Laboratory practice No. 2: BIG O

Cristian Alexis Giraldo Agudelo
Universidad Eafit
Medellín, Colombia
cagiraldoa@eafit.edu.co

Daniel Alejandro Cifuentes Londoño
Universidad Eafit
Medellín, Colombia
dacifuentl@eafit.edu.co

3.1

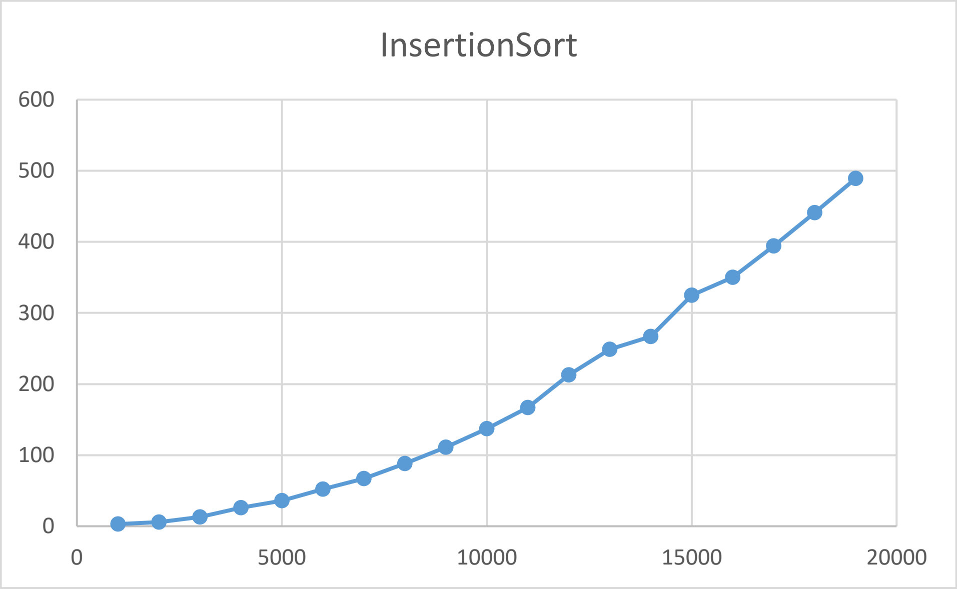
Times of InsertionSort

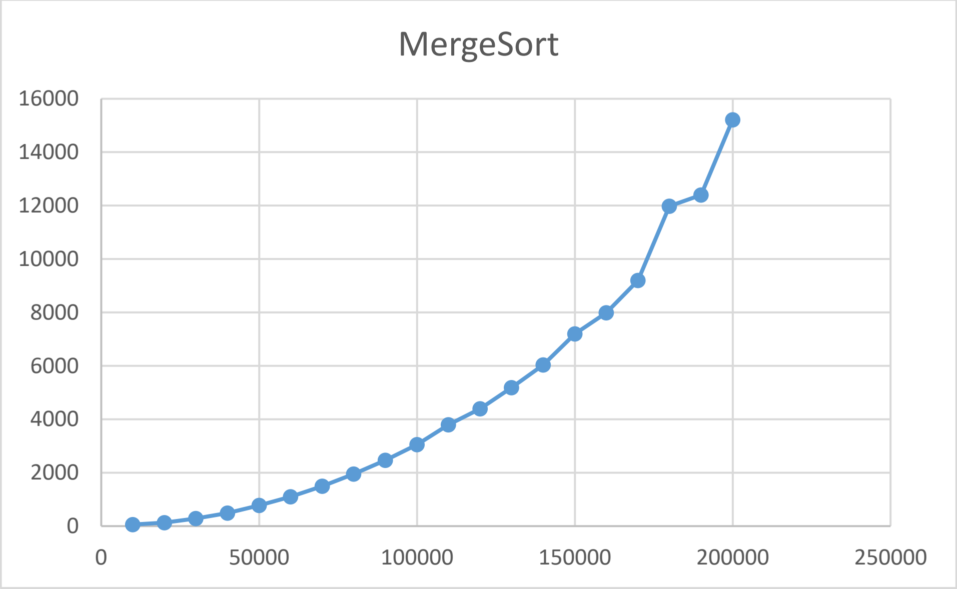
Size	Time
10000	38
20000	145
30000	314
40000	655
50000	897
60000	1356
70000	1677
80000	2517
90000	2827
100000	3732
110000	4361
120000	4828
130000	7819
140000	7081
150000	7937
160000	8502
170000	9448
180000	10695
190000	11809
200000	13782

