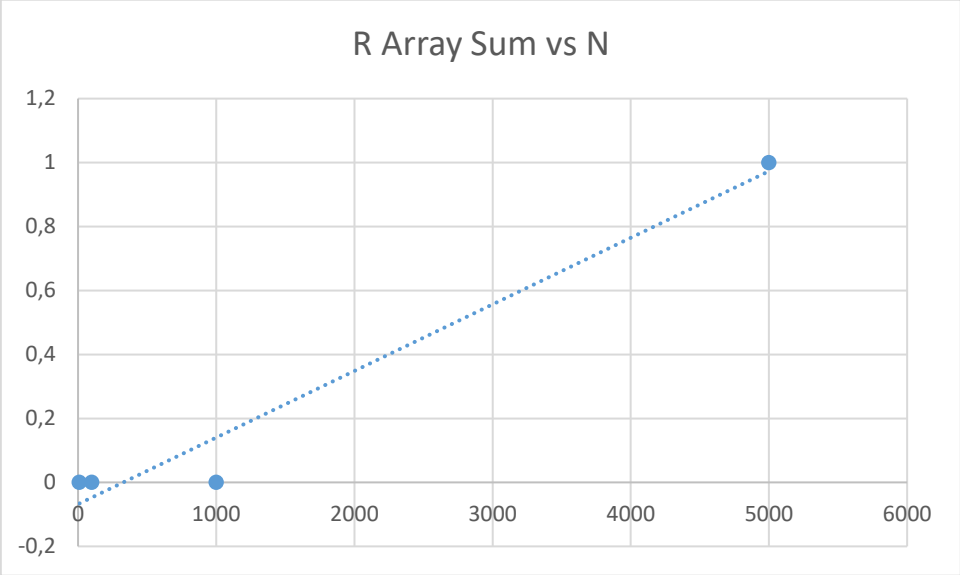
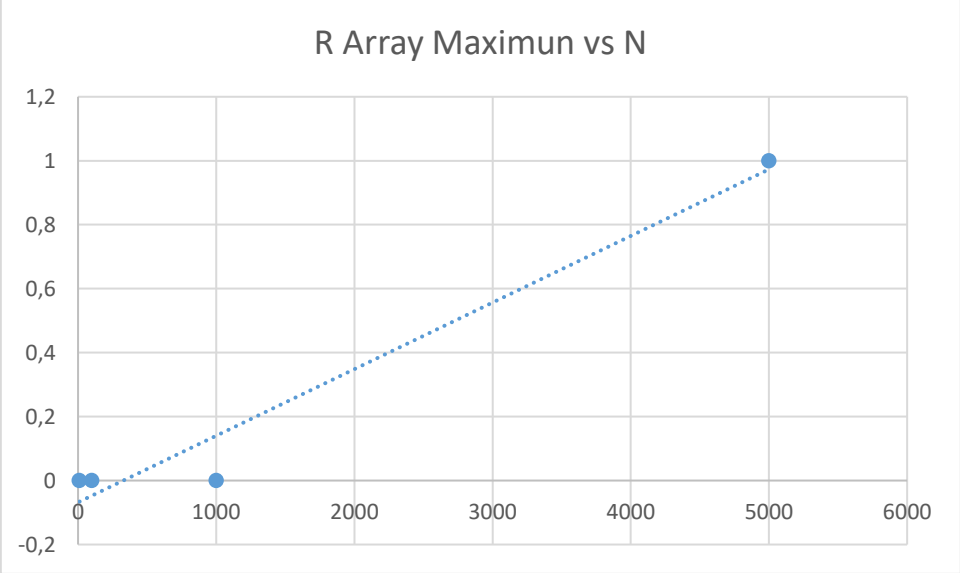
3.1)

