Laboratory practice No. 2: Big O Notation

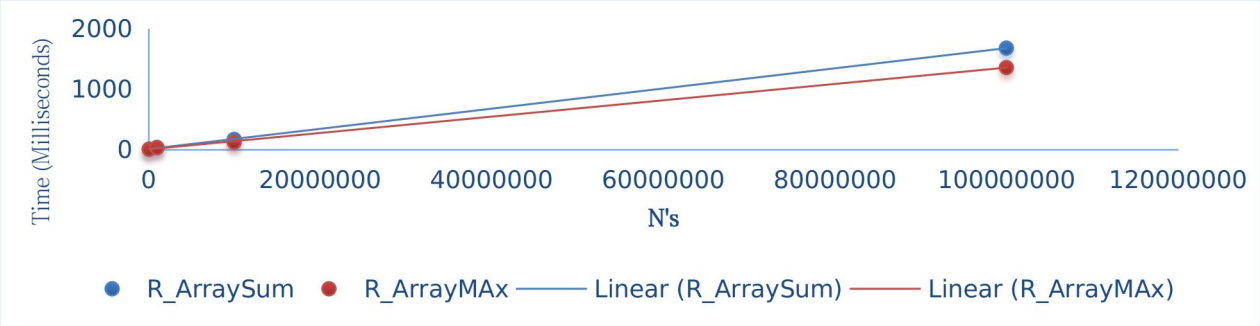
Kevin Gutierrez Gomez
Universidad Eafit
Medellín, Colombia
kgutierreg@eafit.edu.co

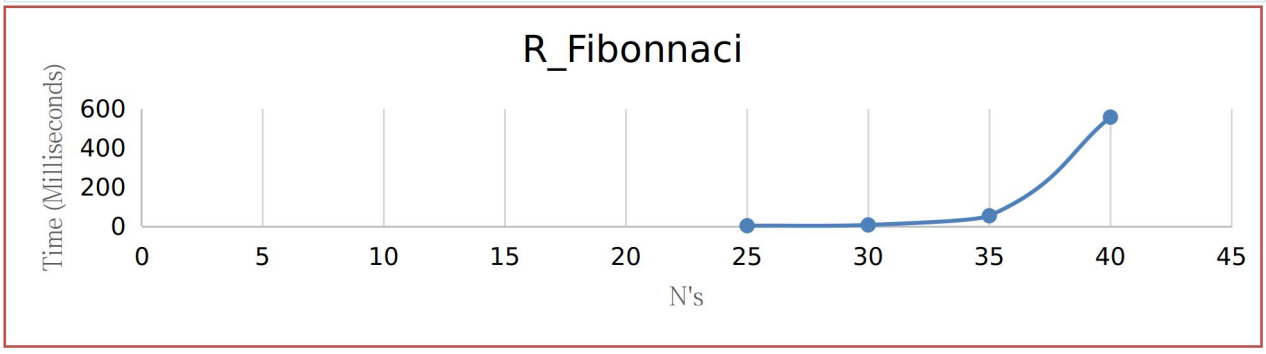
Jose Villamizar Peñaranda
Universidad Eafit
Medellín, Colombia
javillamip@eafit.edu.co

3.1

	N = 100.000	N= 1’000.000	N= 10’000.000	N = 100’000.000
R_ArraySum	**4(ms)**	**28(ms)**	**230(ms)**	**1797(ms)**
R_ArrayMax	**3(ms)**	**23(ms)**	**209(ms)**	**1345(ms)**
R_Fibonnaci	**N = 25 (2ms)**	**N = 30 (6ms)**	**N = 35 (53ms)**	**N = 40 (556ms)**

