Laboratory practice No. 2: Arrays, sorting algorithms and complexity

Juan David Echeverri Villada
Universidad Eafit
Medellín, Colombia
jdecheverv@eafit.edu.co

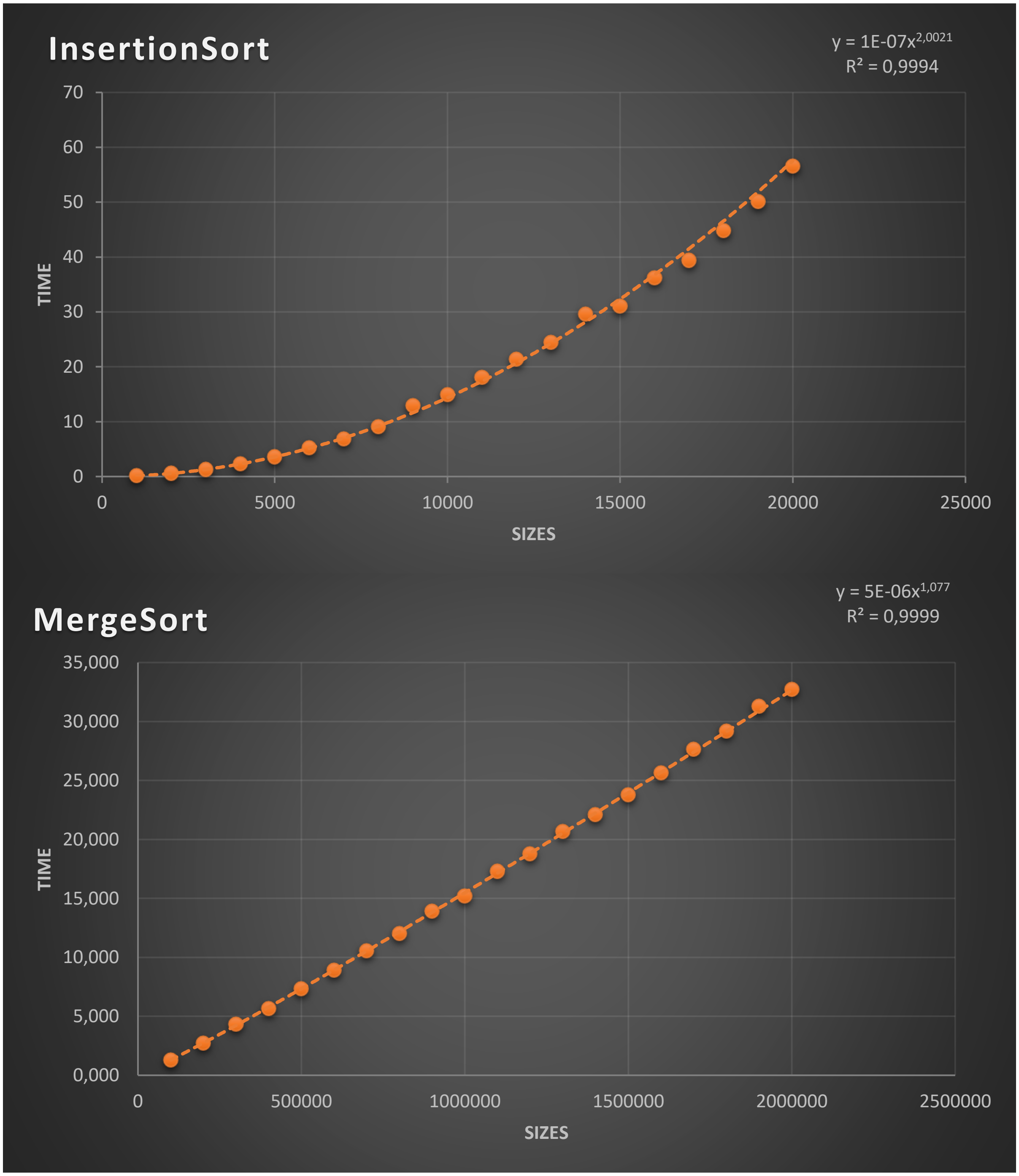
Octavio Vasquez Zapata
Universidad Eafit
Medellín, Colombia
ovasquezz@eafit.edu.co

