Laboratory practice No. 2: Big O notation

Cristian Camilo Rendón Cardona
Universidad EAFIT
Medellín, Colombia
crendo11@eafit.edu.co

Jhesid Steven Suarez Berrio
Universidad EAFIT
Medellín, Colombia
jssuarezb@eafit.edu.co

March 4, 2018

1) ONLINE EXERCISES (CODINGBAT)

For the complexity calculation of the online exercises, we assume each line that is not a loop as a constant ci and the summation gives as result another constant

1.a. Array II

i.
```
    public boolean either24(int[] nums) {
  int a=0;                                      //c1
  int b=0;                                      //c2
  for (int i =0;i<nums.length-1;i++){           //c3 * n
    if((nums[i]==nums[i+1])&&(nums[i]==2)){     //c4 * n
      a=1;                                      //c5 * n
    }
    else if((nums[i]==nums[i+1])
    &&(nums[i]==4)){                            //c6
      b=1;                                      //c7
    }
  }
  if((a==1)&&(b==1)){                           //c8
    return false;                               //c9
  }
  else if((a==1)||(b==1)){                      //c10
    return true;                                //c11
  }
  return false;                                 //c12
}
```

Then the complexity of the algorithm is given by

$$T(n) = C_1 + C_2 * n \therefore is\, O(n) \tag{1}$$

ii.

```
public boolean only14(int[] nums) {
  boolean a = true;                               //c1
  for (int i = 0; i < nums.length; i++) {         //c2 * n
    if ((nums[i]!=1)&&(nums[i]!=4)){              //c3 * n
      a = false;                                  //c4 * n
      break;                                      //c5 * n
    }
  }
  return a;                                       //c6
}
```

Then the complexity of the algorithm is given by

$$T(n) = C_1 + C_2 * n \therefore is\, O(n) \tag{2}$$

iii.

```
public boolean tripleUp(int[] nums) {
  boolean a=false;                                //c1
  for(int i=0;i<nums.length-2;i++){               //c2 * n
    if((nums[i]==(nums[i+1]-1))                   //c3 * n
    &&(nums[i+1]==(nums[i+2]-1))){                //c4 * n
      a=true;                                     //c5 * n
    }
  }
  return a;                                       //c6
}
```

Then the complexity of the algorithm is given by

$$T(n) = C_1 + C_2 * n \therefore is\, O(n) \tag{3}$$

iv.

```
public String[] fizzArray2(int n) {
  String[] a=new String[n];                       //c1
  for(int i=0;i<a.length;i++){                    //c2 * n
    a[i]=String.valueOf(i);                       //c3 * n
  }
  return a;                                       //c4
```

Profoessor Mauricio Toro Bermudez
Phone: (+57)(4)2619500 Ext. 9473. Office: 19 – 627
E-mail: mtorobe@eafit.edu.co

```
}
```

Then the complexity of the algorithm is given by

$$T(n) = C_1 + C_2 * n \therefore is\, O(n) \tag{4}$$

v.
```
public int[] fizzArray(int n) {
  int[] a=new int[n];                          //c1
  for(int i=0;i<a.length;i++){                 //c2 * n
    a[i]=i;                                    //c3 * n
  }
  return a;                                    //c4
}
```

Then the complexity of the algorithm is given by

$$T(n) = C_1 + C_2 * n \therefore is\, O(n) \tag{5}$$

1.b. Array III

i.
```
public int countClumps(int[] nums) {
  int cont = 0;                                //c1
  if (nums.length != 0){                       //c2
    int temp;                                  //c3
    temp = nums[0];                            //c4
    boolean b = false;                         //c5
    for (int i = 1; i < nums.length; i++){     //c6 * n
      if ((temp == nums[i]) && (b == false)){  //c7
        cont++;                                //C8
        b = true;                              //c9
      }
      if (nums[i] != temp){                    //c10
        temp = nums[i];                        //c11
        b = false;                             //c12
      }
    }
  }
  return cont;                                 //c13
}
```

Then the complexity of the algorithm is given by

$$T(n) = C_1 + C_2 * n \therefore is\, O(n) \tag{6}$$

ii.

```
public int[] squareUp(int n) {
  int[] A = new int[n*n];                    //c1
  for (int i = 0; i < n; i++){               //c2 * n
    for (int j = 1;j <= i + 1; j++){         //(c3 * n) * n
    A[(n*(i+1))-j] = j;                      //(c4 * n) * n
            }
  }
return A;                                    //c5
}
```

