Laboratory practice No. 2: Big O Notation

Isabel Piedrahita Velez
Universidad EAFIT
Medellín, Colombia
ipiedrahiv@eafit.edu.co

Vincent Alejandro Arcila Larrea
Universidad EAFIT
Medellín, Colombia
vaarcilal@eafit.edu.co

March 11, 2019

1) PROJECT SUPPORT QUESTIONS IN REPORT FOLDER

i. Insertion sort and Merge sort comparison is given on table 1.

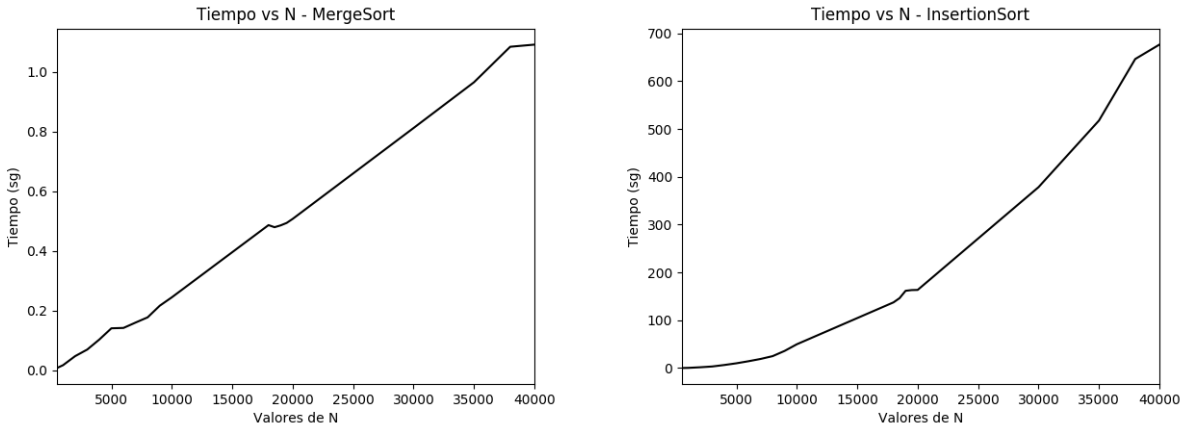
Table 1: Algorithms comparison

Size N	Merge Sort	Insertion Sort	Size N	Merge Sort	Insertion Sort
500	0.007	0.079	10000	0.244	49.756
1000	0.017	0.344	18000	0.486	137.395
2000	0.047	1.627	18500	0.479	146.206
3000	0.069	3.265	19000	0.485	161.622
4000	0.103	6.388	19500	0.493	163.115
5000	0.140	9.950	20000	0.507	163.432
6000	0.142	14.256	30000	0.811	378.424
7000	0.159	19.020	35000	0.965	517.754
8000	0.177	24.924	38000	1.083	646.485
9000	0.216	35.958	40000	1.091	676.608

ii. Charts illustrating time required for each algorithm to solve the problem for fixed values of N (being N the length of a random array of integers) are shown on figure 1.

iii. After analyzing results illustrated on figure 1 is not difficult to conclude that Merge sort is by far a faster sort algorithm than Insertion sort. For very small values of N (in essence smaller than 500) the difference is barely noticeable, but for very big values of N there is no point of comparison, given that times for Insertion sort increases exponentially, opposed to times for Merge sort which increases according to a natural logarithmic

Figure 1: Merge sort VS Insertion sort



expression as shown in 1:

$$\begin{matrix} Mergesort & & InsertionSort \\ O(n \ln n) & VS & O(n^2) \end{matrix} \tag{1}$$

iv. In no way is it a wisee decision to implement Insertion sort for such values of N. If I was the one playing that game it would be a disaster knowing that for values of N rounding 40000 my PC takes around 700 seconds to solve the problem.

v. For big values of N the only way for Insertion sort to be faster than Merge sort is that almost all elements on the array are already sorted.

