Reporte de Tarea 3
Tiempos y Ordenes de Duración

Luis Diego Fernández Coto - B22492
Daniel Jiménez León - B23467

2 de febrero de 2017

Índice

---

1. Enunciado

Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

`IE-0217` Estructuras abstractas de datos y algoritmos para ingeniería

Tarea 3: Tiempos y Ordenes de Duración

M. Sc. Ricardo Román Brenes - `ricardo.roman@ucr.ac.cr`

III-2016

Tabla de contenidos

---

1. Enunciado

1. Escriba en pseudocódigo los algoritmos de *Selection sort* y *Binary search*.

2. Calcule la función de tiempo de ejecución $T(N)$ y el orden de Duración (complejidad temporal) $O(T(N))$ de dichos algoritmos.

3. Programe ambos algoritmos en Python y C++ de tal forma que reciban por línea de comandos la ruta de un archivo que contenga una lista de números a operar. Dicho archivo estará compuesto de una única línea separada por espacios: `2 85 2 553 43 8 7 3 18 9 78`

```
./t4 ss arch.ivo
2 2 3 8 7 9 18 43 85 79 533
time: XXXX ms.

.\t4 bs 3 arch.ivo
1
time: XXXX ms.

.\t4 bs 99 arch.ivo
0
time: XXXX ms.
```

4. Averigüe como tomar tiempos de ejecución tanto en Python como en C++.

5. Mida cuanto tardan los algoritmos del punto 1. No tome en cuenta el tiempo de levantado del archivo en memoria ni la impresión del resultado.

6. Tome los tiempos de ejecución de ambos algoritmos, al menos 3 veces, para listas aleatorias de largo $N = \{10^3, 10^6, 10^9, 10^{12}, 10^{15}\}$

7. Verifique que los tiempos medidos sean acordes con los tiempos y ordenes calculados. Grafique sus resultados.

2. Consideraciones

- Trabajo en grupos de 2.
- Genere un reporte en LaTeX que incluya al menos el enunciado, la solución propuesta, su código y sus conclusiones.
- Suba su código y documentación (doxygen, README, INSTALL) al GitHub respectivo de su grupo y el directorio del laboratorio.
- Suba su código con documentación interna al GitHub respectivo de su grupo y el directorio del laboratorio.
- Cada estudiante debe subir el reporte a Schoology. (`https://app.schoology.com/assignment/958732694/`).
- Recuerde que por cada día tardío de entrega se le rebajaran puntos de acuerdo con la formula: 3^d, donde $d > 1$ es la cantidad de días naturales tardíos.

