada de llevar a cabo la impresión y llamado del arreglo y sub-arreglos.



*función “partition”:*

Por último, esta función se encarga de partir o dividir el arreglo principal en los arreglos secundarios o subarreglos, en cierta forma esta función es la que tiene la lógica y musculo del programa. Al igual que las anteriores funciones, se le pasan los mismos argumentos, el arreglo, el valor mayor y el valor menor.

### 1.3) Similitudes y diferencias

Realmente no existen diferencias entre lo visto en la clase (Teoría) con respecto a lo que acabamos de programar salvo tal vez las implementaciones y la forma en que se llevaron a cabo en un lenguaje de programación particular.

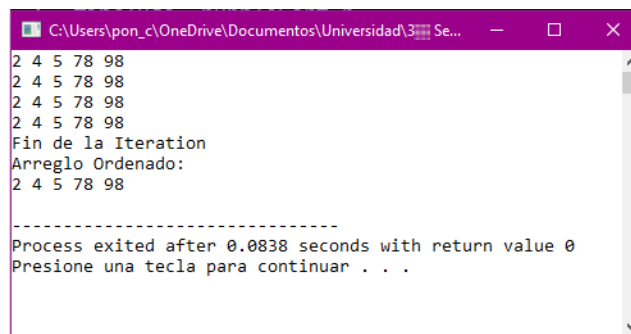
Por ejemplo, el programar Quicksort en Python realmente resulta muy sencillo con respecto a programarlo en C, sin embargo, la esencia o más que nada el algoritmo es el mismo.

Por último, como bien sabemos Quicksort es un algoritmo de ordenamiento que tiene distintas versiones ya que por ejemplo existe la versión recursiva o la versión iterativa (La que se utilizó durante el laboratorio) la cual obviamente es diferente.

### 1.4) Implementación de BubbleSort (Pseudocódigo en Clase)

Para este apartado se pidió que se llevara a cabo el pseudocódigo visto en clase el cual era la versión del algoritmo de la burbuja solamente que mejorada de manera que una vez que no hubiera cambios en una lista (O sea que ya esté o estuviera ordenada) simplemente terminara de ejecutar el código y no siguiera comparando el resto de los elementos.

A continuación, se muestra la salida de la siguiente lista (arreglo) {2,4,5,78,98}:



```
C:\Users\pon_c\OneDrive\Documentos\Universidad\3er Se...
2 4 5 78 98
2 4 5 78 98
2 4 5 78 98
2 4 5 78 98
Fin de la Iteration
Arreglo Ordenado:
2 4 5 78 98

-----
Process exited after 0.0838 seconds with return value 0
Presione una tecla para continuar . . .
```

*Imagen 8: Salida del programa que empleando BubbleSortModificado*

Como se puede ver solamente se hace una iteración y al no haber cambios directamente regresa la lista (arreglo).

### 1.5) BubbleSort con resultado descendente

Previamente en el la actividad de BubbleSort en Python ya se había realizado la inversión de la lista, lo único que realmente se hizo en este apartado fue adaptar el código previamente programado en Python en C.

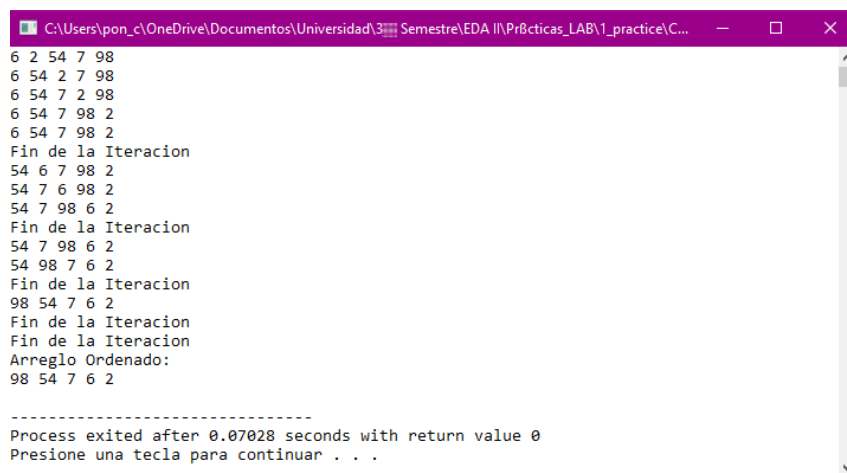
Lo primero que se hizo fue asociar directamente el valor o tamaño del arreglo a una variable, posteriormente declarar una bandera o comparador para poder utilizarla en un ciclo while (Esto para no hacer iteraciones innecesarias). Posteriormente y a diferencia del código en Python en este caso solamente se cambiaron los valores dentro de los arreglos tanto en la función swap como en la condición if.

```
for (j=0; j<i; j++) { //A diferencia de los otros bubble sort solo cambiamos la manera de iterar
    if(a[j]<a[j+1]){ //Invertimos los valores dentro de los arreglos
        swap(&a[j], &a[j+1]);
        index = j;
        cmp = 1; // si entra al ciclo, el valor se vuelve 1
    }
```

Imagen 9: Parte del código modificado para la salida inversa del arreglo ordenado

NOTA: Realmente puede hacerse esto o lo que se realizó en el caso del código en Python que fue invertir los signos de comparaciones (Mayor y menor que).