Times of Mergesort

Size	Time
10000	59
20000	128
30000	290
40000	494
50000	780
60000	1099
70000	1491
80000	1950
90000	2468
100000	3055
110000	3791
120000	4393
130000	5184
140000	6031
150000	7200
160000	7986
170000	9200
180000	11982
190000	12393
200000	15218

3.2


InsertionSort
600
500
400
300
200
100
0
0
5000
10000
15000
20000


MergeSort
16000
14000
12000
10000
8000
6000
4000
2000
0
0
50000
100000
150000
200000
250000

3.3

In the case of very large arrays, the MergeSort is much more efficient, since the insertionsort reviews each element of the array to order them, while the MergeSort separates the elements of the array into several parts allowing greater optimization at the time of ordering.

3.4

It would not be appropriate to use the insertionSort, since it would take much longer to process all the elements and order them, which causes a notable loss of data and memory in the game, thus causing bad execution in the video game.

3.5

For the insersionSort to be faster than the MergeSort, the elements that are in the array should be almost completely sorted, this allows the steps to achieve the objective or ordering are very few

3.6 MaxSpan Explanation

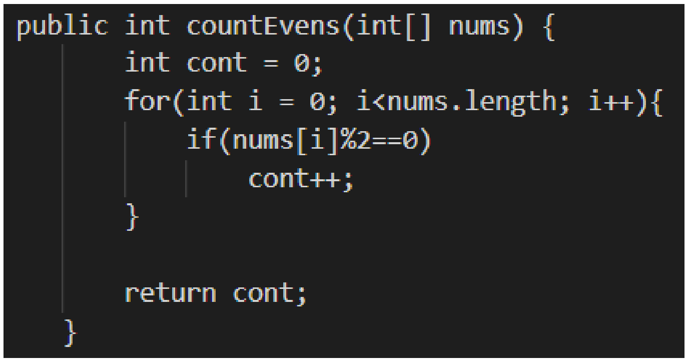
What is sought in this algorithm is basically to find the number of elements including the ends in an interval that must have the same number as left and right terminals.

To achieve the above mentioned the following is done:

- The variable m is initialized to 0 as the value to return.
- The arrangement is traversed twice, one ascending and one descending.
- Then we create a conditional, which at the time you find two equal numbers will make a subtraction between the major and the minor position and even add one to be able to include the extremes; This operation will be added to the variable s.
- If s is greater than m, the value to be returned will be that of the operation.