3) Practice for final project defense presentation

3.1 Times for Merge and Insertion sort with different sizes

MergeSort		InsertionSort	
sizes	time	sizes	time
100000	1,306	1000	0,136
200000	2,722	2000	0,586
300000	4,308	3000	1,274
400000	5,679	4000	2,303
500000	7,338	5000	3,593
600000	8,885	6000	5,197
700000	10,556	7000	6,857
800000	12,001	8000	9,076
900000	13,919	9000	12,927
1000000	15,222	10000	14,893
1100000	17,289	11000	18,082
1200000	18,790	12000	21,327
1300000	20,690	13000	24,424
1400000	22,091	14000	29,584
1500000	23,786	15000	31,013
1600000	25,658	16000	36,218
1700000	27,665	17000	39,373
1800000	29,174	18000	44,814
1900000	31,304	19000	50,134
2000000	32,711	20000	56,553

3.2 Graphs of Merge and Insertion sort



3.3 Is it efficient to use insertion sort in 3d videogames?

No, due to the fact that the times obtained with insertion sort increase exponentially as its complexity, making it more inefficient to use when the size of the data sets are enormous (it could even take almost one minute to rearrange an array of size 20000, now imagine an array of size 1000000). For that, using insertion sort in a

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

video game that requires short execution times and almost instantaneous responses is not a quite good idea.

3.4 Why does a logarithm appears in the asymptotic complexity of insertion and merge sort?

Insertion sort does not have an logarithm in its asymptotic complexity, but merge sort does, this is due to the fact that merge sort divides the received array in two halves (it decreases each time the half of the data set), so in the first recursive call of the algorithm, the new arrays have a size of n/2, then, they become 4 new arrays of size n/4, and them n/8. Those divisions happen log2n, and it stops when n can't be divided anymore. For this reason, the complexity of merge sort is the quantity of divisions that were done log2n by the quantity of data in the datasets, this fact justifies the appearance of the logarithm in the complexity of merge sort.

3.5 (Optional) For big size arrays, in which way should be arranged the data so insertion sort would be faster than merge sort?

Merge sort has a complexity of nlog n for every scenario, meanwhile, insertion sort has a complexity of O(n2) for the average and the worst scenario. But, happens that insertion sort has a complexity of O(n) when the given array is already sorted from the smallest to the highest values. In this best scenario, insertion sort is faster than merge sort.

3.5 Calculation of the complexity of 2.1 and 2.2 exercises

Array 2

```
public int matchUp(int[] nums1, int[] nums2) {
  int i = 0;                                          // constant
  int count = 0;                                      // constant
  int con = 0;                                        // constant
  while(i<nums1.length){                              // n-1
    con = nums1[i] - nums2[i];                        // constant
    if( con <= 2 && con >= -2 && con != 0){           // constant
      count++;                                        // constant
    }
    i++;                                              // constant
    }
    return count;                                     // constant
```

T(n) = c_2 + n-1
O(c_2 + n-1) by O’s definition
O(n-1) by sum rule
O(n) by sum rule

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

```
public boolean has77(int[] nums) {
  int i = 0;                                              // constant
  while(i<=nums.length-2){                                // n-2
    if(nums[i] == 7){                                     // constant
      if (nums[i+1] == 7){                                // constant
        return true;                                      // constant
      }else if (i+2 < nums.length && nums[i+2] == 7){     // constant
        return true;                                      // constant
      }
      }
      i++;                                                // constant
    }
  return false;                                           // constant
}
```

T(n) = c_2 + n-2
O(c_2 + n-1) by O's definition
O(n-2) by sum rule
O(n) by sum rule

```
public boolean no14(int[] nums) {
 boolean one=true, four=true;                             // constant
    for (int i : nums) {                                  // n
        if(i==1) one = false;                             // constant
        if(i==4) four = false;                            // constant
    }
      return one || four;                                 // constant
}
```

T(n) = c_2 + n
O(c_2 + n) by O's definition
O(n) by sum rule

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

```
public boolean only14(int[] nums) {


    for (int i : nums) {                                    // n
        if(i!=1&&i!=4){                                     // constant
          return false;                                     // constant
        }

    }
      return true;                                          // constant
}
```

T(n) = c_2 + n
O(c_2 + n) by O’s definition
O(n) by sum rule

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

    return false;                                           //constant
}
```

T(n) = c_2 + n-2
O(c_2 + n-1) by O’s definition
O(n-2) by sum rule
O(n) by sum rule

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

Array 3

```
public boolean linearIn(int[] outer, int[] inner) {
int count = 0;                                          // constant
  for(int j = 0; j < inner.length; j++){                // m
    for(int i = 0; i < outer.length; i++){              // n
      if(outer[i] == inner[j]){                         // constant
        count++;                                        // constant
        break;                                          // constant
      }
      }
    }
    return count >= inner.length;                       // constant
  }
```

T(n+m) = n*m + c_2
O(n*m + c_2) by O's definition
O(n*m) by sum rule