A continuación, se muestra de la salida del programa, donde podemos ver que el arreglo una vez ordenado ya de manera descendente.



```
C:\Users\pon_c\OneDrive\Documents\Universidad\3er Semestre\EDA II\Prácticas_LAB\1_practice\C...
6 2 54 7 98
6 54 2 7 98
6 54 7 2 98
6 54 7 98 2
6 54 7 98 2
Fin de la Iteracion
54 6 7 98 2
54 7 6 98 2
54 7 98 6 2
Fin de la Iteracion
54 7 98 6 2
54 98 7 6 2
Fin de la Iteracion
98 54 7 6 2
Fin de la Iteracion
Fin de la Iteracion
Arreglo Ordenado:
98 54 7 6 2

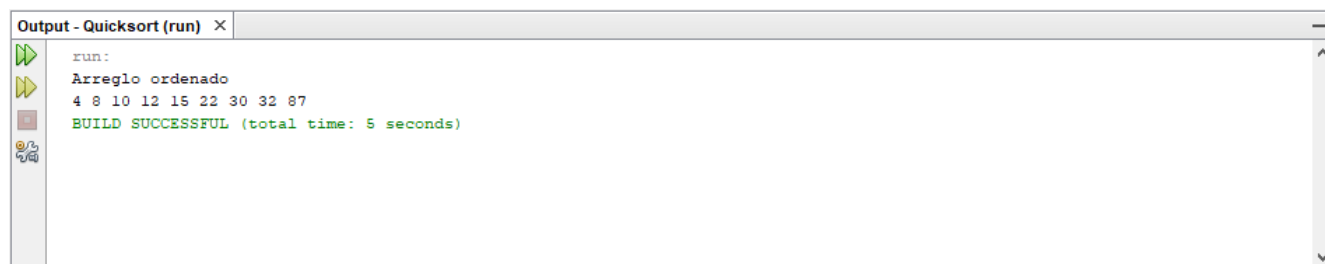
-----
Process exited after 0.07028 seconds with return value 0
Presione una tecla para continuar . . .
```

Imagen 10: Salida del programa que empleando BubbleSortInverso (Salida descendente)

## 2.2 QuickSort en Java

La gran diferencia entre Java y C realmente no recae en la sintaxis o incluso en las palabras reservadas que ambos tienen pues en estos aspectos realmente son casi similares, sin embargo, donde realmente encontramos la diferencia es en el paradigma al que están destinados. En el caso de C es un lenguaje estructurado y por ejemplo para llevar a cabo o pasar datos entre funciones es necesario en muchos casos utilizar apuntadores, sin embargo, en el caso de Java podemos omitir totalmente esto, pero aún mejor en este caso ya no vemos como tal “funciones”

sino métodos, métodos u acciones que están asociadas a un objeto. A continuación, se muestra la salida del programa:



```
run:
Arreglo ordenado
4 8 10 12 15 22 30 32 87
BUILD SUCCESSFUL (total time: 5 seconds)
```