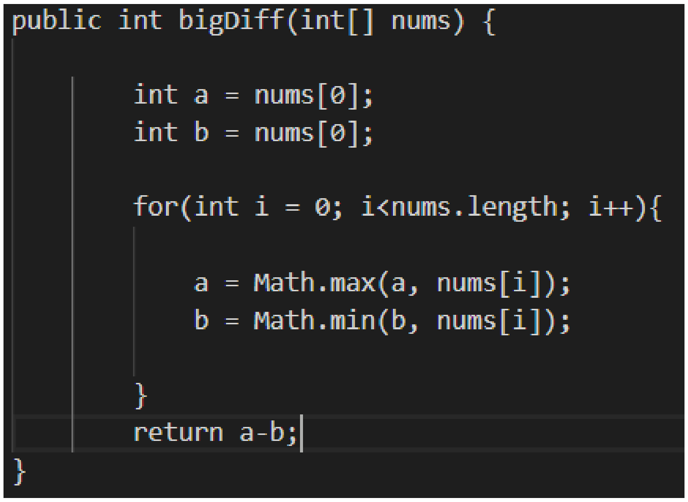
3.7

Array 2



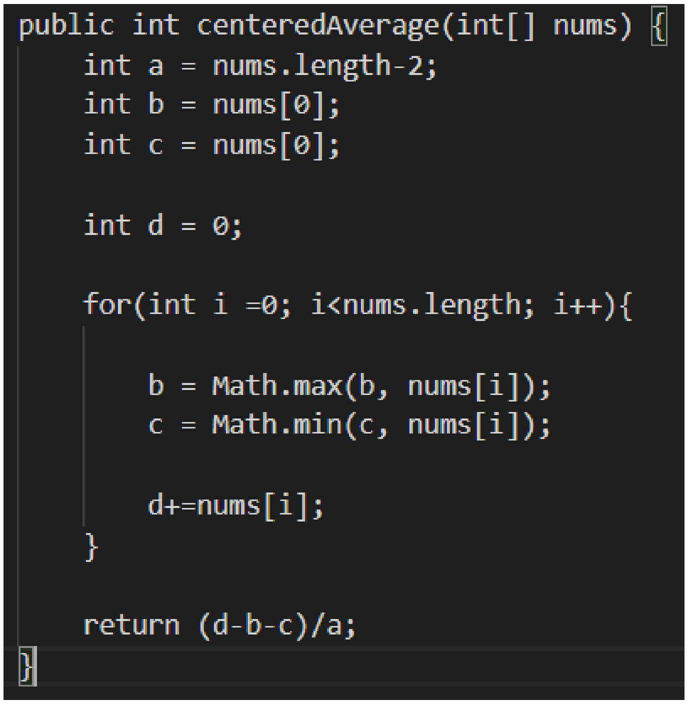
```
public int countEvens(int[] nums) {
        int cont = 0;
        for(int i = 0; i<nums.length; i++){
            if(nums[i]%2==0)
                cont++;
        }

        return cont;
    }
```

O(n)



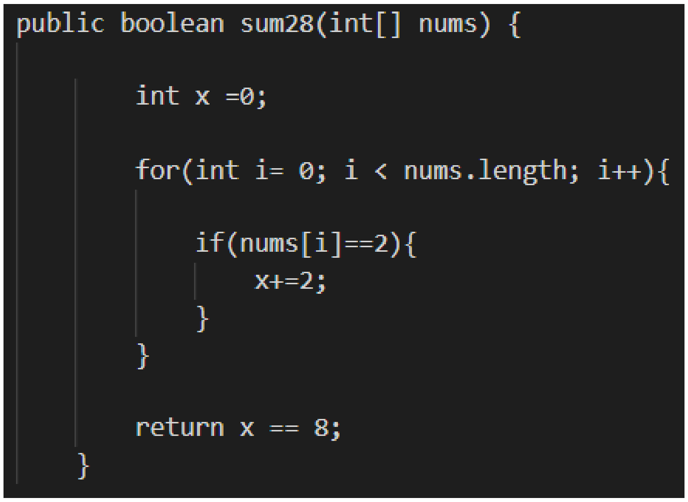
```
public int bigDiff(int[] nums) {

        int a = nums[0];
        int b = nums[0];

        for(int i = 0; i<nums.length; i++){

            a = Math.max(a, nums[i]);
            b = Math.min(b, nums[i]);

        }
        return a-b;
}
```

O(n)



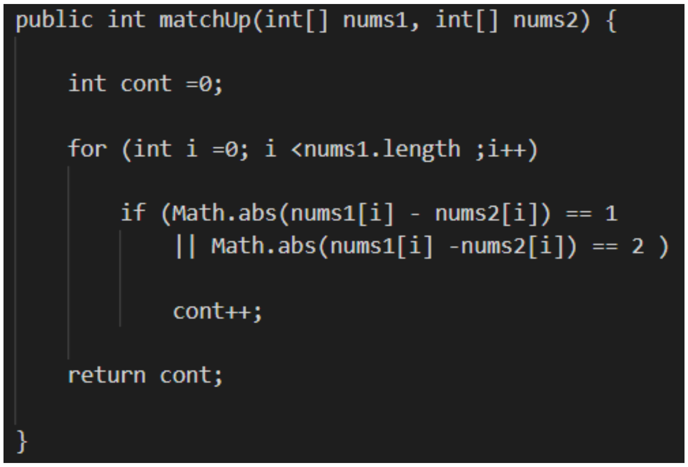
```
public int centeredAverage(int[] nums) {
    int a = nums.length-2;
    int b = nums[0];
    int c = nums[0];

    int d = 0;

    for(int i =0; i<nums.length; i++){

        b = Math.max(b, nums[i]);
        c = Math.min(c, nums[i]);

        d+=nums[i];
    }

    return (d-b-c)/a;
}
```

