Laboratory practice No. 2: Big O Notation

Agustin Restrepo Cadavid
Universidad Eafit
Medellín, Colombia
arestrepoc@eafit.edu.co

Sebastian Gonzalez
Universidad Eafit
Medellín, Colombia
Sgonzalez1@eafit.edu.co

3) Project Questions
3.1 Problems 1

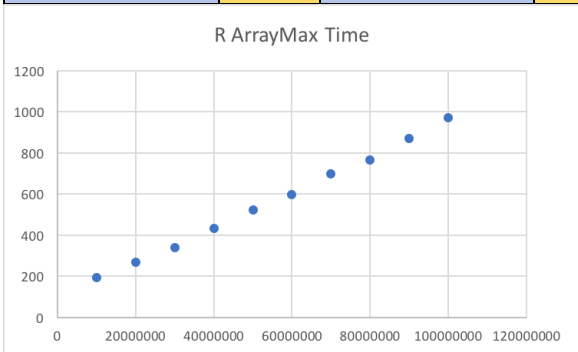
	N=100	N=1000	N=10000	N=100000	N=10000000
R Array Sum	0	0	1	3	198
R Array Maximum	0	0	1	2	86
Fibonacci	>1min	>1min	>1min	>1min	>1min

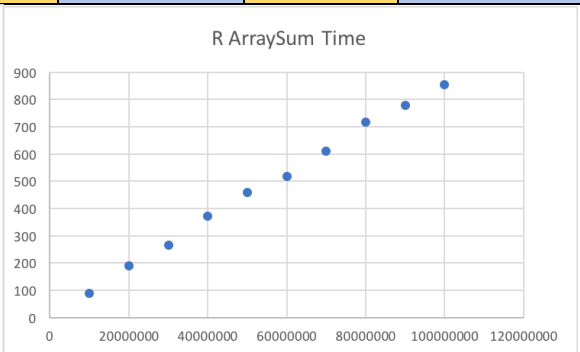
	N=100	N=1000	N=10000	N=100000	N=10000000
Array Sum	0	0	0	2	14
Array Maximum	0	0	0	2	6
Insertion Sort	0	3	29	2602	>1min
Merge Sort	0	1	2	16	1537

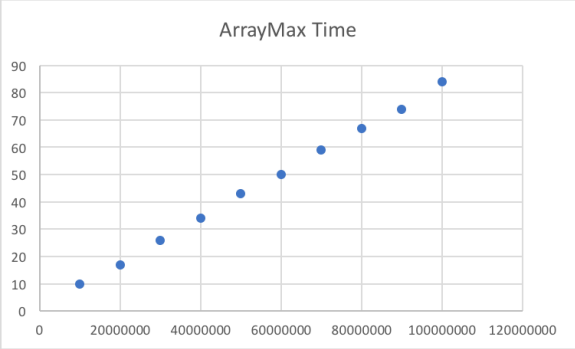
3.2

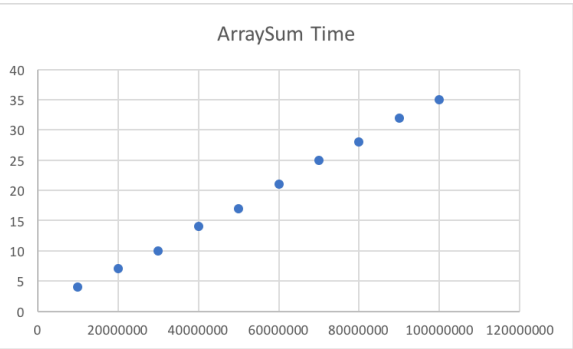
R ArrayMax		R ArraySum		Fibonacci	
N	Time	N	Time	N	Time
10000000	194	10000000	89	32	11
20000000	268	20000000	190	34	26
30000000	338	30000000	265	36	69
40000000	431	40000000	372	38	180
50000000	523	50000000	460	40	417
60000000	598	60000000	517	42	1229
70000000	696	70000000	611	44	3229
80000000	766	80000000	716	46	8446
90000000	871	90000000	778	48	22461
100000000	972	100000000	853	50	59184

ArrayMax		ArraySum		Insertion Sort		Merge Sort	
N	Time	N	Time	N	Time	N	Time
10000000	10	10000000	4	10000	29	10000000	1500
20000000	17	20000000	7	20000	102	20000000	2992
30000000	26	30000000	10	30000	239	30000000	4506
40000000	34	40000000	14	40000	415	40000000	6034
50000000	43	50000000	17	50000	650	50000000	7641
60000000	50	60000000	21	60000	930	60000000	9356
70000000	59	70000000	25	70000	1245	70000000	10889
80000000	67	80000000	28	80000	1629	80000000	12521
90000000	74	90000000	32	90000	2049	90000000	14416
100000000	84	100000000	35	100000	2543	100000000	15801