3.2





3.3

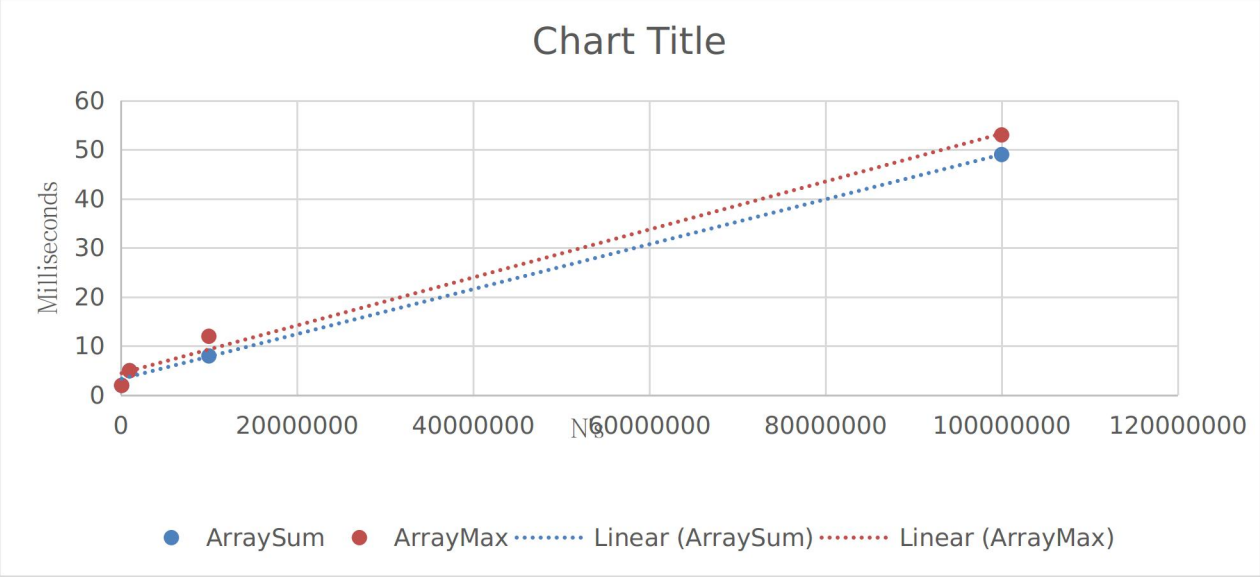
PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co

The experimental results agree with the theoretical ones obtained with the asymptotic notation.