2. Código

2.1. Código en C++

```
/*! \brief Tarea 3. Tiempos y Ordenes de Duracion
 *  \author Luis Diego Fernandez
 *  \author Daniel Jimenez
 *  \date 1 de Febrero del 2017
 *  \version 1.0
 */

#include <cstring>
#include <ctime>
#include <fstream>
#include <iostream>
#include <stdlib.h>
#include <string>
#include <vector>

using namespace std;

/*! \brief Metodo que hace una busqueda en una lista ordenada del tipo
    Binary Search
 *
 *  \param myVector[] Vector en el cual vienen los numeros ordenados
     para hacer la busqueda.
 *  \param size Tamanno del vector myVector[].
 *  \param find Numero que se quiere encontrar.
 */
int binarySearch (int myVector[], int size, int find) {
        int min = 0;
        int max = size-1;
        int flag = 0;

        clock_t begin = clock(); //se inicia un clocl

        while (flag == 0) {
                //si el elemento en la posicion de buscqueda es igual a
                    find hay exito
                if (myVector[(min + max) / 2] == find) {
            flag = 1;
            clock_t end = clock(); //se inicia otro clock
            cout << 1;
            //se imprime el tiempo de ejecucion del programa
            cout << "\ntime: " << (1000000 * (end - begin)) /
               CLOCKS_PER_SEC << "\u00B5s" << endl;
            return 1;
        }//Si el numero encontrado es mayor que el buscado, se cambia
```

```
            el limite mayor.
        else if (myVector[(min + max) / 2] > find)
            max = (min + max) / 2;
        //Si el numero encontrado es menor que el buscado, se cambia el
            limite menor.
        else if (myVector[(min + max) / 2] < find)
            min = (min + max) / 2;

       //si se revisaron todos los numeros se acaba la ejecucion con
          una condicio de no exito
       if(max == min + 1) {
                clock_t end = clock(); //se inicia otro clock
                cout << 0;
                //se imprime el tiempo de ejecucion del programa
                cout << "\ntime: " << (1000000 * (end - begin)) /
                   CLOCKS_PER_SEC << "\u00B5s" << endl;
                return 0;
        }
        }
        return 0;
}

/*! \brief Metodo ordena un vector de numeros en orden ascendente
    utilizando el Selection Sort
 *
 *  \param myVector[] Vector en el cual vienen los numeros no ordenados
    .
 *  \param size Tamanno del vector myVector[].
 */
void selectionSort (int myVector[], int size) {
    clock_t begin = clock(); // se inicia un clock

    int temp = 0;
    int posMin = 0;
    //primer for de ordenamiento
    for (int n = 0; n < size - 1; n++){
        posMin = n;
        //segundo for de ordenamiento
                for (int m = n + 1; m < size; m++) {
                        if (myVector[m] < myVector[posMin])
                                posMin = m;
                }
                //intercambio de valores en caso de que haya que
                    ordenar
                if (posMin != n) {
                        temp = myVector[n];
                        myVector[n] = myVector[posMin];
                        myVector[posMin] = temp;
```

```
                }
        }

        clock_t end = clock(); //se inicia otro clock

        //se imprime el vector
        for (int i = 0; i < size; ++i) {
                cout << myVector[i] << " ";
        }

        //se imprime el tiempo de ejecucion del programa
        //cout << "\ntime: " << (1000000 * (end - begin)) /
           CLOCKS_PER_SEC << "\u00B5s" << endl; //imprime en micro
           segundos
        cout << "\ntime: " << (1000* (end - begin)) / CLOCKS_PER_SEC <<
            "ms." << endl;
}

/*! \brief Metodo Main
 *
 *  \param argc Contador de la cantidad de argumentos que recibe el
    programa.
 *  \param argv Un vector que tiene los argumentos pasados por la linea
     de comandos.
 */
int main(int argc, char *argv[]) {
        //se definen ciertas variables de utilidad
        int find = 0; //va a recibir el numero que quiere buscar el
           binary search
        int size = 0; //ve a tener el tamano del vector
    int temp = 0;
    int ss = true; //para saber si meterme a la funcion del selection
       sort o binary search
    char* fileName; //va a tener el nombre del archivo de datos

    //dependiendo de la funcion que haya que hacer se van a recibir los
        argumentos de la
    //linea de comando de manera diferente
    if (strcmp (argv[1], "ss") == 0) //"ss" = selection sort
        fileName = argv[2];
    else if (strcmp (argv[1], "bs") == 0){ //"bs" = binary search
        find = atoi(argv[2]);
        fileName = argv[3];
        ss = false;
    }

    ifstream dataFile;
```

```
    dataFile.open(fileName); //abre el archivo

    //obtiene el tamanno + 1 del archivo
        while (!dataFile.eof()) {
                dataFile >> temp;
                size += 1;
        }

    dataFile.close(); //cierra el archivo

    size -= 1; //pone el tamanno real

    int myVector[size]; // vector que va a contener los numeros

    dataFile.open(fileName); //se vuelve a abrir el archivo

    //se llena el vector
    for (int index = 0; index < size; ++index)
        dataFile >> myVector[index];

    dataFile.close(); //cierra el archivo

    //se llama alguna de las 2 funciones dependiendo de los argumentos
    //que se le pasaron al programa
    if (ss)
        selectionSort(myVector, size);
    else
        return binarySearch(myVector, size, find);

    return 0;
}
```

