Laboratorio Nro. 2
Escribir el tema del laboratorio

Juan Jose Madrigal Palacio
Universidad Eafit
Medellín, Colombia
jjmadrigap@eafit.edu.co

Luis Ángel Jaimes Mora
Universidad Eafit
Medellín, Colombia
lajaimesm@eafit.edu.co

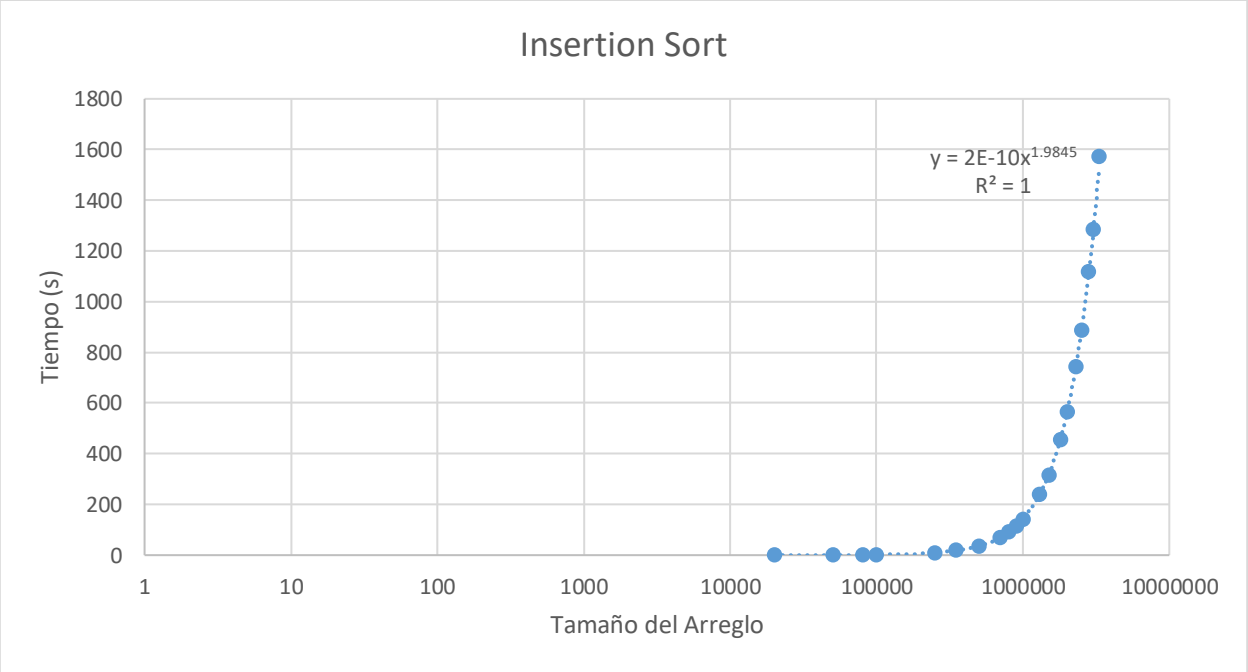
3) Simulacro de preguntas de sustentación de Proyectos