O(n)



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

        int x =0;

        for(int i= 0; i < nums.length; i++){

            if(nums[i]==2){
                x+=2;
            }
        }

        return x == 8;
    }
```

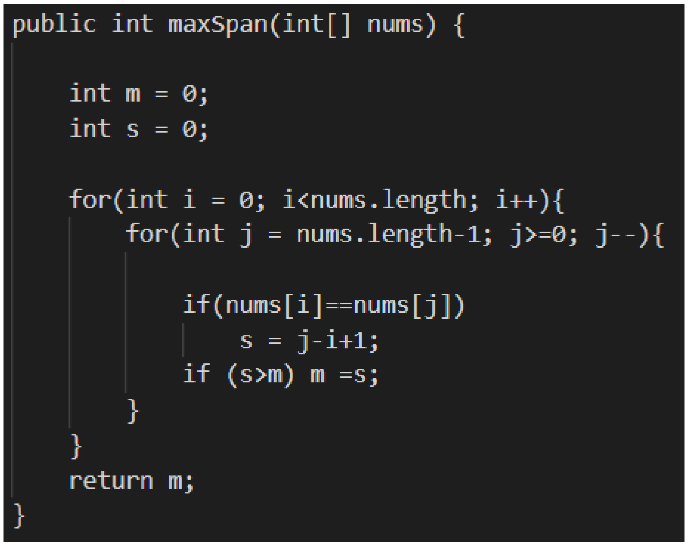
O(n)



```
public int matchUp(int[] nums1, int[] nums2) {

    int cont =0;

    for (int i =0; i <nums1.length ;i++)

        if (Math.abs(nums1[i] - nums2[i]) == 1
            || Math.abs(nums1[i] -nums2[i]) == 2 )

            cont++;

    return cont;