Imagen 11: Salida del programa que empleando BubbleSort en Java

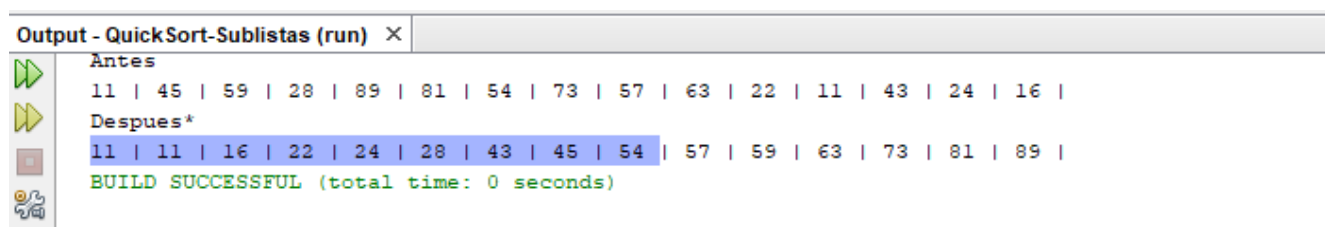
## 2.3 QuickSort mediante Sub-listas

Realmente este fue el problema más complicado de toda la práctica ya que el pasar de una lista o arreglo principal a varias sub-listas resultó bastante complicado. Lo primero que tuve que hacer fue decidir en qué lenguaje iba a llevar a cabo el algoritmo, al saber que debía pasar datos entre las distintas funciones o método directamente descarté utilizar C puesto me hubiera podido resultar mucho más complicado por no mencionar lo difícil que aún me resulta utilizar apuntadores, por lo que en un principio fui directamente con Python, sin embargo después de un largo rato de estar intentando hacer el algoritmo tuve muchos problemas sobre todo con la parte de la partición del arreglo (lista) inicial. (NOTA: Le arego el código en Python solamente que no funciona.)

Ante los pésimos resultados obtenidos, decidí totalmente ir con Java, pues al ser orientado a objetos me podría ofrecer una mejor comodidad con los métodos en comparación a C.

Debo destacar que el código está casi totalmente basado en el código de Quicksort que el profesor mandó, sin embargo, la parte lógica tuvo que cambiar pues en esta versión se necesitaban sub-listas para guardar elementos mayores y menores y además hacer uso de la recursividad para posteriormente unirlos.

A continuación, se muestra la entrada y salida de un arreglo:



```
Antes
11 | 45 | 59 | 28 | 89 | 81 | 54 | 73 | 57 | 63 | 22 | 11 | 43 | 24 | 16 |
Despues*
11 | 11 | 16 | 22 | 24 | 28 | 43 | 45 | 54 | 57 | 59 | 63 | 73 | 81 | 89 |
BUILD SUCCESSFUL (total time: 0 seconds)
```

Imagen 12: Salida del programa que empleando Quicksort mediante sub-listas en Java

NOTA: Agradecimientos totales a sitios web como **stackoverflow** y **tutorialpython** por ofrecerme ideas e incluso parte de códigos que fueron empleados en esta práctica

## Conclusiones

Durante esta práctica se conocieron 2 algoritmos de ordenamiento de datos por intercambio, fue el caso de Bubble Sort y Quicksort. En el caso de Bubble Sort realmente fue un algoritmo de implementación sencillo sobre todo porque la lógica no es difícil de entender, sin embargo, tiene la gran desventaja de que su complejidad es  $O(n^2)$  (En su mejor y promedio caso) la cual resulta realmente ineficiente respecto a otros algoritmos. Aunque modificando y optimizando el algoritmo esto puede cambiar muchísimo.

Por otro lado, Quicksort realmente resulta un poco más complejo de comprender sobre todo por todo lo que necesita para poder validar y realizar sus iteraciones.

Esta práctica realmente significo un primer y gran avance a nuestro apartado real y no solamente teórico con algoritmos de ordenamiento de datos y pese no era el propósito principal o incluso no se había llegado a pensar, realmente me resultó interesante el poder notar las diferencias entre los 3 lenguajes de programación que se emplearon en esta práctica como fueron Python, C y Java.

Por ejemplo, el ver Quicksort en Java resulta muy llamativo sobre todo porque todo parece muy metódico y ordenado y esto en parte se debe a el paradigma al que está pensado, sin embargo y siendo sincero el llevar a cabo los algoritmos en Python me resultó mucho más sencillo y esto supongo se debe a su forma fácil y dócil de poder otorgar gran simplicidad al momento de escribir el código.

En verdad, el programar en distintos lenguajes te da la posibilidad de no solamente poder ver las diferencias sintácticas sino también las diferencias de implementación y de recursos y posibilidades que cada lenguaje ofrece.

Por otro lado, el abstraer un problema realmente requiere una gran capacidad de lógica debido a que el plantear ya sea de forma general o particular un problema se necesita saber hacerlo sin ambigüedades y considerar que debe ser lo óptimo para el caso del que se está tratando, tal vez en algunos casos simplemente se necesite una solución cualquiera mientras que en otros casos tal vez se necesite una solución que no consuma tanta memoria o incluso la solución más rápida.

## REFERENCIAS

Elba Karen Saenz García. (2017). *Manual de Prácticas del laboratorio de Estructuras de Datos y algoritmos II*. UNAM, Facultad de Ingeniería.

Bradley N. Miller y David L. Ranum, Franklin, Beedle & Associates. (2011). *Problem Solving with Algorithms and Data Structures using Python*. Segunda Edición.

Mark J. Guzdial, Barbara Ericson. (2015). *Introducción a la computación y programación con Python*. 3ra Edición. Madrid España.