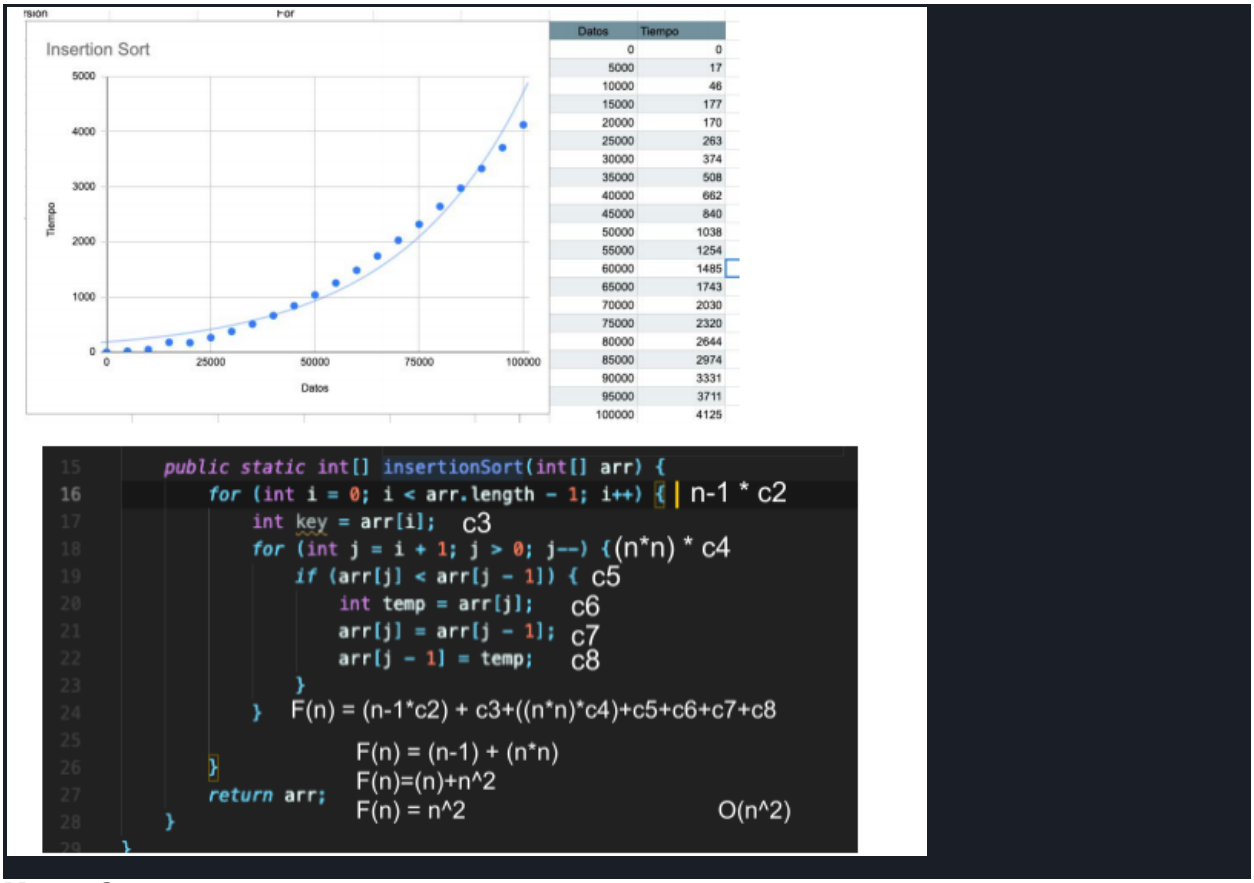
https://github.com/mauriciotoro/ST0245-Eafit/blob/master/laboratorios/lab02/ED1-Laboratorio2%20Vr%2016.0.pdf