}
```

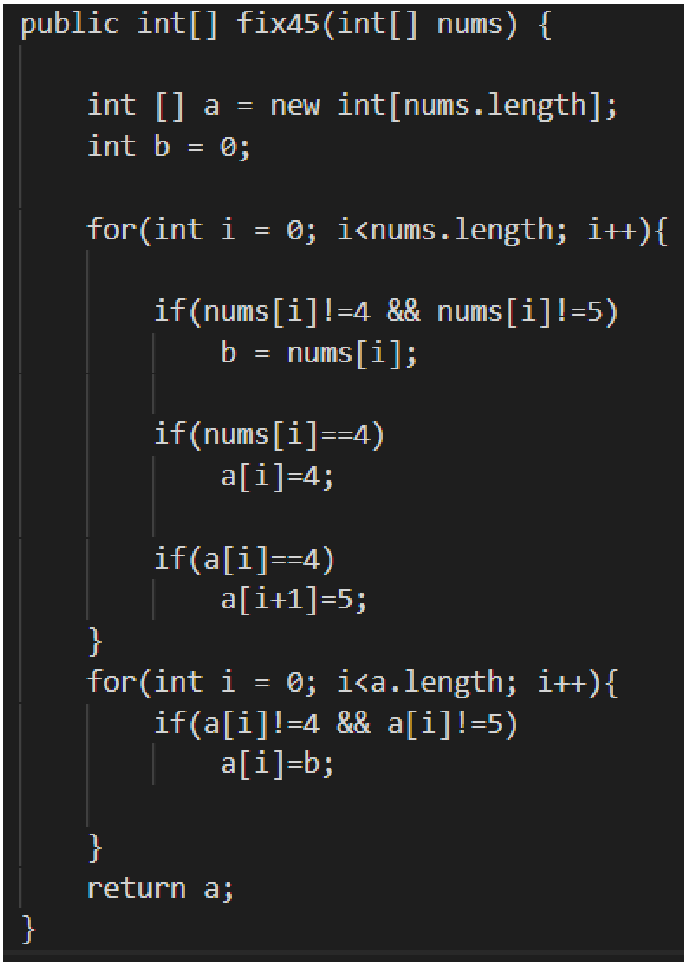
O(n)

Array 3



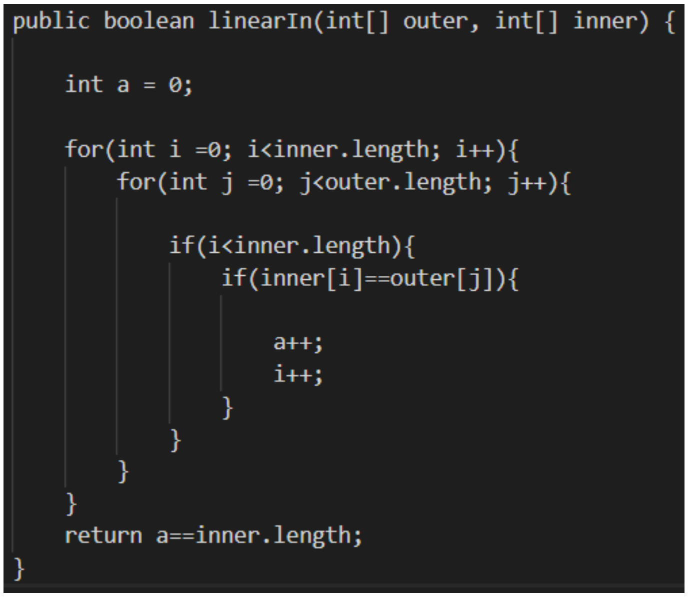
```
public int maxSpan(int[] nums) {

    int m = 0;
    int s = 0;

    for(int i = 0; i<nums.length; i++){
        for(int j = nums.length-1; j>=0; j--){

            if(nums[i]==nums[j])
                s = j-i+1;
            if (s>m) m =s;
        }
    }
    return m;
}
```

$O(n^2)$



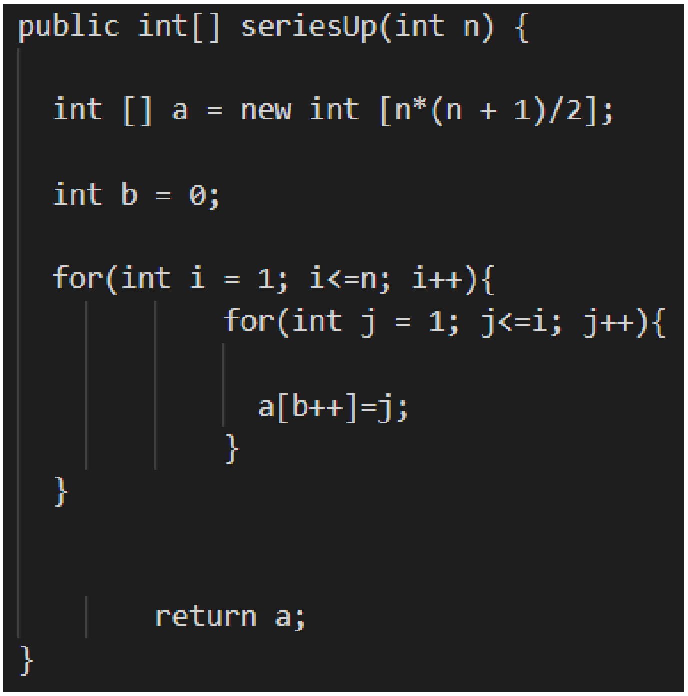
```
public int[] fix45(int[] nums) {

    int [] a = new int[nums.length];
    int b = 0;

    for(int i = 0; i<nums.length; i++){

        if(nums[i]!=4 && nums[i]!=5)
            b = nums[i];

        if(nums[i]==4)
            a[i]=4;

        if(a[i]==4)
            a[i+1]=5;
    }
    for(int i = 0; i<a.length; i++){
        if(a[i]!=4 && a[i]!=5)
            a[i]=b;