2) PRACTICE FOR FINAL PROJECT DEFENSE PRESENTATION

2.a. Array II

i.

```
public int countEvens(int[] nums) {
  int acum=0;
  for(int i = 0 ; i<nums.length ; i++){
    if(nums[i]%2 == 0){
      acum += 1;
    }
  }
  return acum;
}

T(n) = n + c  or  O(n) where n is the length of the array
```

ii.

```
public int bigDiff(int[] nums) {
  int min=nums[0];
  int max=nums[0];
  for(int i=0 ; i<nums.length ; i++){
    min = Math.min(min,nums[i]);
    max = Math.max(max,nums[i]);
  }
  return max-min;
}
```

T(n) = n + c or O(n) where n is the length of the array

iii.

```
public int centeredAverage(int[] nums) {
  int min = nums[0];
  int max = nums[0];
  int acum = 0;
  for(int i=0 ; i<nums.length ; i++){
    max = Math.max(max, nums[i]);
    min = Math.min(min, nums[i]);
    acum += nums[i];
  }
  return (acum-(min+max))/(nums.length-2);
}
```

T(n) = n + c or O(n) where n is the length of the array

iv.

```
public int sum13(int[] nums) {
  int acum = 0;
  for(int i=0 ; i<nums.length ; i++){
    if(nums[i] == 13 || (i>0 && nums[i-1]==13)){
      acum += 0;
    }else{
      acum += nums[i];
    }
  }
  return acum;
}
```

T(n) = n + c or O(n) where n is the length of the array

v.

```
public boolean sum28(int[] nums) {
```

```
  int acum=0;
  for(int i=0 ; i<nums.length ; i++){
    if(nums[i]==2){
      acum += 2;
    }
  }
  return acum == 8;
}
```

```
T(n) = n + c  or  O(n) where n is the length of the array
```

2.b. Array III

i.

```
public int maxSpan(int[] nums) {
  int maxSpan = 0;
  for(int i=0 ; i<nums.length ; i++){
    for(int j=nums.length-1 ; j>=0 ; j--){
      if(nums[i]==nums[j]){
        maxSpan = Math.max(maxSpan,Math.abs(i-j+1));
      }
    }
  }
  return maxSpan;
}
```

```
T(n) = n^2 + c  or  O(n^2) where n is the length of the array
```

ii.

```
public int[] fix34(int[] nums) {
        int j = 1;
        for(int i = 0; i < nums.length - 1; i++){
                if(nums[i] == 3 && nums[i+1] != 4){
                        for( ; nums[j] != 4; j++);
                        nums[j] = nums[i+1];
                        nums[i+1] = 4;
                }
        }
        return nums;
}
```

```
T(n) = n^2 + c  or  O(n^2) where n is the length of the array
```

iii.

```
public boolean canBalance(int[] nums) {
  int acum1=0;
  int acum2=0;
  for(int i=0 ; i<nums.length ; i++){
    acum1+=nums[i];
  }

  for(int i=nums.length-1 ; i>=0 ; i--){
    acum2+=nums[i];
    acum1-=nums[i];
    if(acum1==acum2){
    return true;
    }
  }

  return false;
}

T(n) = n^2 + c  or  O(n^2) where n is the length of the array
```

iv.

```
public int[] seriesUp(int n) {
  int len = n * (n + 1)/2;
  int[] ans = new int[len];
  int pos = 0;

  for(int i = 1 ; i <= n ; i++){
    for(int j = 1 ; j <= i ; j++){
      ans[pos] = j;
      pos++;
    }
  }

  return ans;
}

 T(n) = $n^2$ + c  or  O(n^2) where n is the length of the array
```

v.

```
public int countClumps(int[] nums) {
  int clumpCount = 0;
  int i = 0;
  while(i<nums.length){
    int value = nums[i];
```

```
      i++;
      int span = 1;
      while(i < nums.length && nums[i] == value){
        span++;
        i++;
      }
      if(span>1){
        clumpCount++;
      }
    }
    return clumpCount;
}

T(n) = n^2 + c  or  O(n^2) where n is the length of the array
```

3) PRACTICE FOR MIDTERMS

4.1) c	4.2) d
4.4) b	4.5) d
4.6) a	4.9) d
4.11) c	4.12) Sí

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