Then the complexity of the algorithm is given by

$$T(n) = C_1 + C_2 n + C_3 n^2 \therefore is\, O(n^2) \tag{7}$$

iii.

```
public boolean linearIn(int[] outer, int[] inner) {
  int comp;                                  //c1
  int cont = 0;                              //c2
  boolean entro;                             //c3
  for (int i = 0; i < inner.length; i++){    //c4 * n
    comp = inner[i];                         //c5 * n
    entro = false;                           //c6 * n
    for (int j = 0; j < outer.length; j++){ //(c7 * n) * n
      if (comp == outer[j] && !entro){       //c8 * n^2
        cont++;
        entro = true;                        //c10 * n^2
        }
    }
  }
  return cont == inner.length;               //c11
    }
```

Then the complexity of the algorithm is given by

$$T(n) = C_1 + C_2 n + C_3 n^2 \therefore is\, O(n^2) \tag{8}$$

iv.

```
public int maxSpan(int[] nums) {
  int cont = 0;                              //c1
  if ((nums.length != 0)
  &&(nums[0] == nums[nums.length-1])){       //c2
    cont++;                                  //c3
```

```
    }
    for (int i = 0; i < nums.length - 1; i++){    //c4 * n
      cont++;                                     //c5 * n
    }
    return cont;                                  //c6
  }
```

Then the complexity of the algorithm is given by

$$T(n) = C * n \therefore is\, O(n) \tag{9}$$

v.

```
    public int[] seriesUp(int n) {
      int[] A = new int[n*(n+1)/2];               //c1
      int i = n*(n+1)/2;                          //c2
      while (i > 0){                              //c3 * n
        for (int j = n; j > 0; j-- ){             //(c4 * n) * n
          A[i - (n -j + 1)] = j;                  //c5 * n^2
        }
        n--;                                      //c6 * n
        i = n*(n+1)/2;                            //c7 * n
      }
      return A;                                   //c8
    }
```

Then the complexity of the algorithm is given by

$$T(n) = C_1 + C_2 n + C_3 n^2 \therefore is\, O(n^2) \tag{10}$$

2) EXECUTION TIME GRAPHS

2.a. Recursive version times

Table 1 shows the execution time for the recursive version of ArraySum, ArrayMax and Fibonacci

	N = 100.000	N = 1'000.000	N = 10'000.000	N = 100'000.000
R Arraysum	5	16,12	104	More than1 Min
R Arraymax	4	36	128	More than1 Min
R Fibonacci	More than 1 Min	More than,1 Min	More than,1 Min	More than1 Min

Table 1: Execution time for recursive algorithms (in ms)