Insertion sort



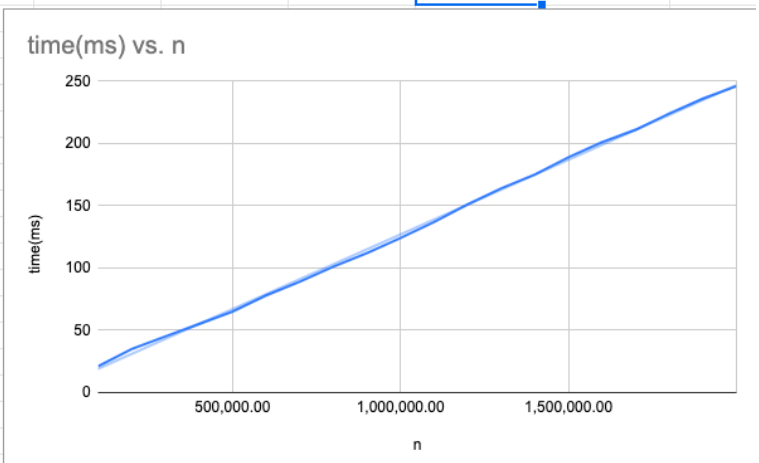
Merge Sort

```
import java.util.Arrays;



public class Merge {

   public static void main(String[] args) {

       int[] nums= {12, 11, 13 ,5, 6, 7 };

       sortList(nums, 0, nums.length - 1);

       System.out.println(Arrays.toString(nums));

   }
```

```java
public static void merge(int[] nums, int start,int medium,int end){

    int b = end-medium;
    int t = medium-start+1;

    int[] botom = new int[b];
    int[] top = new int[t];


    for(int tt = 0;tt < t;tt++){
        top[tt] = nums[start+tt];
    }

    for(int tt = 0;tt < b;tt++){
        botom[tt] = nums[medium+tt+1];
    }

    int contadorBotom  = 0;
    int contadorTop = 0;
    int mainContador = start;

    while (contadorBotom < b && contadorTop < t) {

        if (botom[contadorBotom] <= top[contadorTop]) {

            nums[mainContador] = botom[contadorBotom];
            contadorBotom++;
        }
        else {
            nums[mainContador] = top[contadorTop];
            contadorTop++;
        }
        mainContador++;
    }

    while (contadorBotom < b) {

        nums[mainContador] = botom[contadorBotom];
        contadorBotom++;

        mainContador++;
```

```
        }

        while (contadorTop < t) {
            nums[mainContador] = top[contadorTop];
            contadorTop++;

            mainContador++;
        }

    }

    public static void sortList(int[] nums,int start,int end){

        if(start < end){

            int mid = start+(end-start)/2;

            sortList(nums,start,mid);
            sortList(nums,mid+1,end);

            merge(nums,start,mid,end);

        }

    }

}
```

n	time(ms)
100,000.00	21
200,000.00	35
300,000.00	45
400,000.00	55
500,000.00	65
600,000.00	78
700,000.00	89
800,000.00	101
900,000.00	112
1,000,000.00	124
1,100,000.00	137
1,200,000.00	151
1,300,000.00	164
1,400,000.00	175
1,500,000.00	189
1,600,000.00	201
1,700,000.00	211
1,800,000.00	224
1,900,000.00	236
2,000,000.00	246



1)
```java
public boolean canBalance(int[] nums) {
    int left = 0;
    int right = nums.length - 1;
    int sum = 0;
    boolean equal = false;
    for (int i = 0; i < nums.length - 1; i++) {
        sum += nums[i];
        int auxSum = 0;
        for (int j = i + 1; j < nums.length; j++) {
            auxSum += nums[j];
            if (j == right && auxSum == sum) {
                return true;
            }
        }
    }
    return false;
}
```

O(n^2) peor caso

2)
```java
public static boolean linearIn(int[] outer, int[] inner){
```

```
   int innerContador = 0;
   for(int tt = 0;tt<outer.length;tt++){

       if(innerContador < inner.length && outer[tt] == inner[innerContador]){
           innerContador++;
       }
   }

    return innerContador == inner.length;
}
       // F(n) = c4 + n(c7)
       // F(n) = n(c7)
       // F(n) = n

       O(n) - donde n es el numero de elementos en el arreglo outer
```

3)

```
public static int[] seriesUp(int n) {

       int[] nums = new int[(n*(n+1))/2];

       int start = 1;
       int contador = 0;
       for(int grande = 0; grande < nums.length; grande+= contador){
           for(int peque = 0; peque <= contador;peque++){
              nums[grande+peque] = start;
              start++;
           }
           start = 1;
           contador++;
       }
       return nums;
    }
       F(n) = c3 + n(c5)
O(n) - donde n está dado por (n*(n+1))/2 siendo n el el input original de la función
```

4)

```
public static int[] squareUp3(int n) {

       if(n==0){return new int[0];}
```

```
        int[] nums = new int[n*n];

        int indexes = 1;
        for(int grande = n;grande <= n*n;grande+=n){
            for(int peque = indexes; peque > 0;peque--){
                 nums[grande-peque] = peque;
            }
            indexes++;
        }
         return nums;
}
        //O(n) - donde n está dado por n*n de el input de la función original.
```

5)

```
        public int maxMirror(int[] nums) {
        int max = 0;
        int iaux = 0;
        boolean equal = false;
        for (int i = 0; i < nums.length; i++) {
            int cont = 0;
            iaux = i;
            for (int j = 0; j < nums.length - i; j++) {
                equal = false;
                if (nums[iaux] == nums[nums.length - 1 - j]) {
                    cont++;
                    iaux++;
                    equal = true;
                }
                if (!equal && cont > 0) {
                    if (max < cont) {
                        max = cont;
                    }
                    cont = 0;
                }
            }
            if (max < cont) {
                max = cont;
            }

        }
        return max;
    }
```