2.2. Código en Python

```
##
#@brief Tarea 3. Tiempos y Ordenes de Duracion
#@author Luis Diego Fernandez
#@author Daniel Jimenez
#@date 1 de Febrero del 2017
#@version 1.0

import sys
from time import time

##@brief Metodo ordena un vector de numeros en orden ascendente
   utilizando el Selection Sort.
#@param Lista Lista en la cual vienen los numeros no ordenados.
```

```
def Selection_Sort(Lista):
    temp = 0
    posMin = 0
    #primer for de ordenamiento
    for n in range(0, len(Lista)-1):
        posMin = n
        #segundo for de ordenamiento
        for m in range(n + 1, len(Lista)):
            if (Lista[m] < Lista[posMin]):
                posMin = m
        #intercambio de valores en caso de que haya que ordenar
        if(posMin != n):
            temp = Lista[n]
            Lista[n] = Lista[posMin]
            Lista[posMin] = temp
    return Lista

##@brief Metodo que hace una busqueda en una lista ordenada del tipo
   Binary Search
#@param Lista Lista en la cual buscar el numero.
#@param NUM Numero que se desa encontrar.
def Binary_Search(Lista, NUM):
    MIN = 0
    MAX = len(Lista)
    flag = 0

    while (flag == 0):
        #si el elemento en la posicion de buscqueda es igual a find hay
            exito
        if (Lista[(MIN + MAX)/2] == NUM):
            flag = 1
            return 1
        #Si el numero encontrado es mayor que el buscado, se cambia el
           limite mayor.
        elif (Lista[(MIN + MAX)/2] > NUM):
            MAX = (MIN + MAX)/2
        #Si el numero encontrado es menor que el buscado, se cambia el
           limite menor.
        elif (Lista[(MIN + MAX)/2] < NUM):
            MIN = (MIN + MAX)/2
        #si se revisaron todos los numeros se acaba la ejecucion con
           una condicio de no exito
        if(MAX == MIN + 1):
            return 0


#Programa Principal
#Leo cual algoritmo quiero ejecutar.
```

```
if (sys.argv[1] == "bs"):
    #Guardo el numero que deseo buscar.
    num = int(sys.argv[2])

    #Abro el archivo, leo la primer linea y la paso a una lista de
       enteros.
    datos = open(sys.argv[3], 'r')
    datos = datos.readline()
    Lista = datos.split(' ')
    for i in range(0,len(Lista)):
        if(Lista[i] != ""):
            Lista[i] = int(Lista[i]);
        else:
            Lista.pop(i)

    #Inicio la medicion de tiempo.
    start_time = time()
    #Llamo la funcion.
    BS = Binary_Search(Lista, num)
    #Determino el tiempo.
    elapsed_time = time() - start_time
    #Imprimo la informacion.
    print BS
    print "\nTime: "
    print elapsed_time*1000 , "ms."

if (sys.argv[1] == "ss"):
    #Abro el archivo, leo la primer linea y la paso a una lista de
       enteros.
    datos = open(sys.argv[2], 'r')
    datos = datos.readline()
    Lista = datos.split(' ')
    for i in range(0,len(Lista)):
        if(Lista[i] != ""):
            Lista[i] = int(Lista[i]);
        else:
            Lista.pop(i)

    #Inicio la medicion de tiempo.
    start_time = time()
    #Llamo la funcion.
    SS = Selection_Sort(Lista)
    #Determino el tiempo.
    elapsed_time = time() - start_time
    #Imprimo la informacion.
    print SS
    print "\nTime: "
    print elapsed_time*1000 , "ms."
```

3. Solución propuesta

3.1. Punto 1

3.1.1. Selection Sort

```
temp
posmin

for (int n = 0; n < size - 1; ++n){
        posmin = n
        for (int m = n + 1; m < size) {
                if (lista[m] < lista[posmin])
                        posmin = m
        }
        if(posmin ! n) {
                temp = lista[n]
                lista[n] = lista[posmin]
                lista[posmin] = temp
        }
}
```

