# Laboratory practice No. 2: Big O Notation

**Santiago Isaza Cadavid**
Universidad EAFIT
Medellín, Colombia
sisazac@eafit.edu.co

**Hamilton Smith Gómez Osorio**
Universidad EAFIT
Medellín, Colombia
hsgomezo@eafit.edu.co

September 3, 2018

**1) *GitHub's codes***

**2) *Project Questions Simulation***

**2.a. *Algorithms's chart***

**2.b. *Algorithms's graphics***

**2.c. *Given the above information, how efficient is merge sort compared with insertion sort for large arrays? Is it appropriate to use insertion sort for a data base with millions of elements?***

**2.d. *Explain with your own words how does the Codingbat's Array3 exercise maxSpan works. Why?***

**2.e. *Calculate the complexity of the on-line exercise***

**i.**

```
public int countEvens(int[] nums) {
 int n=0;
  for(int i=0;i<nums.length;i++){ // C_1 *(n + 1)
      if(nums[i]%2==0) n+=1; // C_2 * n
  }
  return n;  //C_3
}
```

$$T(n) = C_1 * n + C_2 * (n+1) + C_3$$
$$T(n) = O(C_1 * n + C_2 * (n+1) + C_3)$$
$$T(n) = O(n + n + 1)$$
$$T(n) = O(n)$$

The complexity of this algorithm is O($n$)

**ii.**

```
public boolean lucky13(int[] nums) {
    for(int i=0;i<nums.length;i++){ // C_1 * (n + 1)
        if(nums[i]==3 || nums[i]==1) return false; //C_2 * n
    }
    return true; //C_3
}
```

$$T(n) = C_1 * n + C_2 * (n + 1) + C_3$$
$$T(n) = O(C_1 * n + C_2 * (n + 1) + C_3)$$
$$T(n) = O(n + n + 1)$$
$$T(n) = O(n)$$

The complexity of this algorithm is O($n$)

**iii.**

```
public boolean isEverywhere(int[] nums, int val) {
    for(int i=0;i<nums.length-1;i++){ // C_1 * n
        if(nums[i]!=val && nums[i+1]!=val) return false; //C_2 * (n - 1)
    }
    return true;// C_3
}
```

$$T(n) = C_1 * n + C_2 * (n - 1) + C_3$$
$$T(n) = O(C_1 * n + C_2 * (n - 1) + C_3)$$
$$T(n) = O(n + n - 1)$$
$$T(n) = O(n)$$

The complexity of this algorithm is O($n$)

**iv.**

```
public boolean modThree(int[] nums) {
    for(int i=0;i<nums.length-2;i++){ // C_1 * (n - 1)
        if(nums[i]%2==0 && nums[i+1]%2==0 &&
        nums[i+2]%2==0) return true; // C_2 * (n - 2)
        if(nums[i]%2==1 && nums[i+1]%2==1 &&
        nums[i+2]%2==1) return true; // C_3 * (n - 2)
    }
    return false; //C_4
}
```

$$T(n) = C_1 * (n-1) + C_2 * (n-2) + C_4$$
$$T(n) = O(C_1 * (n-1) + C_2 * (n-2) + C_4)$$
$$T(n) = O(n + n - 3)$$
$$T(n) = O(n)$$

The complexity of this algorithm is O($n$)

**v.**

```
    public boolean tripleUp(int[] nums) {
    for(int i=0;i<nums.length-2;i++){ // C_1 * (n - 1)
        if(nums[i+1]==nums[i]+1 &&
        nums[i+2]==nums[i]+2) return true; // C_2 * (n - 2)
    }
    return false; //C_3
}
```

$$T(n) = C_1 * (n-1) + C_2 * (n-2) + C_3$$
$$T(n) = O(C_1 * (n-1) + C_2 * (n-2) + C_3)$$
$$T(n) = O(n + n + -3)$$
$$T(n) = O(n)$$

The complexity of this algorithm is O($n$)

**vi.**

```
public boolean linearIn(int[] outer, int[] inner) {
    int n=0;
    for(int i=0;i<inner.length;i++){   // C_1 * (n + 1)*m
        for(int j=0;j<outer.length;j++){     //C_2 * (n*m)
            if(inner[i]==outer[j]){       // C_3 * (n*m)
                n++;       //C_4
                break;
            }
        }
    }
    return n==inner.length; //C_5
}
```

$$T(n) = C_1 * (n+1) * m + C_2 + (n*m) + C_3 + (n*m) + C_4 + C_5$$
$$T(n) = O(C_1 * (n+1) * m + C_2 * (n+1*m) + C_3 * (n*m) + C_4 + C_5)$$
$$T(n) = O(n*m + m + n*m + n*m)$$
$$T(n) = O(3n*m)$$

$$T(n) = O(n * m)$$

The complexity of this algorithm is O($n * m$)

**vii.**

```
    public int[] seriesUp(int n) {
        int [] arr=new int[n*(n+1)/2]; //C_1
        int num=0; //C_2
        for(int i=1;i<=n;i++){ //C_3 * (n + 1)
            for(int j=1;j<=i;j++){ //C_4 * (n*(n+1))
                arr[num]=j; //C_5
                num++; //C_6
            }
        }
        return arr; // C_7
    }
}
```

$$T(n) = C_1 + C_2 + C_3 * (n+1) + C_4 * (n * (n+1)) + C_5 + C_6 + C_7$$
$$T(n) = O(C_1 + C_2 + C_3 * (n * (n+1)) + C_4 * (n * (n+1)) + C_5 + C_6 + C_7)$$
$$T(n) = O(n^2 + n + n^2 + n)$$
$$T(n) = O(2n^2)$$
$$T(n) = O(n^2)$$

The complexity of this algorithm is O($n^2$)

### *2.f. Explain what the variable n means in the previous exercises*

## *3) Midterm Simulation*

### *3.a. Exercise 1*

c) O(n+m)

### *3.b. Exercise 2*

a) O($m * n$)

### *3.c. Exercise 3*

b) O(ancho)

### *3.d. Exercise 4*

b) O($n^3$)

### *3.e. Exercise 5*

d) O($n^2$)

### *3.f. Exercise 6*

a) T(n)= T(n-1)+T(n-2)+C

### *3.g. Exercise 7*

#### 3.7.1 Worst case-scenario number of steps

T(n)=T(n-1)+C

#### 3.7.2 Asymptotic Complexity

O(n)

### *3.h. Exercise 8*

The mystery(n) function executes $n * \sqrt{n}$ steps

### *3.i. Exercise 9*

d) Executes more than $n^2 + n * m$

### *3.j. Exercise 10*

a) Executes less than $n * \log n$ steps

### *3.k. Exercise 11*

c) Executes T(n) = T(n-1)+T(n-2)+C steps

### *3.l. Exercise 12*

b) O($m\sqrt{n}$)

### *3.m. Exercise 13*

a) O($n^3$)

## *4) Recommended reading*

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