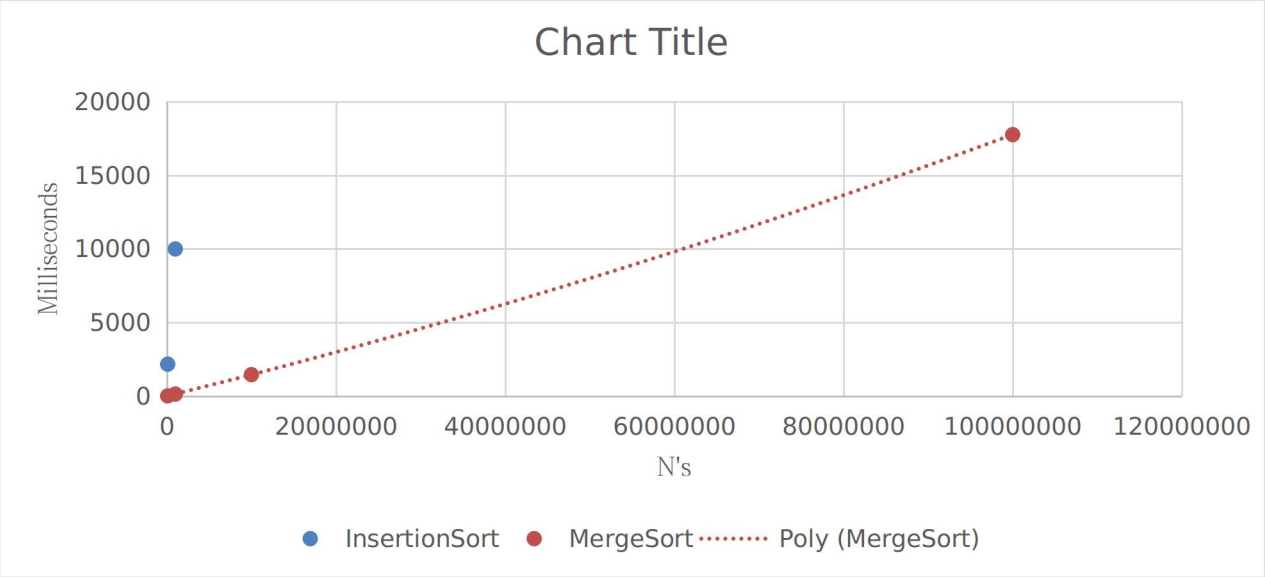
3.4

	N=100.000	***N=1'000.000***	***N=10'000.000***	***N=100'000.000***
ArraySum	***2***	***5***	***8***	***49***
ArrayMax	***2***	***5***	***12***	***53***
InsertionSort	***2178***	***More than one minute***	***More than one minute***	***More than one minute***
MergeSort	***27***	***148***	***1468***	***17757***

3.5



PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co



3.6
The experimental results agree with the theoretical ones obtained with the asymptotic notation.

3.7
The insertionSort algorithm becomes inefficient with large values, this can be explained with its complexity. O(n^2)

3.8
The ArraySum algorithm takes a while to process large values, however this is normal because its execution time is dependent to the size of the problem, the difference in execution time respect to the InsertionSort algorithm is very large, and this can be explained comparing the nature of its complexities. O(n) != O(n^2)

3.9
The MergeSort algorithm is more efficient for large values, however InsertionSort can be used without problems for sizes smaller than n = 100, demonstrating efficiency for these sizes.

3.10
The maxspan algorithm must calculate the maximum number of values that exist between an index of the array with value n and another index with the same value.

for this, an integer variables named "span" and "maxspan" are initialized; this algorithm acts in a similar way to insertionSort becuase with two nested cycles what it does is locate itself in an index and count how many values it finds until finding a value equal to the index, then compare if the "span" counter is greater than "maxspan" and if so, "maxspan" remains with the value of "span", followed by the index of the cycle in charge of traversing the array advance a position, once the entire array has been traversed, the variable "maxspan" is returned.

3.11

```
public int[] zeroFront(int[] nums) {
     int[] res      = new int[nums.length]; //C1
     int zeroPos    = 0;                    //C2
     int nonZeroPos = res.length - 1;       //C3

     for (int i = 0; i < nums.length; i++)  //C4*n
        if (nums[i] == 0)                   //C5*n
           res[zeroPos++]    = 0;           //C6*n
        else
           res[nonZeroPos--] = nums[i];

     return res;                            //C8
  }
Complexity:
    In the best case: T(n)=C8
    In the worst case: T(n)=n O(n)

  public int[] withoutTen(int[] nums) {
    int[] copy = new int[nums.length]; //C1
    int index = 0;                     //C2

    for (int i = 0; i < nums.length; i++)//C3*n
       if (nums[i] != 10) {            //C4*n
          copy[index] = nums[i];       //C5*n
          index++;                     //C6*n
       }
    return copy;                       //C7*n
}
Complexity:
    In the best case: T(n)=C7
    In the worst case: T(n)=n O(n)
```

```
public int[] zeroMax(int[] nums) {
    int lOdd = 0;                                          //C1
    for (int i = nums.length - 1; i >= 0; i--) {           //C2*n
        if (nums[i] % 2 == 1 && nums[i] > lOdd)            //C3*n
            lOdd = nums[i];                                //C3*n
        if (nums[i] == 0)                                  //C3*n
            nums[i] = lOdd;                                //C4*n
    }
    return nums;                                           //C5
}
```

Complexity:

In the best case: T(n)=C5

In the worst case: T(n)=n O(n)

```
public int[] evenOdd(int[] nums) {
    int[] narr   = new int[nums.length];                   //C1
    int evenPos = 0;                                       //C2
    int oddPos  = narr.length - 1;                         //C3

    for (int i = 0; i < nums.length; i++)                  //C4*n
        if (nums[i] % 2 == 0)                              //C5*n
            narr[evenPos++] = nums[i];                     //C6*n
        else
            narr[oddPos--]  = nums[i];
    return narr;                                           //C7
}
```