3.1.2. Binary Search

```
min = 0;
max = n;
flag = 0;

while (flag == 0) {
  if(array[(min + max)/2] == num)
    flag = 1
    return 1
  else if((min + max)/2 > NUM)
    max = (min + max)/2
  else if((min + max)/2 < NUM)
    min = (min + max)/2
  if(max == min + 1)
    return 0
}

return 0;
```

3.2. Punto 2

3.2.1. Selection Sort

Para obtener la función de tiempo de ejecución del algoritmo Selection Sort utilizamos el pseudocódigo que se puede observar en la sección 3.1.1. De esta manera observamos que tenemos un anidación de ciclos for lo que de inmediato nos da una pista que nos indica que la función probablemente es cuadrática. Analizando a fondo podemos preveer la ejecución de las siguientes instrucciones:

- Fuera de los ciclos for tenemos 2 instrucciones de asignación, más la asignación del ciclo for que se realiza solo una vez en la ejecución del programa. Tenemos 3 instrucciones fuera de los for.
- Dentro del 1 for tenemos una asignación normal, la asginación del segundo for y las 2 instrucciones restantes del 1 ciclo for, además de eso tenemos una instrucción de comparación y 3 más de asignación con acceso a memoria del vector (que cuentan por 2 instrucciones). Esto nos da un total de 11 instrucciones que se realizarán n - 1 (las veces que el 1 ciclo for itere) cantidad de veces.
- Dentro del 2 for tenemos 2 instrucciones propias del for mas 2 accesos a memoria de un vector (que cuentan por 2 instrucciones) además de una asignación. Un total de 7 instrucciones que se ejecutan cantidad de veces (tomando el peor caso posible).

Podemos ver el resultado de la función de tiempo de ejecución en la ecuación 2, basándonos en el análisis visto anteriormente.

$$T(n) = (3) + (11n - 11)(7n) \tag{1}$$

Simplificando le ecuación 1 tenemos:

$$T(n) = 77n^2 - 77n + 3 \tag{2}$$

Para el orden de duración, a partir de la ecuación 2, lo que prevalece es la variable con el orden mas alto, en este caso es n^2. Se muestra en 3 la ecuación.

$$O(T(n)) = O(n^2) \tag{3}$$

3.2.2. Binary Search

En caso del Binary Search, utilizando el pseudo código expuesto anteriormente en la sección 3.1.2, se inicia tomando el tiempo de las tres asignaciones, cada una con una unidad de tiempo. Posteriormente inicia el ciclo while que se toma una unidad de tiempo al hacer la comparación y esto lo realiza n veces, por lo tanto todo lo que se encuentre dentro del while se debe multiplicar por n.

Dentro de los if, los detalles importantes son los accesos a memoria que se toman dos unidades de tiempo, en este caso serian los arreglos. Además se debe adicionar una unidad mas por el cálculo del indice del arreglo y otro por la comparación. A esto se le suman las asignaciones dentro de los if o else if. La ecuación 4 muestra la suma de todas las unidades de tiempo expuestas anteriormente.

$$T(n) = (1 + 1 + 1) + n * (4 + 1 + 1 + 2 + 2 + 2 + 2 + 2 + 1) \tag{4}$$

Simplificando la ecuación obtenemos lo siguiente:

$$T(n) = 17n + 3 \tag{5}$$

Para el orden de duración, a partir de la ecuación 5, lo que prevalece es la variable con el orden mas alto, en este caso es n. Se muestra en 6 la ecuación.

$$O(T(n)) = O(n) \tag{6}$$

3.3. Punto 3

Ambos algoritmos fueron implementados en C++ y Python. Al ejecutar el programa, por linea de comando se indica cual algoritmo se desea ejecutar, seguidamente del número que se requiere buscar, esto para el caso del Binary Search, y por último la ruta donde esta el archivo de texto con el arreglo de números. La implementación se muestra en la sección 2.

