LABORATORIO 2

Juan Pablo Leal Jaramillo-Santiago Albisser Cifuentes

2. EJERCICIOS EN LINEA

```
public int[] shiftLeft(int[] nums) {
	if(nums.length==0){
		return nums;
	}
	int aux=nums[0];
	for(int i=0;i<nums.length-1;i++){
		nums[i]=nums[i+1];
	}
	nums[nums.length-1]=aux;
	return nums;
}
```

O(n)=n

```
public boolean has22(int[] nums) {
	for(int i=0;i<nums.length-1;i++){
		if(nums[i]==2&&nums[i+1]==2){
			return true;
		}
	}
	return false;
}
```

O(n)=n

```
public boolean only14(int[] nums) {
  for (int i = 0; i<nums.length ; i++)
    if(nums[i] !=4 && nums [i] != 1)
```

```
    return false;
    return true;
}
```

O(n)= n

```
public boolean more14(int[] nums) {
  int cont1=0;
  int cont4=0;
  for (int i= 0; i<nums.length ; i++){
    if(nums[i]==1){
      cont1++;
    }
    if(nums[i]==4){
      cont4++;
    }
  }
  return cont1> cont4;
}
```

O(n)= n+c

```
public boolean has77(int[] nums) {
  for (int j=0 ;j<nums.length-1 ;j++)
    if(nums[j]==7 && nums[j+1]==7 )return true;

    for(int j=0; j<nums.length -2 ; j++)
    if(nums[j]==7 && nums[j+2]==7) return true;
    return false;
```

O(n)= (n-1) +(n-2) +C1

```
}
```

ARRAY3

```
public int[] fix34(int[] nums) {
    for (int j = 0; j < nums.length; j++)
        if (nums[j] == 3) {
            int rev = nums[j + 1];
            nums[j + 1] = 4;
            for (int k = j + 2; k < nums.length; k++)
                if (nums[k] == 4) nums[k] = rev;
        }
    return nums;
}
```

COMPLEJIDAD: O(N)=n

```
public boolean canBalance(int[] nums) {
    for (int k = 0; k< nums.length; k++) {
        int suma = 0;
        for (int j = 0; j < k; j++) suma += nums[j];
        for (int j = k; j < nums.length; j++) sum -= nums[j];
        if (sum == 0) return true;
    }
    return false;
}
```

3. SIMULACRO DE PREGUNTAS DE SUSTENTACION DE PROYECTOS

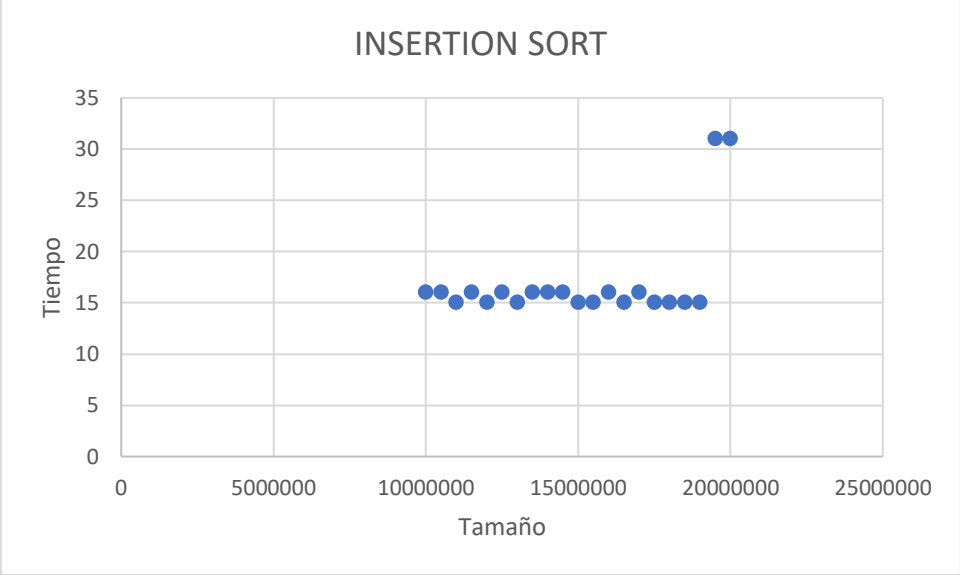
3.1

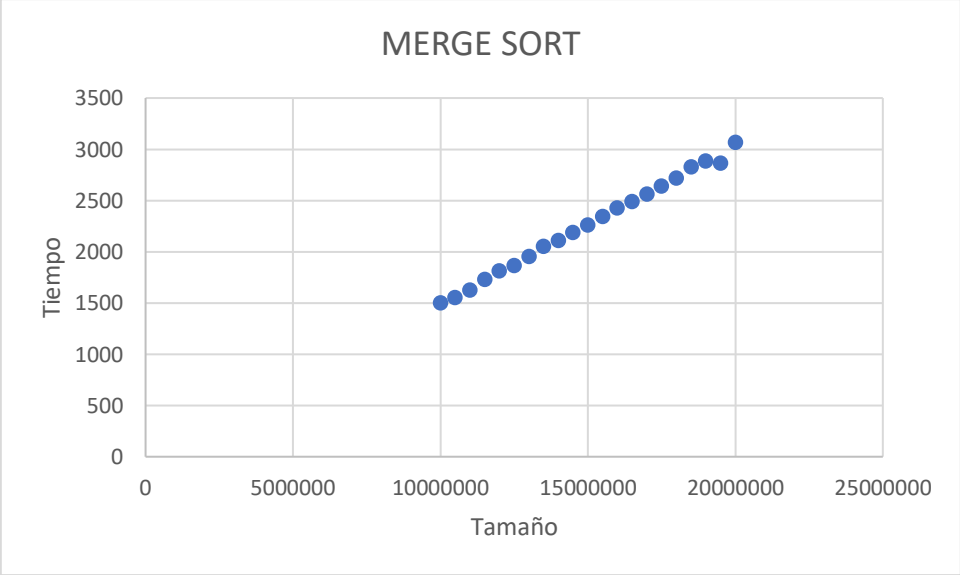
INSERTION SORT	
TAMAÑO	TIEMPO
10000000	16
10500000	16
11000000	15

11500000	16
12000000	15
12500000	16
13000000	15
13500000	16
14000000	16
14500000	16
15000000	15
15500000	15
16000000	16
16500000	15
17000000	16
17500000	15
18000000	15
18500000	15
19000000	15
19500000	31
20000000	31

MERGE SORT	
TAMAÑO	TIEMPO
10000000	1500
10500000	1551
11000000	1625
11500000	1727
12000000	1813
12500000	1864
13000000	1953
13500000	2053
14000000	2109
14500000	2184
15000000	2259
15500000	2342
16000000	2426
16500000	2486
17000000	2559
17500000	2638
18000000	2718
18500000	2826
19000000	2884
19500000	2860
20000000	3065

3.2





3.3

Es mucho más eficiente usar insertionSort ya que su tiempo en milisegundos siempre ronda entre los 15 y los 16 en cambio el de mergSort es un número muy alto que cada vez crece más. Esto significa que para un arreglo con millones de elementos es más efectivo usar insertionSort que ahorra más tiempo y memoria que mergeSort.