2.b. Recursive version graphs

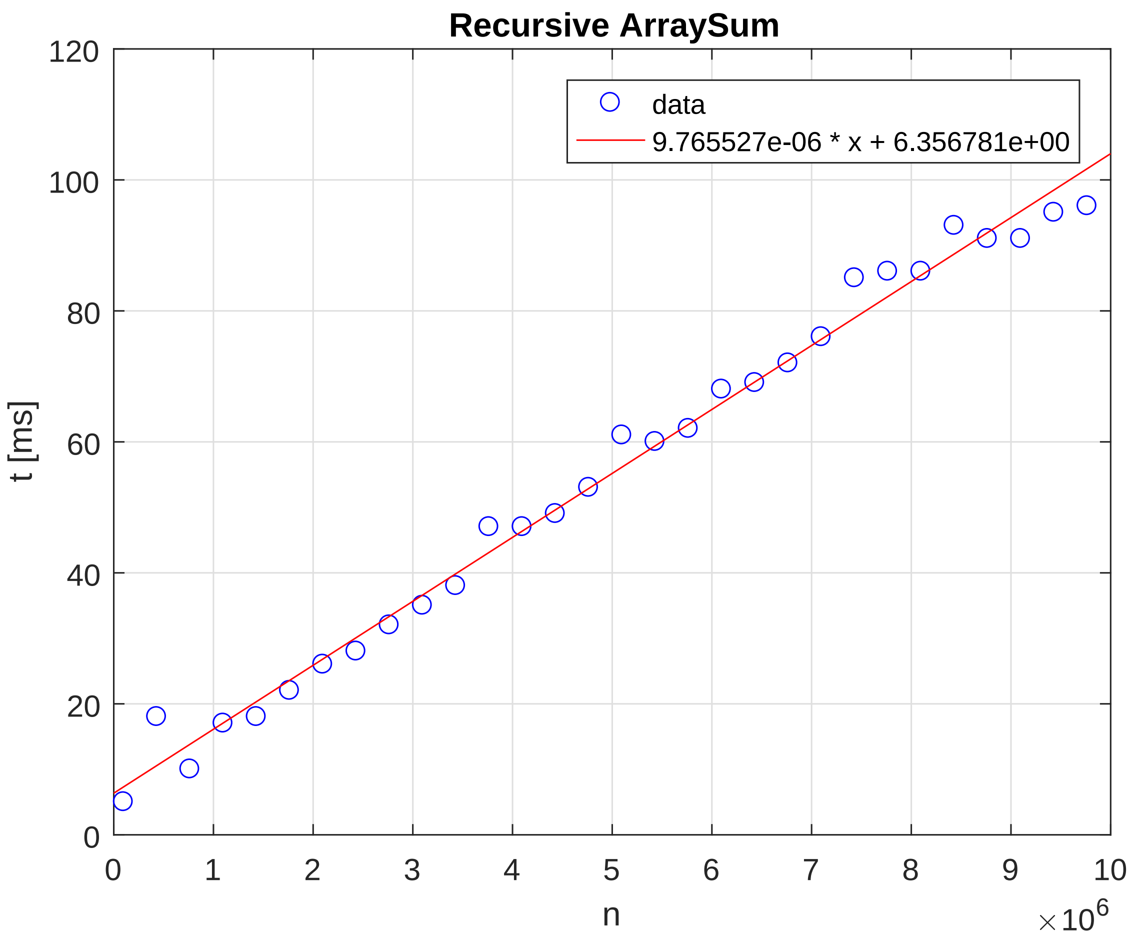

Figure 1: Arraysum graph with linear fitting

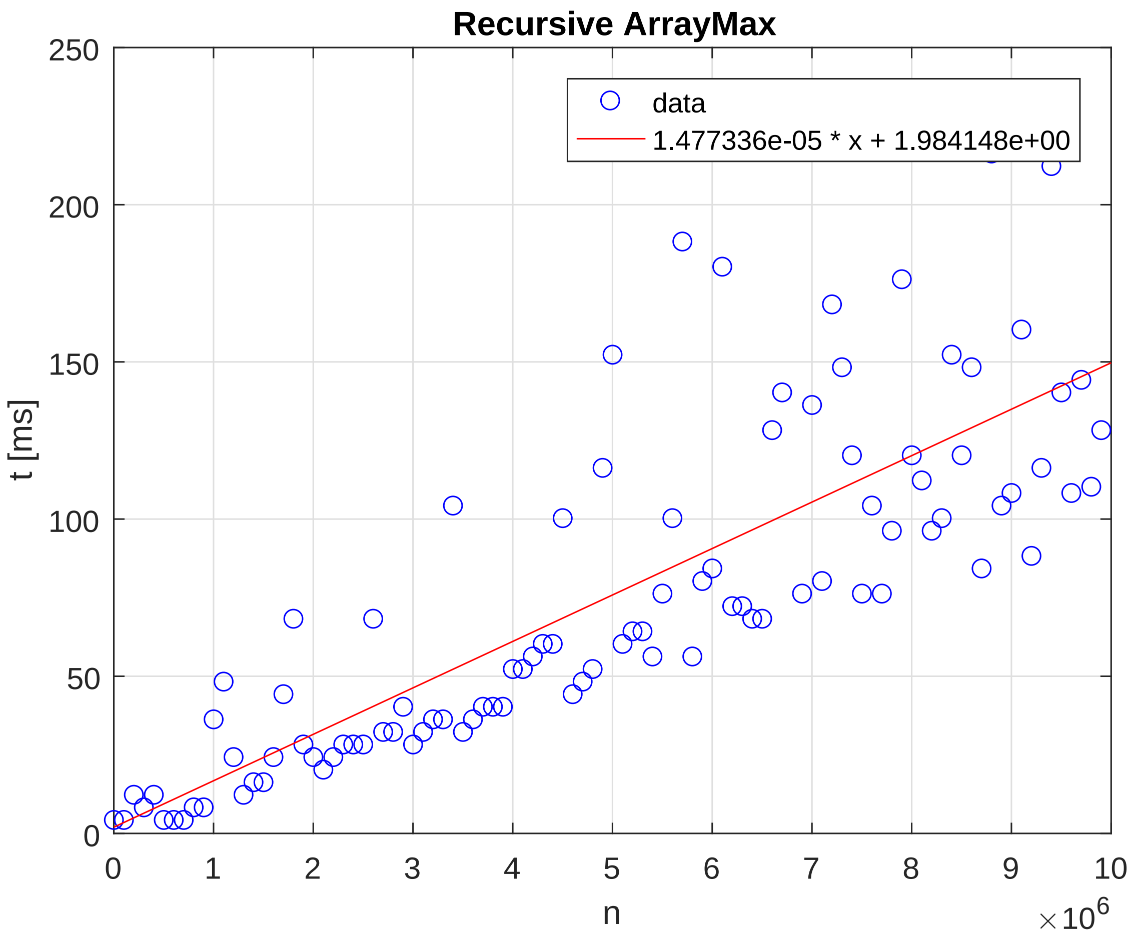

Figure 2: ArrayMax graph with linear fitting

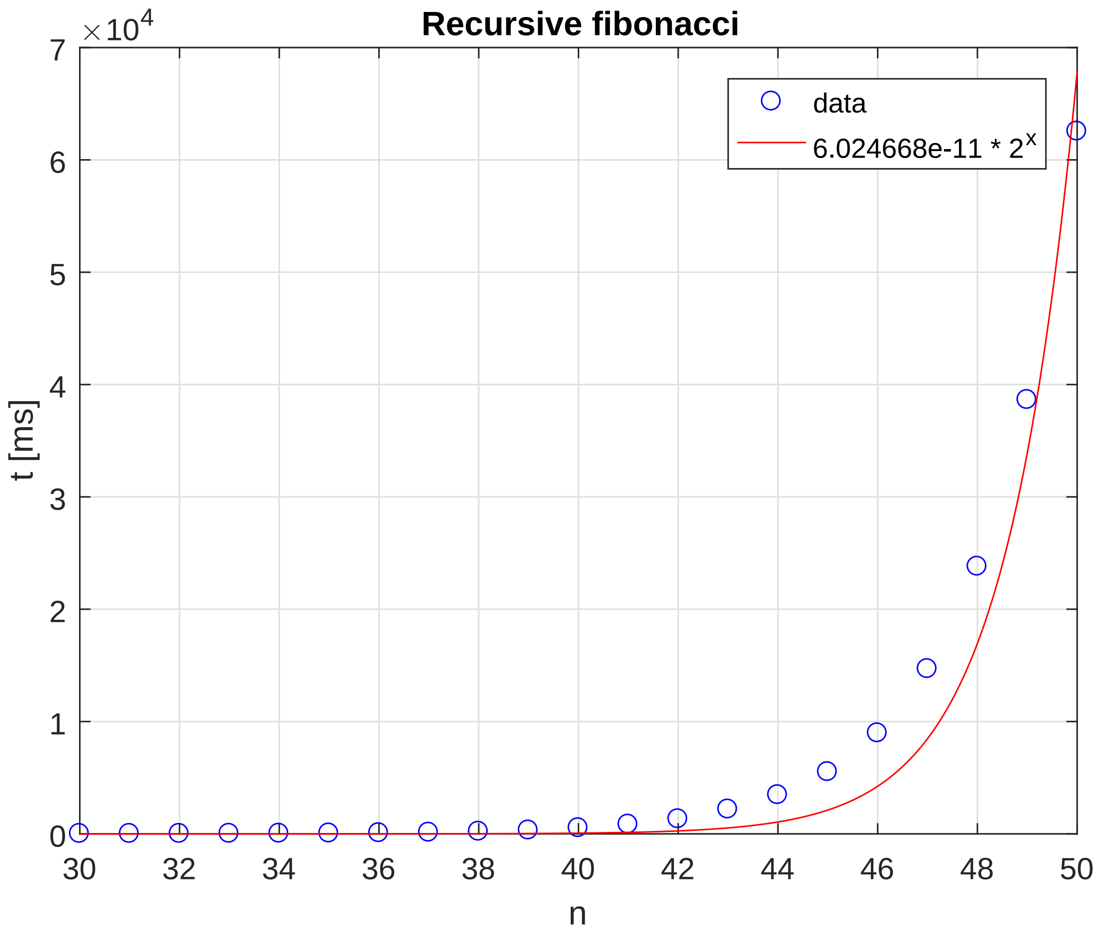

Figure 3: Fibonacci graph with exponential fitting

2.c. *What did you learn with respect obtained times in section 2.1 and theoretical results?*

As expected the behaviour of the algorithms tends to be as calculated with some error generated by the machine which is running it.