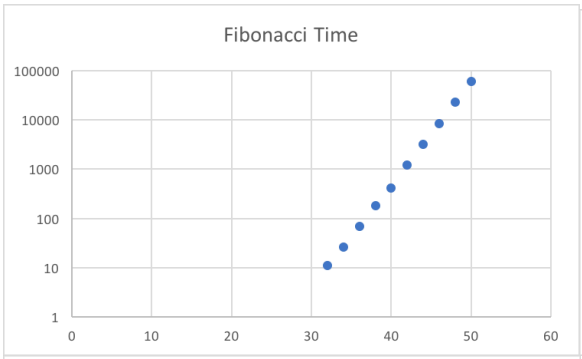


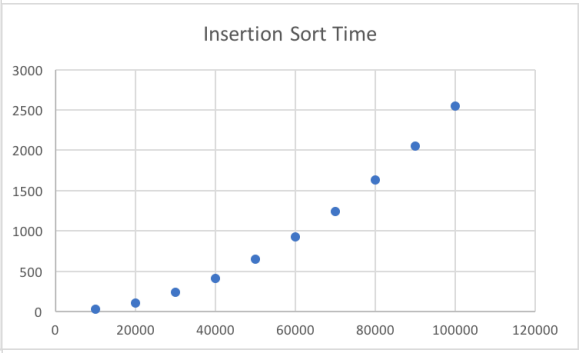


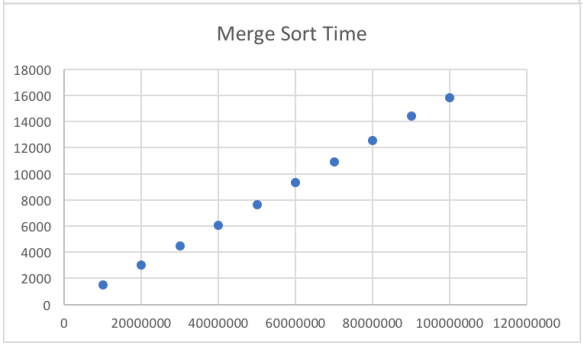




PROFESSOR MAURICIO TORO BERMÚDEZ
Phone: (+57) (4) 261 95 00 Ext. 9473. Office: 19 - 627
E-mail: mtorobe@eafit.edu.co







*Times run on –Xmx8g –Xss4g

3.3 The graphs obtained by graphing the recursive algorithm for ArraySum and ArrayMax proves that there is a linear relationship between n and the time it takes to run the algorithm. This goes accordingly to the theoretical result of 0(n) for both of the algorithms.
When we graphed the fibonacci algorithm, a logarithmic scale was needed to make the results linear, this goes accordingly to the theoretical hypothesis of O(2^n).

3.6 The graphs of Merge Sort, ArraySum and ArrayMax were lineal, and go accordingly to their theoretical Big O Notation of O(n), this is because they were composed of an algorithm with a single loop, unlike the algorithm for InsertionSort. This algorithm had a loop inside of a loop, which gives it a theoretical time rating of O(n^2). The graph for InsertionSort confirms this by displaying a hyperbole.

3.7 One can see that Insertion Sort takes too long when given a large value of N, this is because the time it takes to run the algorithm increases geometricly, or in other words, at a rate of N^2. This is because there is a loop inside of a loop in InserionSort.

3.8 As N gets bigger, one can see that the Algorithm ArraySum is not as afected as InsertionSort in terms of time. This is because ArraySum is composed of a single loop. So if n grows by 1, The loop in ArraySum only has to run one more time.

3.9 Merge Sort is more efficient than InsertionSort for bigger N because it has a Complexity of O(n), which means that as N grows, the time taken to execute Merge Sort grows at a slower rate than InertionSort. Since InsertionSort has a complexity of O(n^2), then theoretically at small numbers it should be faster than Merge Sort, However, since computers are so fast, at theese numbers both Merge Sort and InsertionSort would take 0 milliseconds to fun the algorithm.

3.10 MaxSpan is an algorithm that recives an Array of Ints and determines the maximum number of elements between any two equivalent Ints (including the two ints). The algorithm is composed of two loops, one where an idex is fixed, and then another one that determines if the value of the array at the fixed index is equal to the value at the index in the second loop. If the values are equal, the algorithm then sets the maximum to the highest value between the span between the two indexes and the previos value set to an int (this int is set to 0 when the algorithm begins).