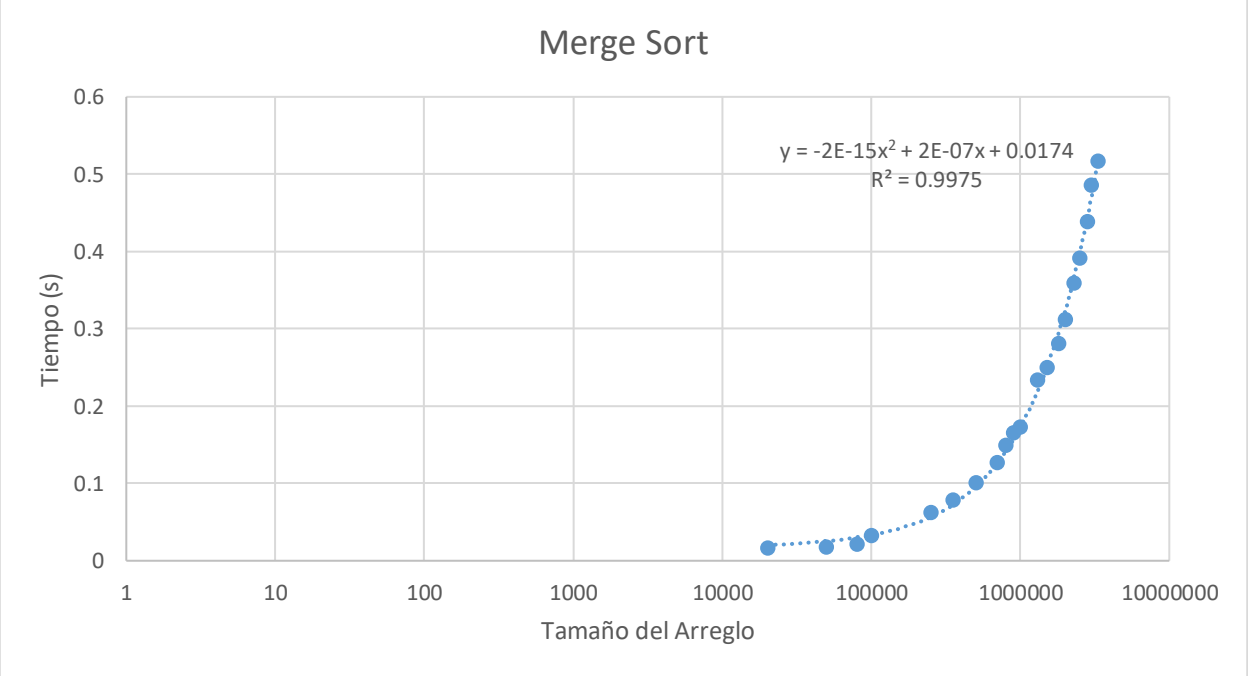
3.1

Insertion Sort			Merge Sort	
Tiempo (ms)	Tamaño del Arreglo		Tiempo (ms)	Tamaño del Arreglo
62	20000		16	20000
375	50000		17	50000
937	80000		21	80000
1500	100000		32	100000
8748	250000		62	250000
17762	350000		78	350000
35192	500000		101	500000
68814	700000		127	700000
89196	800000		149	800000
113073	900000		165	900000
139409	1000000		172	1000000
236380	1300000		234	1300000
314125	1500000		250	1500000
454454	1800000		281	1800000
562855	2000000		312	2000000
743201	2300000		359	2300000
885178	2500000		391	2500000
1116228	2800000		438	2800000
1284673	3000000		485	3000000
1570107	3300000		516	3300000

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

3.2





3.3 Merge Sort es más eficiente en números grandes pues, aunque con el tiempo también empieza a escalar y volverse más lento al igual que insetion sort, merge tiene una curva mucho menos pronunciada permitiendo ingresar cantidades mayores de datos y esto se mantiene a lo largo del tiempo, siendo solamente para números bastante pequeños en los cuales apenas hay una diferencia la cual rápidamente se diluye.

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

3.4 Insertion Sort no es nada apropiado para un juego con millones de capas ya que al aumentar exponencialmente el tiempo que demora en procesar los datos tal y como se ve en la prueba anterior con ya 3 millones de datos ya demora alrededor de 21 min(1284673 ms) lo cual en un videojuego conllevaría a un tiempo de carga excesivo que impediría jugar de forma normal.

3.5 -En los casos probados en ninguno insertion sort es más rápido que merge
-Para que insertion sort sea más rápido que merge es necesario que el arreglo tenga una cantidad pequeña de datos; ya que merge aunque sea más eficiente en números más grandes para un número mínimo de datos y en uno de los mejores casos hace un mayor cantidad de operación por ende se demorara más tiempo aunque debido a que insetion es menos eficiente no demora mucho para que con mayores cantidades de datos estos pasos "extra" al principio sean compensados por una mayor eficacia en su forma de organizar mayores cantidades de datos.

3.6-maxSpan o lapso Max se trata de un problema/ejercicio en el cual a partir de un arreglo de enteros(el cual puede estar vacío) que se manda al método maxSpan, se debe calcular cual la cantidad máxima de números entre dos números iguales incluyendo estos dos números en el arreglo, si no hay números iguales, solo hay uno, o este vacío se devolverá respectivamente 1,1,0.
-En nuestro código funciona así: Inicia el método maxSpan con un arreglo de enteros "nums", se crea un int(número entero) "max" iniciado en 0, luego se crea un ciclo for con un int "i" iniciando en 0, luego se crea otro int "j" iniciado con el tamaño del arreglo -1, se crea un bucle while el cual comprueba si el número de "nums" en la posición "i" es diferente al número en la posición "j"; si son distintos se resta 1 a "j"; sino se crea un int "lapso" igual a 1+("j"-"i"), se comprueba si "lapso" es mayor a "max", "max" pasa a tener el valor de lapso; se sigue hasta terminar el for cuando este termina se retorna el valor de "max"; fin del código.