3.4. Punto 4

3.4.1. C++

Para tomar tiempos en C++ se utilizó la clase **ctime**. De esta utilizamos los objetos tipo clock_t. Estos objetos nos dan un conteo de *clock ticks*, estas son unidades de tiempo constantes dependientes del sistema en el cuál se trabaje. Para obtener el tiempo se dividen los *clock ticks* por la macro *CLOCKS_PER_SEC* que viene dada en $\frac{clockticks}{s}$, de esta manera obtenemos un resutlado en segundos.

Para tomar los tiempos de ejecución de cierta parte del código se pone un clock al inicio de la sección y otro al final, luego se restan sus *clock ticks* y se convierten a segundos.

3.4.2. Python

Para tomar los tiempos de ejecución en Python se utilizó el modulo *time* el cual cuenta con funciones capaces de tomar el tiempo el algun momento determinado del código y volver a tomar otra muestra y a partir de la diferencia obtener el tiempo de ejecución de esa porción del programa.

Esto fue precisamente lo que se realizo, se tomo el tiempo al inicio de ejecutar la función con el algoritmo, y posterior a la función.

3.5. Punto 5

Se implementaron ambos métodos para tomar tiempos de ejecución tal y como fueron mencionados en la sección anterior (3.4) y esto puede ser consultado en el código mostrado en la sección 2.

3.6. Punto 6

3.6.1. Selection Sort

Cuadro 1: Tiempos de ejecución para el algortimos Selection Sort en C++ y Python

	Tiempo C++	**Tiempo Python**
Lista de largo		
10^1	2μs	20μs
	2μs	20μs
	2μs	20μs
10^2	36μs	636μs
	36μs	633μs
	36μs	631μs
10^3	2680μs	$45ms$
	2683μs	$46ms$
	2677μs	$46ms$
10^4	$182ms$	3,094s
	$169ms$	3,108s
	$164ms$	3,096s
10^5	13,610s	$6min\ 50s$
	13,581s	$6min\ 16s$
	13,618s	$6min\ 38s$
10^6	$23min\ 28s$	-
	$22min\ 57s$	-
	$22min\ 37s$	-

Se puede observar en el cuadro 1 que para lista de largo 10^6 en Python no hay un resultado, esto lo justificamos de la siguiente manera. Como se observa claramente, entre mayor es la magnitud del largo de la lista, mayor es la diferencia entre el tiempo de ejecución de un programa en C++ y uno en Python. Vemos que para un lista de 100 000 elementos el programa en Python dura aproximadamente 30 veces más que el de C++. Ahora como la función de tiempo es exponecial de grado 2, al aumentar el tamaño de la lista a 1 millón de elementos la diferencia de lo que duraría el programa de C++ y el de Python no sería de 30 veces, sino que sería mucho más grande por lo tanto se puede asumir que duraría varias decenas de horas en completar su ejecución.

3.6.2. Binary Search

Para ser uniforme con las pruebas a realizar para el algoritmo Binary Search, se utilizo en todos los casos el número 24780 para buscar en los archivos con las listas de números. Esto tiene ciertos inconvenientes a la hora de realizar las pruebas.

Es posible que dentro de alguna de las listas utilizadas (generadas bajo funciones que retornan números de manera aleatoria, es decir las listas fueron formadas aleatoriamente) si se encuentre el número utilizado para las pruebas, mientras que en otras no. Dependiendo de la distribución de los números puede que lo encuentre mas rápido.

Naturalmente se piensa que entre mas números se encuentren en la lista mas va a tardar el algoritmo y efectivamente así es. Pero los casos expuestos anteriormente pueden hacer variar esto, y efectivamente es lo que sucede. Para los últimos dos casos donde las listas son las mas grandes