```
public boolean canBalance(int[] nums){
   int total1 = 0;                                      // constant
  for (int i = 0; i < nums.length; i++) {               // n
    total1 += nums[i];                                  // constant
    int total2 = 0;                                     // constant
    for (int j = nums.length-1; j > i; j--) {           // n = n-1
      total2 += nums[j];
    }
    if (total1 == total2)                               // constant
      return true;                                      // constant
  }
  return false;                                         // constant
}
```

$$T(n) = \sum_{i=0}^{n} i + c\_2$$

$T(n) = n(n+1) / 2 + c\_2$
$T(n) = (n^2 + n)/ 2 + c\_2$
$O((n^2 + n)/ 2 + c\_2)$ by O's definition
$O((n^2 + n)/ 2)$ by sum rule
$O(n^2 + n)$ by simplification rule
$O(n^2)$ by sum rule

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

```
public int maxSpan(int[] nums) {
  int m = 0;                                              // constant
    for(int i = 0; i < nums.length; i++) {                // n
      int j = nums.length - 1;                            // constant
      while(nums[i] != nums[j])                           //  n
        j--;                                              // constant
        int s = j - i + 1;                                // constant
        if(s > m)                                         // constant
          m = s;                                          // constant
  }
  return m;                                               // constant
}
```

$T(n) = n * n + c_2$

$T(n) = n^2 + c_2$

$O(n^2 + c_2)$ by O's definition

$O(n^2)$ by sum rule

```
public int[] fix45(int[] nums) {
  int i = 0;                                              // constant
  int j = 0;                                              // constant
  while(j < nums.length && nums[j] != 5)                  // n
    j++;                                                  // constant
  while(i < nums.length) {                                // n
    if(nums[i] == 4) {                                    // constant
        int t = nums[i+1];                                // constant
        nums[i+1] = nums[j];                              // constant
        nums[j] = t;                                      // constant
        while((j < nums.length && nums[j] != 5) || j == i + 1)   // n = n-1
          j++;                                            // constant
      }
      i++;                                                // constant
  }
  return nums;
}
```

$$T(n) = \sum_{i=0}^{n} i + c_2$$

$T(n) = n(n+1) / 2 + c_2$

$T(n) = (n^2 + n)/ 2 + c_2$

$O((n^2 + n)/ 2 + c_2)$ by O's definition

$O((n^2 + n)/ 2)$ by sum rule

$O(n^2 + n)$ by simplification rule

$O(n^2)$ by sum rule

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

```
public int[] fix34(int[] nums) {
  int i = 0;                                        // constant
  while(i < nums.length && nums[i] != 3)            // constant (if executed, then worst case doesn't happen)
    i++;                                            // constant
  int j = i + 1;                                    // constant
  while(j < nums.length && nums[j] != 4)            // constant (if executed, then worst case doesn't happen)
    j++;                                            // constant
  while(i < nums.length) {                          // n
    if(nums[i] == 3) {                              // constant
      int temp = nums[i+1];                         // constant
      nums[i+1] = nums[j];                          // constant
      nums[j] = temp;                               // constant
      while(j < nums.length && nums[j] != 4)        // n-1
        j++;                                        // constant
      }
      i++;                                          // constant
  }
  return nums;                                      // constant
}
```

$T(n) = c\_2 + n * (n-1)$

$T(n) = c\_2 + n^2 – n$

$O(c\_2 + n^2 – n)$ by O's definition

$O(n^2 – n)$ by sum rule

$O(n^2)$ by sum rule

3.6 Variables 'n' and 'm'

When calculating an algorithm complexity, the variable 'n' means the size of the dataset that is going to be used. When the complexity of the algorithm depends on two different part of the datasets, the variable 'n' is usually used to represent the first part and the 'm' the second one.

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

4) Practice for midterms

4.1
C ---- O(n+m)
4.2
*B ---- O(n*m*√n)*
4.3
B ---- O(ancho)
4.4
B ---- $O(n^3)$
4.5
D ---- T(n)=T(n/10)+c, O(logn)
B ---- No
4.6
10000 seconds
4.7
1,2,3,4
4.8
A ---- T(n)=c+T(n-1), O(n)
4.9
C ---- $O(n^3)$
4.10
C
4.11
C ---- T(n-1)+T(n-2)
4.12
B ---- $O(m*n*\log(n)+n*m^2+n^2*\log(n)+m^3)$
4.13
C ---- 2T(n/2)+n
4.14
A ---- $O(n^3+n(\log(\log(m))+m*\sqrt{m})$

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473