2.d. *NON recursive version times*

	N = 100.000	N = 1'000.000	N = 10'000.000	N = 100'000.000
R Arraysum	1	1	4	38
R Arraymax	2	2	4	37
Insertion Sort	1234	more than 5 Min	more than 5 Min	more than 5 Min

Table 2: NON recursive times

2.e. *NON Recursive version graphs*

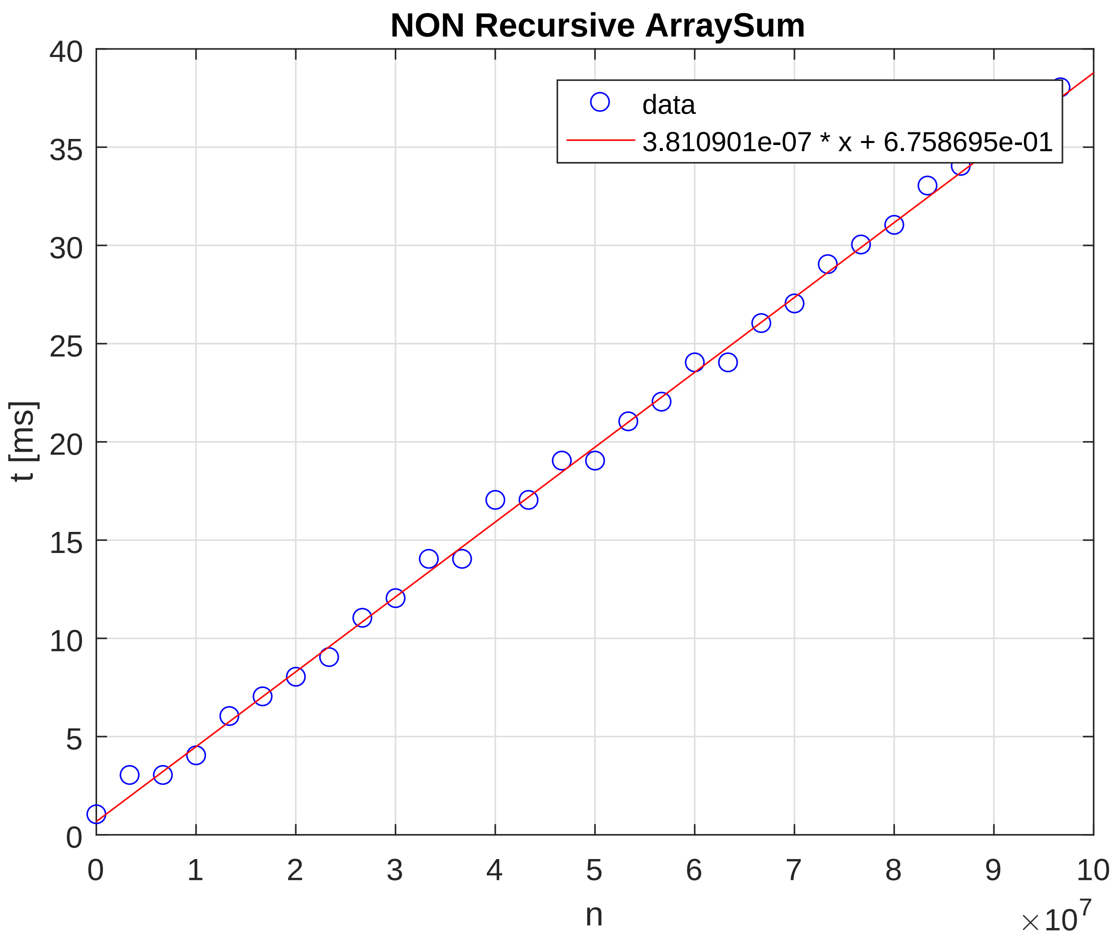

Figure 4: Arraysum graph with linear fitting

Profoessor Mauricio Toro Bermudez
Phone: (+57)(4)2619500 Ext. 9473. Office: 19 – 627
E-mail: mtorobe@eafit.edu.co