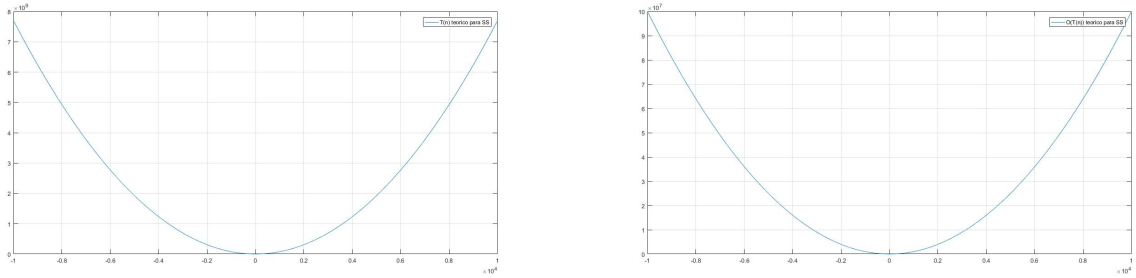
el tiempo de ejecución es el mas corto, porque si encuentra el numero y por eso el programa termina con un tiempo mucho menor. Principalmente se resalta en C++, esto porque los tiempos de ejecución son mucho menores y hay menos diferencia entre uno y otro.

Cuadro 2: Tiempos de ejecución para el algortimo Binary Search en C++ y Python

	Tiempo C++	**Tiempo Python**
Lista de largo		
10^1	2μs	4,053μs
	2μs	3,099μs
	2μs	3,099μs
10^2	3μs	5,960μs
	3μs	5,007μs
	3μs	5,007μs
10^3	3μs	6,914μs
	3μs	6,914μs
	4μs	5,960μs
10^4	4μs	13,113μs
	5μs	9,06μs
	5μs	15,974μs
10^5	1μs	11,921μs
	1μs	12,874μs
	1μs	14,067μs
10^6	2μs	15,974μs
	1μs	16,928μs
	1μs	15,020μs

3.7. Punto 7

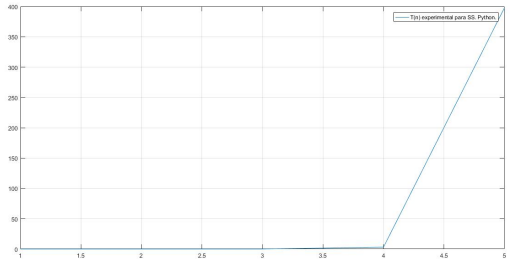
3.7.1. Selection Sort



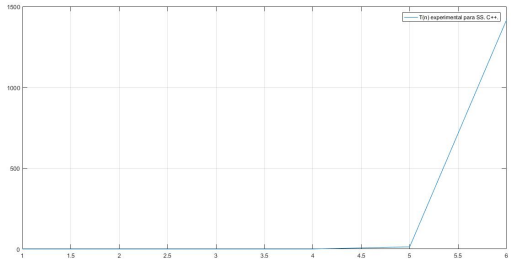
(a) T(n)

(b) O(T(n))

Figura 1: Selection Sort. Gráficas teóricas.



(a) T(n) utilizando Python.



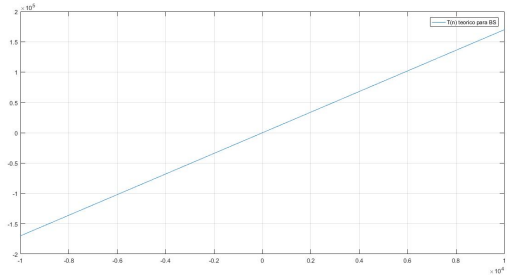
(b) T(n) utilizando C++.

Figura 2: Selection Sort. Gráficas experimentales.

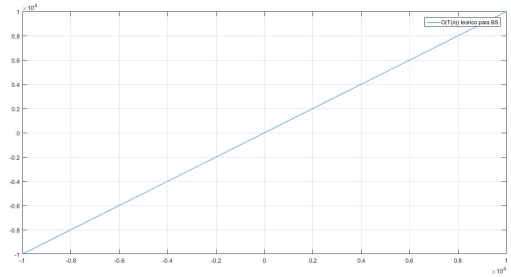
Se puede apreciar en la figuras 1a, 1b, 2a y 2 que las gráficas no se asemejan mucho, sin embargo hay que tomar ciertas consideraciones a la hora de interpretar estos resultados. La gráficas mostradas en las figuras 1a y 1b son teóricas y consideran el eje x negativo, aspecto que en la vida real es utópico ya que el tiempo es positivo, por tanto se puede ignorar dicha parte de las curvas. Luego en las figuras 2a y 2b se ve que la curva empieza pegada al 0 en el eje y postivo y no empieza a crecer como lo hace teóricamente, esto se da por 2 razones:

- Se trabaja con casos de prueba que al inicio tienen valores muy pequeños en cuanto a sus resultados, mas incrementan de manera exponencial luego y eso es una característica esperada. Se observa inclusive que la pendiente de crecimiento para ambas curvas se asemeja a gran manera.
- Se tiene muy pocas muestras que no permiten crear un mapeo de datos tan preciso que nos permita observar la concavidad y el crecimiento exponencial de la curva.

3.7.2. Binary Search

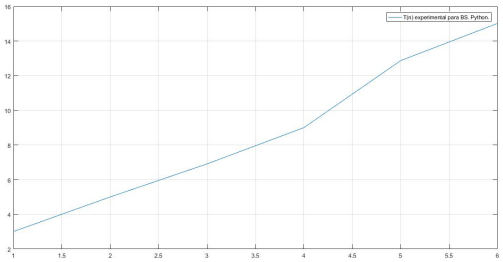


(a) T(n)

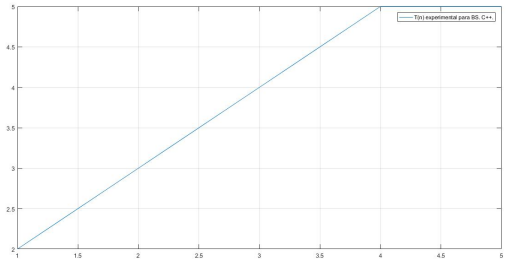


(b) O(T(n))

Figura 3: Binary Search. Gráficas teóricas.



(a) T(n) utilizando Python.



(b) T(n) utilizando C++.

Figura 4: Binary Search. Gráficas experimentales.

En el caso de este algoritmo de búsqueda, teóricamente se había llegado a que el tiempo de ejecución lo modelaba una ecuación lineal, y de la misma manera el orden de duración. Esto se puede ver reflejado en las gráficas 3a y 3b.

Posteriormente en las gráficas experimentales se esperan comportamientos similares y efectivamente así es, el crecimiento de la curva se da de manera lineal, conforme se aumenta el número de datos en la lista.

Si se observa en la figura 4a tiene una distorsión al llegar al final. Esto se debe a lo expuesto en la sección anterior, los casos donde el algoritmo si encontraba el número. Estos casos fueron eliminados de la gráfica utilizando C++ (4b) ya que distorsionaban demasiado la gráfica porque el cambio en los datos era demasiado significativo, tal y como se puede apreciar en la tabla 2 para los últimos dos casos.

4. Conclusiones

- A partir de un pseudo código bien hecho para un algoritmo se puede sacar su función de tiempo de ejecución $T(N)$ y su complejidad temporal $O(T(n))$ con una precisión aceptable.
- La ejecución de la implementación de un mismo algoritmo puede tener diferencias muy grandes en cuanto a su tiempo de ejecución dependiendo del lenguaje de programación en el cual haya sido programado.
- El lenguaje C++ es mucho más rápido en ejecución que el lenguaje Python.
- El algoritmo del Selection Sort es muy eficiente y rápido para listas de números pequeños, sin embargo, muy ineficiente y lento para listas de una tamaño muy grande.
- Los resultados teóricos y experimentales en cuanto a tiempo de ejecución y orden del algoritmo Selection Sort para ambos lenguajes son satisfactorios. En el momento que la cuva experimental crece toma un comportamiento algo parecido a la teórica exponencial.
- Respecto al algoritmo de búsqueda Binary Search, efectivamente se concluye que conforme aumenta el numero de elementos en la lista, el tiempo de respuesta aumenta linealmente. Los gráficas experimentales son las esperadas y respaldan lo obtenido teóricamente.
- El tiempo de respuesta del Binary Search es bastante aceptable en cualquiera de los dos lenguajes utilizados, aun mas en C++.