Complexity:

In the best case: T(n)=C7

In the worst case: T(n)=n O(n)

```
public String[] fizzBuzz(int start, int end) {
    String[] FB = new String[Math.abs(start-end)];   //C1
    int index = 0;                                   //C2

    for (int i = start; i < end; i++) {              //C3*n

        if (i % 3 == 0 && i % 5 == 0) FB[index] = "FizzBuzz"; //C4*n
```

```
      else if (i % 3 == 0) FB[index] = "Fizz";      //C5*n
      else if (i % 5 == 0) FB[index] = "Buzz";      //C6*n
      else FB[index] = String.valueOf(i);
      index++;                                      //C7*n
   }
   return FB;                                       //C8
}
```

Complexity:

In the best case: T(n)=C8

In the worst case: T(n)=n O(n)

```
public int maxSpan(int[] nums) {
    int maxspan=1;                                  //C1
        int span=0;                                 //C2

       if (nums.length>0) {                         //C3
        for (int  i= 0; i < nums.length; i++) {     //C4*n
                for (int j = nums.length-1; j >= i; j--) {    //C5*n*n

                    if(nums[j] == nums[i]){                   //C6*n*n
                        span = j - i+1;                       //C7*n*n
                        if(span > maxspan){                   //C8*n*n
                            maxspan = span;                   //C9*n*n
                        }

                    }
                }
            }
     }

         else {                                     //C10
            maxspan=0;                              //C11

        }
        return maxspan;                             //C12
}
```

Complexity:

In the best case: T(n)=C6

In the worst case: T(n)=n*n O(n^2)

PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co

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

Complexity:

In the best case: T(n)=C7

In the worst case: T(n)=n*n O(n^2)

```
public int[] fix45(int[] nums) {
   for (int i = 0; i < nums.length; i++)                  //C1*n
      if (nums[i] == 5 && i == 0                          //C2*n
        || nums[i] == 5 && nums[i - 1] != 4) {
         int p5 = i;                                     //C3*n
         for (int j = 0; j < nums.length; j++)           //C4*n*n
            if (nums[j] == 4 && nums[j + 1] != 5) {      //C5*n*n
               int temp = nums[j + 1];                   //C6*n*n
               nums[j + 1] = 5;                          //C7*n*n
               nums[p5] = temp;                          //C8*n*n
               break;                                    //C9*n*n
            }
   }
   return nums;                                          //C10
}
```

Complexity:

In the best case: T(n)=C10

In the worst case: T(n)=n*n O(n^2)

```
public boolean canBalance(int[] nums) {
   for (int i = 0; i < nums.length; i++) {                  //C1*n
      int sum = 0;                                          //C2*n
      for (int j = 0; j < i; j++) sum += nums[j];           //C3*n*n
      for (int j = i; j < nums.length; j++) sum -= nums[j]; //C4*n*n
```

PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co

```
        if (sum == 0) return true;                    //C5*n*n
    }
    return false;                                     //C6
}
Complexity:
     In the best case: T(n)=C6
     In the worst case: T(n)=n*n O(n^2)


public int countClumps(int[] nums) {
 boolean m = false;                                   //C1
  int c = 0;                                          //C2
  for (int i = 0; i < nums.length-1; i++) {           //C3*n
    if (nums[i] == nums[i+1] && !m) {                 //C4*n
      m = true;                                       //C5*n
      c++;                                            //C6*n
    }
    else if (nums[i] != nums[i+1]) {                  //C7*n
      m = false;                                      //C8*n
    }
  }
  return c;                                           //C9
}
Complexity:
     In the best case: T(n)=C9
     In the worst case: T(n)=n*n O(n^2)
}
```

4) Practice for midterms

1. c
2. d
3. b
4. b
5. d
6. a
7. T(n)=T(n-1)+C / O(n)

PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co