3.7
-sameEnds

```
public boolean sameEnds(int[] nums, int len) {
  boolean FinalIgual = true; // C1
  for (int i = 0; i < len; i++) { // C2*n
    if (nums[i] == nums[nums.length-len+i]) // C3
      FinalIgual = true; // C4
    else
     FinalIgual = false; // C5
  }
  return FinalIgual; // C6
}
```

T(n)=C1+C2*n+C3+C4+C5+C6
T(n)=O(C1+C2*n+C3+C4+C5+C6)
T(n)=O(C2*n+C3+C4+C5+C6)
T(n)=O(C2*n)
T(n)=O(n)

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

-shiftLeft

```
public int[] shiftLeft(int[] nums) {
  int largo = nums.length; // C1
  int temporal = 0; // C2
  for(int i=0; i<largo-1; i++){ // C3*n
    temporal = nums[i]; // C4
    nums[i]= nums[i+1]; // C5
    nums[i+1] = temporal; // C6
  }
  return nums; // C7
}
```

T(n)=C1+C2+C3*n+C4+C5+C6+C7
T(n)=O(C1+C2+C3*n+C4+C5+C6)
T(n)=O(C3*n+C4+C5+C6)
T(n)=O(C3*n)
T(n)=O(n)

-post4

```
public int[] post4(int[] nums) {
  int largo = nums.length -1; // C1
  int index = 0; // C2
  for(int i = nums.length-1; i >= 0;i--){ // C3*n
    if(nums[i] == 4){ // C4
      index = i; // C5
      break; // C6
    }
  }
  int[] arr = new int[largo-index]; // C7
  for(int i = index+1, j=0; i < nums.length; i++, j++){ // C8*n*m
    arr[j] = nums[i]; // C9
  }
  return arr; // C10
}
```

T(n)= C1+C2+C3*n+C8*n*m+C4+C5+C6+C7+C9+C10
T(n)=O(C3*n+C8*n*m+C4+C5+C6+C7+C9+C10)
T(n)= O(C8*n*m+C4+C5+C6+C7+C9+C10)
T(n)= O(C8*n*m)
T(n)= O(n*m)

-withoutTen

```
public int[] withoutTen(int[] nums) {
  int resultado = 0; // C1
  for(int i = 0; i < nums.length; i++){ //C2*n
    if(nums[i] != 10){ // C3
      int temporal = nums[i]; // C4
      nums[i] = 0; // C5
      nums[resultado] = temporal; // C6
```

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

```
      resultado++; // C7
    }
    else{
      nums[i] = 0; // C8
    }
  }
  return nums; // C9
}
```

T(n)=C1+C2*n+C3+C4+C5+C6+C7+C8+C9
T(n)=O(C1+C2*n+C3+C4+C5+C6+C7+C8+C9)
T(n)=O(C2*n+C3+C4+C5+C6+C7+C8+C9)
T(n)=O(C2*n)
T(n)=O(n)

-fizzBuzz

```
public String[] fizzBuzz(int start, int end) {
  String[] resultado = new String[end - start]; // C1
  for (int i = start; i < end; i++){ // C2*n*m
    if(i % 3 != 0 && i % 5 != 0){ // C3
      resultado[i - start] = String.valueOf(i); // C4
    }if(i % 3 == 0){ // C5
      resultado[i - start] = "Fizz"; // C6
    }if(i % 5 == 0){ // C7
      resultado[i - start] = "Buzz"; // C8
    }if(i % 3 == 0 && i % 5 == 0){ // C9
      resultado[i - start] = "FizzBuzz"; // C10
    }
  }
  return resultado; // C11
}
```

T(n)=C1+C2*n*m+C3+C4+C5+C6+C7+C8+C9+C10+C11
T(n)=O(C1+C2*n*m+C3+C4+C5+C6+C7+C8+C9+C10+C11)
T(n)=O(C2*n+C3+C4+C5+C6+C7+C8+C9+C10+C11)
T(n)=O(C2*n*m)
T(n)=O(n*m)

3.8 En cada uno de los ejercicios existe la una variable “n” la cual se trata de la cantidad de veces que hay que realizar su respectiva línea de código, esta cantidad de veces varía en función el valor del parámetro ingresado al método cuando se inicia el código.
En algunos de los ejercicios hay una variable “m” la cual funciona de la misma forma que la variable “n” en sus respectivos ejercicios

4) Simulacro de Parcial

4.1 c
4.2 b
4.3 b

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

4.4 b
4.5 d
4.6 a
4.7.1 T(n-1) + C
4.7.2 n
4.8 a
4.9 d
4.10 c
4.11 c
4.12 b
4.13 c
4.14 a

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473