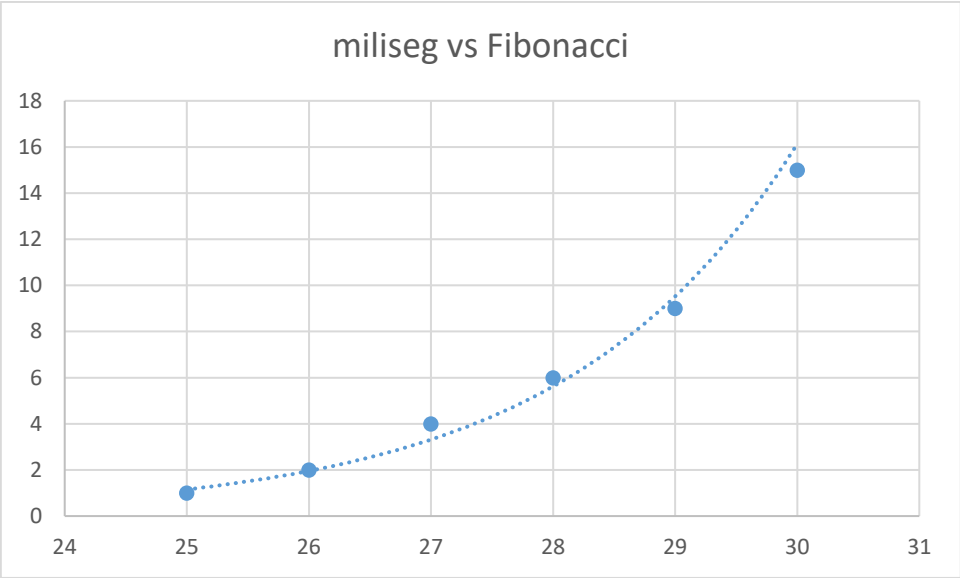
N	10	100	1000	5000
R Array Sum	0	0	0	1
R Array Maximun	0	0	0	1

R Fibonacci	25	26	27	28	29	30
miliseg	1	2	4	6	9	15

3.2=





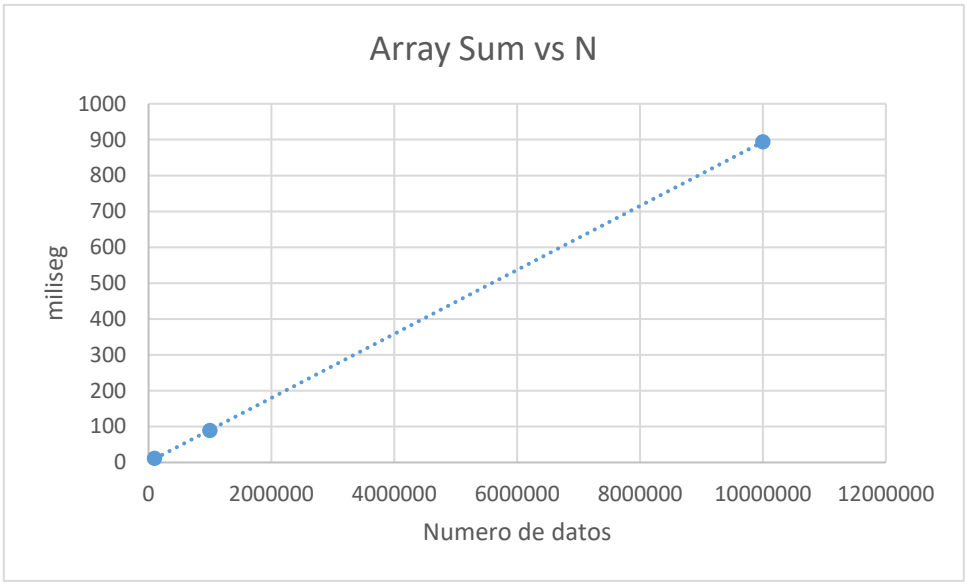


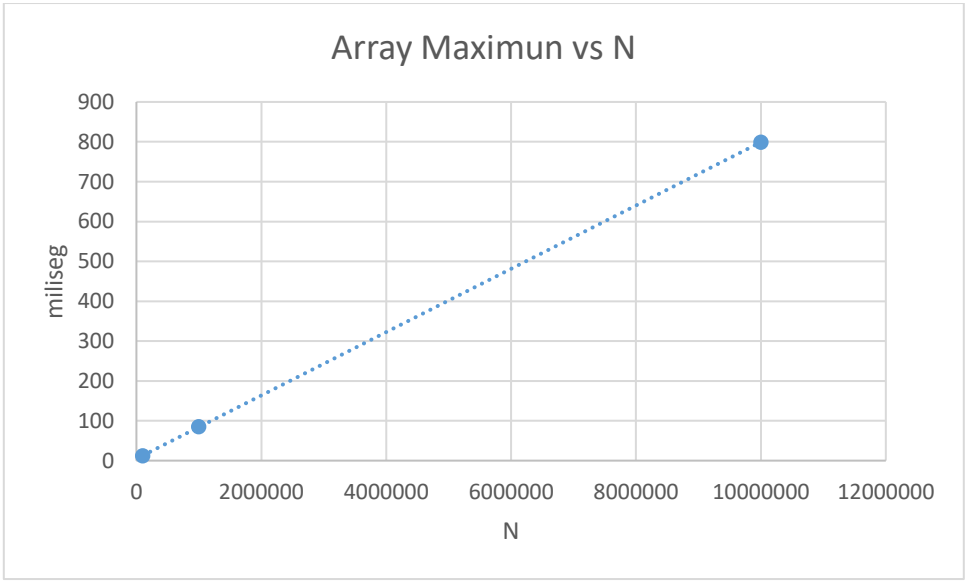
3.3) los tiempos son bastante cortos, para 5000 datos se demora tan solo 1 milisegundo (para sum y maximun) ya que son log(n) pero para fibonacci es (n^2) por lo que es más demorado, y esto se evidencia en las gráficas, además el array sum y el array maximun solamente pueden manejar arreglos pequeños