    }
    return a;
}
```

$O(n^2)$



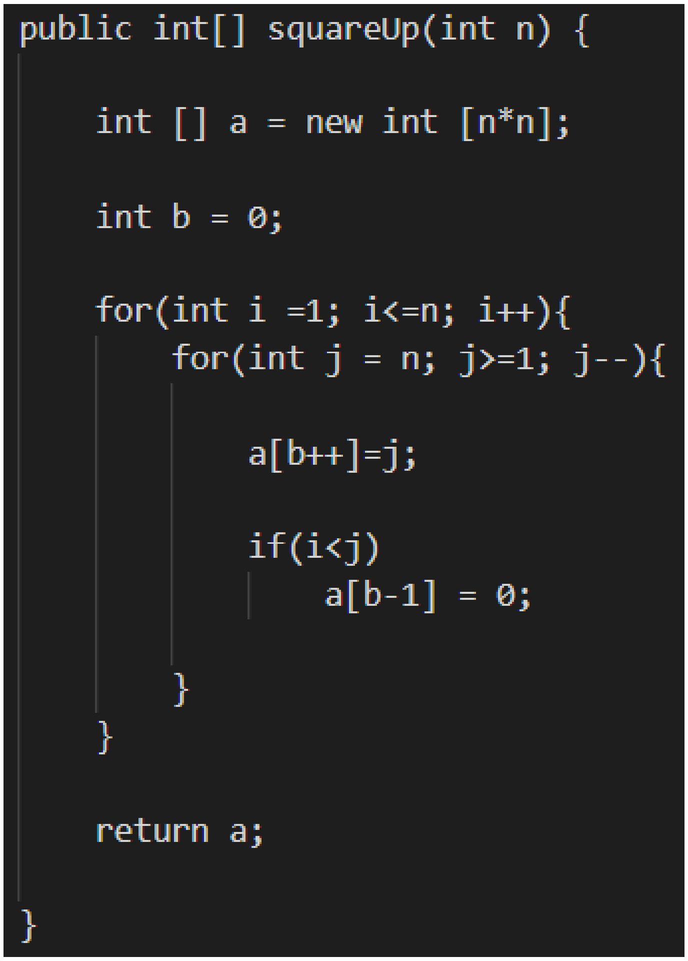
```
public boolean linearIn(int[] outer, int[] inner) {

    int a = 0;

    for(int i =0; i<inner.length; i++){
        for(int j =0; j<outer.length; j++){

            if(i<inner.length){
                if(inner[i]==outer[j]){

                    a++;
                    i++;
                }
            }
        }
    }
    return a==inner.length;
}
```

O(o*i)



```
public int[] seriesUp(int n) {

  int [] a = new int [n*(n + 1)/2];

  int b = 0;

  for(int i = 1; i<=n; i++){
             for(int j = 1; j<=i; j++){

               a[b++]=j;
             }
  }


        return a;
}
```

O(m^2)



```
public int[] squareUp(int n) {

    int [] a = new int [n*n];

    int b = 0;

    for(int i =1; i<=n; i++){
        for(int j = n; j>=1; j--){

            a[b++]=j;

            if(i<j)
                a[b-1] = 0;

        }
    }

    return a;

}
```

O(m^2)

3.8 Variables "n", "m" … in complexity

The previous variables that were used in the complexity equations or any others are those that directly influence the type of equation depending on the complexity of the algorithm presented.

***n* =** Array length
***o* =** Outer length
***i* =** Inner length
***m* =** Variable for the array creation

4) Practice for midterms

4.1 C
4.2 B
4.3 B
4.4 A
4.5 D
4.6 A
4.7 Q
4.7.1 *T(n) = T(n - 1) + c*
4.7.2 *O(n)*
4.8 A
4.9 D

4.10 C
4.11 C
4.12 B
4.13 C
4.14 A