Array 2

```
1) public String[] fizzBuzz(int start, int end) {
       String[] listWords = new String[end - start];
       for (int i = 0; i < end - start; i++) {
           if ((i + start) % 3 == 0 && (i + start) % 5 == 0) {
               listWords[i] = "FizzBuzz";
           } else if ((i + start) % 3 == 0) {
               listWords[i] = "Fizz";
           } else if ((i + start) % 5 == 0) {
               listWords[i] = "Buzz";
           }

           else
               listWords[i] = String.valueOf(i + start);
       }
       return listWords;
   }

2) public int[] evenOdd(int[] nums) {
       int[] newArr = new int[nums.length];
       int even = 0;
       int odd = nums.length - 1;

       for (int i = 0; i < nums.length; i++) {
           if (nums[i] % 2 == 0) {
               newArr[even] = nums[i];
               even++;
           } else {
               newArr[odd] = nums[i];
               odd--;
           }
       }
       return newArr;
   }
3) public boolean haveThree(int[] nums) {
       int cont = 0;
```

```
        for (int i = 0; i < nums.length; i++) {
            if (nums[i] == 3) {
                cont++;
                i++;
            }
        }
        return cont == 3;
    }
```

4)

```
    public int[] withoutTen(int[] nums) {

        for (int i = 0; i < nums.length - 1; i++) {
            for (int j = i + 1; j < nums.length; j++) {
                if (nums[i] == 10) {
                    int temp = nums[i];
                    nums[i] = nums[j];
                    nums[j] = temp;

                }

            }
        }
        for (int i = 0; i < nums.length; i++) {
            if (nums[i] == 10) {
                nums[i] = 0;
            }
        }

        return nums;
    }
```

5)

```
    public int[] zeroMax(int[] nums) {
        int max = 0;
        for (int i = 1; i <= nums.length; i++) {
            if (nums[nums.length - i] % 2 == 1) {
                max = Math.max(max, nums[nums.length - i]);
            }
            if (nums[nums.length - i] == 0) {
                nums[nums.length - i] = max;
            }
        }
```

```
    return nums;
}
```

1.1 Midan los tiempos de ejecución de los algoritmos *Insertion Sort* y *Merge sort*

2.1 Resuelvan –mínimo-- 5 ejercicios del nivel *Array 2* de la página *CodingBat*: **http://codingbat.com/java/Array-2**

2.2 Resuelvan --mínimo-- 5 ejercicios del nivel *Array 3* de la página *CodingBat*: **http://codingbat.com/java/Array-3**

3.1 [Ejercicio opcional] De acuerdo a lo realizado en el numeral 1.1, construyan una tabla donde muestren, para cada algoritmo, cuánto se demora para 20 tamaños del problema.

3.2 Grafiquen los tiempos que tomó, cada algoritmo, para los 20 tamaños del problema.

3.3 Según la complejidad en tiempo, ¿es apropiado usar *insertion sort* para un videojuego con millones de elementos en una escena y demandas de tiempo real en la renderización de escenas 3D?

3.4 ¿Por qué aparece un logaritmo en la complejidad asintótica, para el peor de los casos, de *merge sort* o *insertion sort*?

3.5 [Ejercicio opcional] Para arreglos grandes, ¿cómo deben ser los datos para que *insertion sort* sea más rápido que *merge sort*? ¿ordenados? ¿todos iguales? ¿diferentes?

3.4 [Ejercicio opcional] Expliquen con sus propias palabras cómo funciona su solución del ejercicio de *Array3,* de *Coding Bat,* llamado *maxSpan.*

3.5 Calculen la complejidad de los ejercicios en línea, numerales 2.1 y 2.2.

3.6 Expliquen con sus palabras las variables (qué es 'n', qué es 'm', etc.) del cálculo de complejidad del numeral anterior.

4.2 Dayla sabe que la complejidad asintótica de la función P(n) es $O(\sqrt{n})$. Ayúdenle a Dayla a sacar la complejidad asintótica para la función mistery(n,m).

```
void mistery(int n, int m) {
  for(int i = 0; i < m; ++i){      O(m)
    boolean can = P(n);            O(n^½)
    if(can)
      for(int i = 1; i * i <= n; i++)  O(n^½)
        //Hacer algo en O(1)
    else
      for(int i = 1; i <= n; i++)      O(n-1)
        //Hacer algo en O(1)
} }
```

Peor caso sería O(n*n^½ * m) cuando can sea siempre falso

La complejidad de mistery(n,m) es:

4.3 Considera el siguiente algoritmo:

```
void f(int n){
  for(int i=1; i*i <= n; i++) {        O(n^½ )
    for(int j=1; j*j<=n; j++) {        O(n^½*n^½ )
      for(int k=0; k<n; k++) {         O(n*n^½*n^½ )
        for(int h=0; h<=n; h++) {      O(n*n*n^½*n^½ )
          System.out.println("hola");
        }
      }
    }
  }
}
```

Worst case is O(n^3)

¿Cuál es la complejidad asintótica, para el peor de los casos, del algoritmo *f*(*n*) para *n*>1?

4.5) (d) $T(n)=T(n/10)+c$, que es $O(\log_{10} n)$