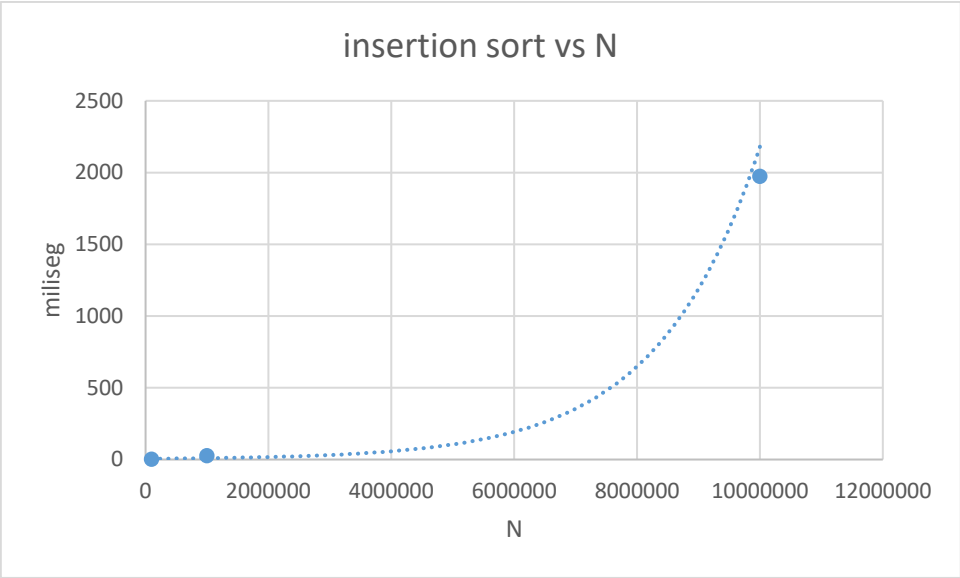
3.4)

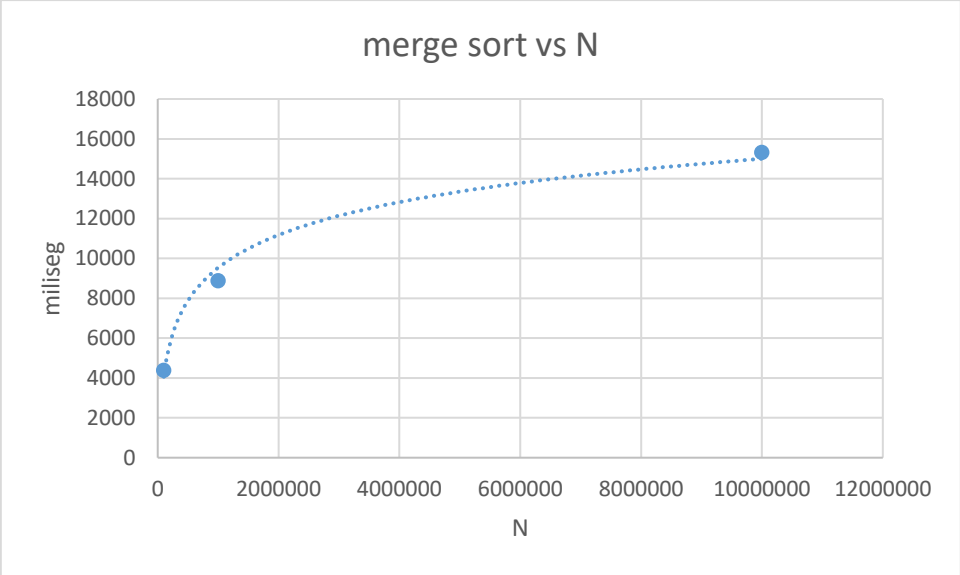
N	100000	1000000	10000000	100000000
Array Sum	12	89	894	out of memory
Array Maximun	12	85	799	out of memory
insertion sort	2	28	1974	error
merge sort	4385	8876	15325	más de 5 minutos