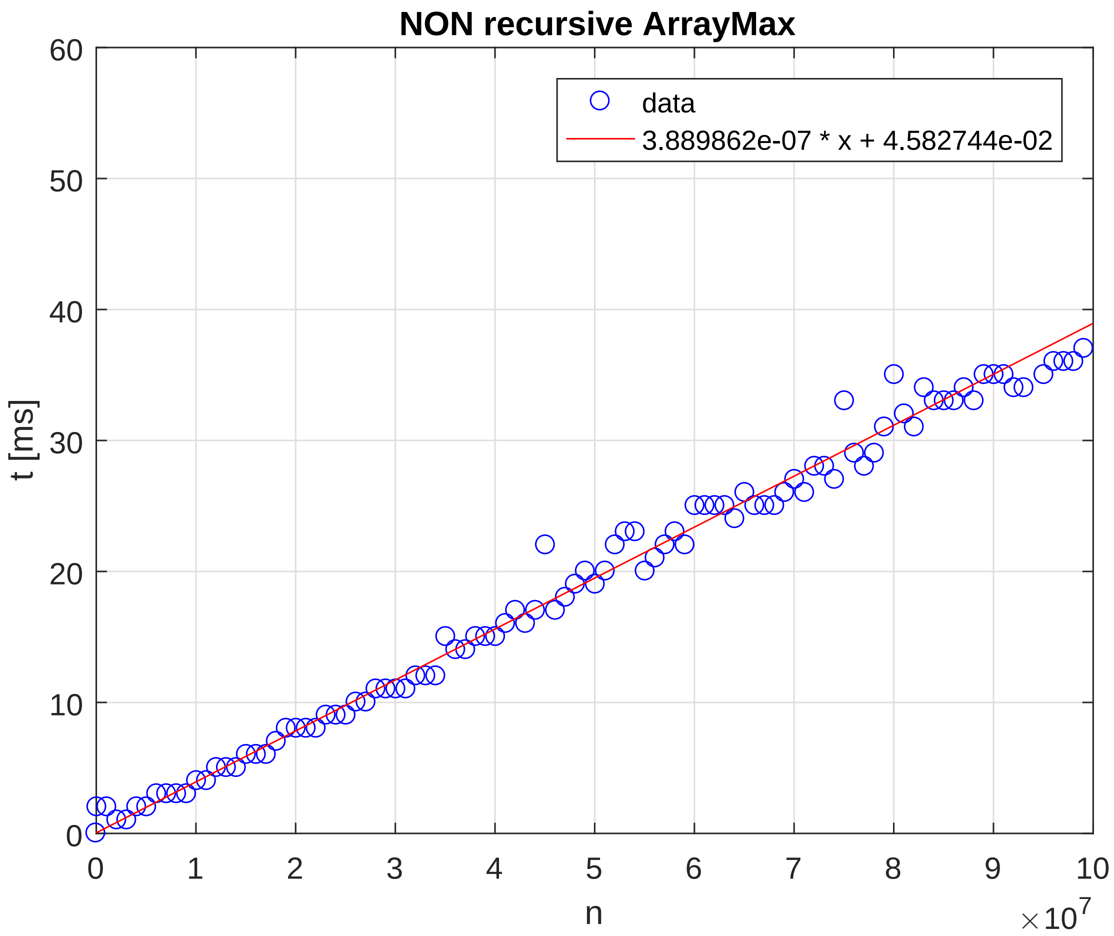

Figure 5: ArrayMax graph with linear fitting

Profoessor Mauricio Toro Bermudez
Phone: (+57)(4)2619500 Ext. 9473. Office: 19 – 627
E-mail: mtorobe@eafit.edu.co