3.11

Problems 2

Array 2

1 MachUp

```
public int matchUp(int[] nums1, int[] nums2) {
  int count = 0;                                                          C1
  for(int i = 0; i<nums1.length;i++){                                     C2*n
    if(nums1[i]-nums2[i]<3&&nums1[i]-nums2[i]>0                           C3*n
    ||nums2[i]-nums1[i]<3&&nums2[i]-nums1[i]>0) count++;                  C3*n
  }
  return count;                                                           C4
}
```

Complexity = C1+C2*n+C3*n+C4
Big O Notation = O(n)
N represents the size of the array Nums1 and Nums2(same length).

2 Sum28

```
public boolean sum28(int[] nums) {
  int sum = 0;                                                            C1
  for(int i = 0; i<nums.length;i++){                                      C2*n
    if(nums[i]==2) sum++;                                                 C3*n
  }
  return sum==4;                                                          C4
}
```

Complexity = C1+C2*n+C3*n+C4
Big O Notation = O(n)
N represents the size of the array Nums.

3 HaveTree

```
public boolean haveThree(int[] nums) {
  int counter = 0;                                                        C1
  for(int i = 0; i<nums.length;i++){                                      C2*n
    if(nums[i]==3){                                                       C3*n
      i++;                                                                C4*n
      counter++;                                                          C5*n
    }
  }
  return counter==3;                                                      C6
}
```

Complexity = C1+C2*n+C3*n+C4*n+C5*n+C6
Big O Notation = O(n)
N represents the size of the array Nums.

4 FizzArray3

```
public int[] fizzArray3(int start, int end) {
  int[] a = new int[end-start];                          C1
  for(int i = start; i<end;i++){                         C2*m
    a[i-start]=i;                                        C3*n
  }
  return a;                                              C4*n
}
```

Complexity = C1+C2*n+C3*n+C4
Big O Notation = O(n)
N represents the span between end and start.

5 Post24

```
public int[] post4(int[] nums) {
  int[] array = new int[nums.length];                    C1
  int index = 0;                                         C2
  for(int i = 0; i<nums.length;i++){                     C3*n
    if(nums[i]==4){                                      C4*n
      index = i;                                         C5*n
      array = new int[nums.length-i-1];                  C6*n
    }
    if(i<nums.length-1) array[i-index]=nums[i+1];        C7*n
  }
  return array;                                          C8
}
```

Complexity = C1+C2+C3*n+C4*n+C5*n+C6*n+C7*n+C8
Big O Notation = O(n)
N represents the size of the array Nums.

Array 3

1 MaxSpan

```
public int maxSpan(int[] nums) {
  int max = 0;                                                    C1
  for(int i = 0; i<nums.length;i++){                              C2*n
    for(int j = i;j<nums.length;j++){                             C3*n*n
      if(nums[j]==nums[i]) max = Math.max(max,j-i+1);             C4*n*n
    }
  }
  return max;                                                     C5
}
```

Complexity = C1+C2*n+C3*n^2+C4*n^2+C5
Big O Notation = O(n^2)
N represents the size of the array Nums.

2 Fix34

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

Complexity = C1+C2*n+C3*n+C4*n^2+C5*n^2+C6*n^2+C7
Big O Notation = O(n^2)
N represents the size of the array Nums.

3 LinearIn

```
public boolean linearIn(int[] outer, int[] inner) {
  for(int i=0; i<inner.length;i++){                               C1
    boolean appears = false;                                      C2*n
    for(int j=0;j<outer.length;j++){                              C3*n*m
      appears = appears||inner[i]==outer[j];                      C4*n*m
    }
    if(!appears) return false;                                    C5*n
  }
  return true;                                                    C6
}
```

Complexity = C1+C2*n+C3*n*m+C4*n*m^2+C5*n+C6
Big O Notation = O(n*m)
N represents the size of the array outer.
M represents the size of the array inner.

4 SquareUp

```
public int[] squareUp(int n) {
   int[] array = new int[n*n];                          C1
   for(int i = 0;i<n;i++){                              C2*n
     for(int j = 0; j<n;j++){                           C3*n*n
       if(n-1-i<=j) array[i*n+j]=n-j;                   C4*n*n
     }
   }
  return array;                                         C5
}
```

Complexity = C1+C2*n+C3*n^2+C4*n^2+C5
Big O Notation = O(n^2)
N represents value of the parameter n.

5 SerisUp

```
public int[] seriesUp(int n) {
  int[] array = new int[n*(n+1)/2];                     C1
   for(int i = 0;i<n;i++){                              C2*n
     for(int j = 0; j<=i;j++){                          C3*n*n
       array[(i*(i+1)/2+j)]=j+1;                        C4*n*n
     }
   }
  return array;                                         C5
}
```

Complexity = C1+C2*n+C3*n^2+C4*n^2+C5
Big O Notation = O(n^2)
N represents value of the parameter n.

4) Practice for midterms

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

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