3.5)


Array Sum vs N
1000
900
800
700
600
500
400
300
200
100
0
miliseg
0
2000000
4000000
6000000
8000000
10000000
12000000
Numero de datos


Array Maximun vs N
900
800
700
600
500
400
300
200
100
0
miliseg
0
2000000
4000000
6000000
8000000
10000000
12000000
N





3.6

insertion $O(n^{2)}$
merge O(n log n)
Sum y max O(n)

Que los milisegundos obtenidos son bastante acertados con la complejidad del algoritmo

3.7)

para valores grandes insertion sort aumenta exponencialmente, es decir que entre más grande sea N se va a demorar mucho mas

3.8)

como insertion sort tiene una complejidad de $O(n^2)$ y array sum tiene una de O(n) para valores grandes Array sum se va a demorar proporcionalmente lo mismo, ya que la tendencia es lineal, y no crece tan rápido ya que el insertion tiene la complejidad de n^2 por lo que cuando aumenta N se demora cada vez mucho más

3.9)

Para arreglos muy grandes (si se tiene memoria para estos) el merge sort se demora proporcionalmente menos cada N+1, es decir que entre más crece N proporcionalmente menos se demora el merge sort para realizarce, mientras que el insertion sort se demora exponencialmente mucho más cada vez que aumenta N, pero para arreglos pequeños el insertion es mas eficiente, ya que al ser pequeños la forma exponencial no es muy demorada, mientras que la logarítmica si lo es

3.10)

MaxSpan lo que hace es correr todo el arreglo en busca de algún 3, y cuando lo encuentra guarda el número siguiente al 3 en una nueva variable y lo remplaza por 4, después de esto recorre el arreglo 2 posiciones después de donde estaba el 3 hasta el final en busca de un 4, y cuando lo encuentra remplaza el 4 por el numero guardado anteriormente.

3.11)

```
public int countEvens(int[] nums) { // C0
  int count = 0; //C1
  for (int i=0;i<nums.length;i++) { //T(n*C2)
     if (nums[i]%2 == 0){  T(n*C3)
     count = count + 1; //T(n*C4)
     }
  }
 return count;//C5
}
```

T(n)=c0+c1+c5+n(c2+c3+c4)=O(n)

Siendo n el número de datos en el arreglo

```
public int bigDiff(int[] nums) { //C0
  int max = nums[0];//C1
  int min = nums[0];//C2
  for (int i = 0; i<nums.length; i++){ //T(n*C3)
   if (min > nums[i]){ //T(n*C4)
     min=nums[i]; //T(n*C5)
   }
   if(max < nums[i]){ //T(n*C6)
     max = nums[i]; //T(n*C7)
   }
  }
   return max-min; //C8
}
```

T(n)=C0+C1+C2+C8+n(C3+C4+C5+C6+C7) = O(n)

Siendo n el número de datos en el arreglo

```
public int centeredAverage(int[] nums) { //C0
  int max = nums[0];//C1
  int min = nums[0];//C2
  int sum = 0; //C3
  for (int i = 0; i<nums.length; i++) //T(n*C4)
    if(max < nums[i]){ //T(n*C5)
    max = nums[i]; //T(n*C6)
    }
    if(min > nums[i]){ //T(n*C7)
    min = nums[i]  ; //T(n*C8)
    }
    sum = sum + nums[i]; //T(n*C9)
 }
 sum = sum - (min + max); //C10
 return sum /(nums.length-2);//C11
}
```

T(n) =c0+c1+c2+c3+c10+c11+n(C4+C5+C6+C7+C8+C9) = O(n)

Siendo n el número de datos en el arreglo

```
public int sum13(int[] nums) { //C0
  int sum = 0;//C1
  if(nums.length==0){ //C2

  }else if(nums[0] != 13){ //C3
  sum = nums[0]; //C4
  }
  for(int i=1 ; i<nums.length; i++){ // T(n*C5)
    if(nums[i] != 13 && nums[i-1] != 13){ // T(n*C6)
      sum = sum + nums[i]; // T(n*C7)
    }
  }
  return sum; //C8
}
```