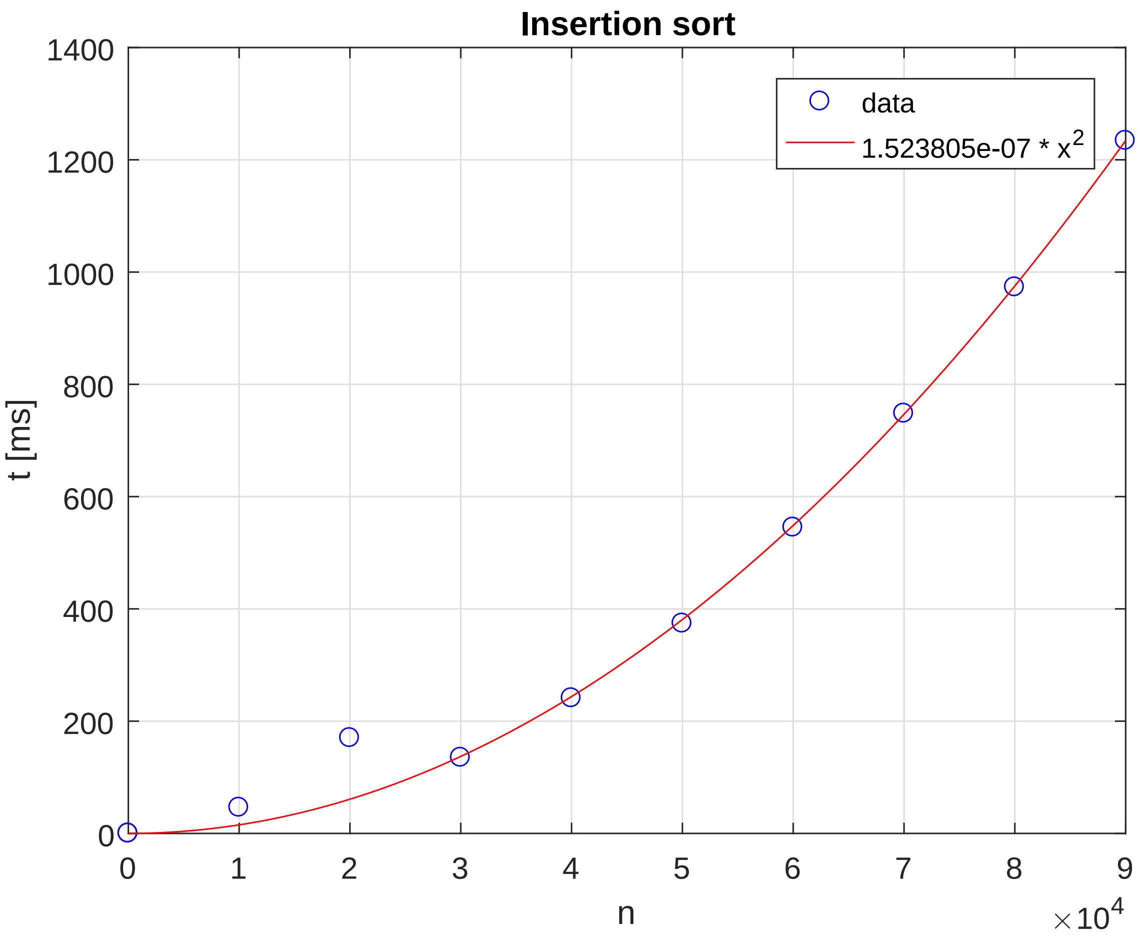

Figure 6: Insertion sort graph with x^2 fitting

2.f. What did you learn with respect obtained times in section 2.4 and O notation?

Notation O shows a nice approximation to the behaviour of the algorithm with, but for more accuracy in the equation of execution time is better to use the experimental results.

2.g. What happen with insertion sort for big N values?

insertion sort presents complexity of $O(n^2)$ so every time that n increases, the execution time is squared.

2.h. What happen with ArraySum for big N values?

In the case of ArraySum, due to its complexity $O(n)$, the increasing of execution time is linear and that is why it don't do it faster as Insertion sort

2.i. How does maxSpan work

We implemented an algorithm which works in a different way than the solution of the problem. The inputs of the function are always arrays which their last number is greater or equal to the first. When the first case happens, the algorithm returns the size of the array minus one. But when the last number is equal to the first, the algorithm returns the size of the array. The implementation can be seen in section 1.2.**iv.**. A advantage of the implementation is that its complexity is $O(n)$ instead of $O(n^2)$. A disadvantage is that the implementation was applied empirically so it could not work for a spacial case.

2.j. What does 'n' mean in the calculation of complexity?

In the calculation of complexity n is the times that a loop is executing and in the worst case in all of the complexities calculated n was equal to N (the size of the complexity problem).

3) EXAM SIMULATION ANSWERS

i. d

ii. b

iii. d

iv. b

v. d

vi. a

vii. **1.** $c * n$ **2.** $O(n)$

Profoessor Mauricio Toro Bermudez
Phone: (+57)(4)2619500 Ext. 9473. Office: 19 – 627
E-mail: mtorobe@eafit.edu.co