T(n) = C0+C1+C2+C3+C4+C8+n(C5+C6+C7) = O(n)

Siendo n el número de datos en el arreglo

```
public boolean has22(int[] nums) { //C0
  boolean wierd = false; //C1
  for(int  i = 1;i<nums.length;i++){ //T(n*C2)
    if (nums[i] == 2 && nums[i-1] == 2) //T(n*C3)
    wierd = true; //T(n*C4)
  }
  return wierd;//C5
}
```

$T(n) = C0+C1+C5+n(C2+C3+C4)=O(n^2)$

Siendo n el número de datos en el arreglo

```
public int maxSpan(int[] nums) { //C0
  int max = 0; //C1
  int relative =0; //C2
  if (nums.length == 0){ //C3
    max = -1; //C4
  }
  for(int i=0;i<nums.length;i++){ //T(n*C5)
    for(int j=i;j<nums.length;j++){//T(n*n*C6)
      if(nums[i] == nums [j]){  //T(n*n*C7)
        relative = j-i; //T(n*n*C8)
        if(max < relative) //T(n*n*C9)
        max = relative; //T(n*n*C10)
      }
    }
  }
   return max + 1; //C11
}
```

$T(n) = (C0+C1+C2+C3+C4+C11+n(C5)+n^2(C6+C7+C8+C9+C10)) = O(n^2)$

Siendo n el número de datos del arreglo

```
public int[] fix34(int[] nums) { //C0
  int save = 0;//C1
  for (int i = 0;i<nums.length;i++){ //T(n*C2)
    if(nums[i] == 3){ //T(n*C3)
      save = nums[i+1]; //C4
      nums[i+1] = 4; //C5
      for(int j = i+2;j<nums.length;j++){ //T(n*n*C6)
       if(nums[j] == 4){ //T(n*n*C7)
         nums[j] = save; //C8
       }
      }
    }
  }
  return nums; //C9
}
```

$T(n) = C0+C1++C4+C5+C8+C9+n(C2+C3)+n^2(C6+C7) =O(n^2)$

Siendo n el número de datos del arreglo

```
public int[] fix45(int[] nums) { //C0
  int save = 0; //C1
  int depui = 0; //C2
  for (int i = 0;i<nums.length;i++){ //T(n*C3)
    if(nums[i] == 5 && i == 0 || nums[i] == 5 && nums[i - 1] != 4){ //T(n*C4)
      depui = i;//C5
      for(int j = 0;j<nums.length;j++){ //T(n*n*C6)
       if(nums[j] == 4 && nums[j+1] !=5){ //T(n*n*C7)
         save = nums[j+1]; //C8
         nums[j+1] = 5; //C9
         nums[depui] = save; //C10
         break; //C11
       }
      }
    }
  }
  return nums; //C12
}
```

$T(n)=C0+C1+C2+C5+C8+C9+C10+C11+n(C3+C4)+n^2(C6+C7) =O(n^2)$

Siendo n el número de datos del arreglo

```
public boolean canBalance(int[] nums) {//C0
  boolean isit = false;//C1
  for(int i = 0; i<nums.length; i++){ //T(n*C2)
    int sum = 0;//T(n*C3)
    for(int j = 0; j < i;j++){ //T(n*n*C4)
      sum = sum + nums[j]; //T(n*n*C5)
    }
    for(int k = i; k<nums.length;k++){ //T(n*n*C6)
      sum = sum -nums[k]; //T(n*n*C7)
    }
      if (sum ==0){ //T(n*C8)
      isit = true; //T(n*C9)
  }


  }
  return isit; //C10
}
```

$T(n)=C0+C1+C10+n(C2+C3+C8+C9)+n^2(C4+C5+C6+C7) = O(n^2)$

Siendo n el número de datos en el arreglo

```
public boolean linearIn(int[] outer, int[] inner) { //C0
   int i = 0; //C1
   int j = 0; //C2
   while (i < inner.length && j < outer.length) {T(n*C3)
      if (outer[j] == inner[i]) { T(n*C4)
         j++;//C5
         i++;//C6
      } else j++; T(n*C7)
   }
   return (i == inner.length);//C8
}
```

T(n)=C0+C1+C2+C5+C6+C8+n(C3+C4+C7) = O(n)

Siendo n el número de datos del arreglo más grande

4)

Punto1) C

Punto2) B

punto3) C

punto4) B

punto5) D

punto6) A

punto7) $T(n) = C0+C1+C2+T(n-1)$ && $O(n)$