

## Lab 2

---

### Question 1. Comparing Algorithms. Problem: Find the THIRD largest in an array.

**Algorithm 1** : Idea – Use three loops one after another. First will find Max. Second will find Second Max,

Third will find third max. Note that it is possible First max == second Max == Third Max as in

7, 20, 18, 4, 20, 19, 20, 3.

and your program should return 20 in this case.

**Algorithm 2** : Idea – Use one loop. Maintain three variable max, preMax and prePreMax such that max

will have the maximum value, preMax will have the second largest and prePreMax will have the third largest value.

In this lab, for both algorithms you will

(a) write the pseudo code. (Must follow the notations and conventions used in today's Lecture)

(b) determine the worst-case time complexity by counting as in Slide 15 Lesson 2.

(c) Perform an empirical time comparison by implementing using Java similar to what you did in W1D1.

(d) Draw a chart to compare algorithms.

a)

#### **Algorithm algorithm1(array)**

Input: array of integers

Output: third largest value

```
if length(array) is 0:  
    return Integer.MIN_VALUE;
```

```
maxIndex ← -1  
secondMaxIndex ← -1
```

```
max ← Integer.MIN_VALUE  
for i ← 0 to length(array) - 1 do:  
    if array[i] > max then:  
        max ← array[i]  
        maxIndex ← i
```

```
max ← Integer.MIN_VALUE  
for i ← 0 to length(array) - 1 do:  
    if i ≠ maxIndex and array[i] > max then:
```

```

    max ← array[i]
    secondMaxIndex ← i

max ← Integer.MIN_VALUE
for i ← 0 to length(array) - 1 do:
    if i ≠ maxIndex and i ≠ secondMaxIndex and array[i] > max then:
        max ← array[i]

return max

```

### Algorithm algorithm2(array)

Input: array of integers

Output: third largest value

```

if length(array) is 0:
    return Integer.MIN_VALUE

max ← Integer.MIN_VALUE
preMax ← Integer.MIN_VALUE
prePreMax ← Integer.MIN_VALUE

for i from 0 to length(array) - 1 do:
    number ← array[i]
    if number > max then:
        max ← number
    else if number > preMax then:
        preMax ← number
    else if number > prePreMax then:
        prePreMax ← number

return prePreMax

```

b)

### Analyzing Algorithm1:

There are 3 loops:

```

// n assignments of i
// n times of incrementing i
// n comparisons made on i along with traversing
for i ← 0 to length(array) - 1 do:
    // n times array[i] has been accessed
    // n comparisons made
    if array[i] > max then:
        // n times array[i] has been accessed
        // n assignments of max
        max ← array[i]
        // n assignments of maxIndex
        maxIndex ← i

```

Total for the loop: 8n

Other 2 loops are the same except they have 1 and 2 additional comparisons on each element respectively:

```

max ← Integer.MIN_VALUE
for i ← 0 to length(array) - 1 do:
    if i ≠ maxIndex and array[i] > max then:
        max ← array[i]
        secondMaxIndex ← i
Total: 9n since 1 more comparison (i ≠ maxIndex) was added

```

```

max ← Integer.MIN_VALUE
for i ← 0 to length(array) - 1 do:
    if i ≠ maxIndex and i ≠ secondMaxIndex and array[i] > max then:
        max ← array[i]

```

Total: 9n since 1 more comparison (ii ≠ secondMaxIndex) was added but no index assignment in this loop

Grand Total Time complexity for the worst case of the algorithm is:  $8n+9n+9n+c=26n$  where c represents number of constant operations

## Analyzing Algorithm 2:

There is one loop:

```

// n assignments of i
// n times of incrementing i
// n comparisons made on i along with traversing
for i from 0 to length(array) - 1 do:
    // n times array[i] has been accessed
    number ← array[i]
    // n comparisons are made at this line
    if number > max then:
        // n assignments are made at this line
        max ← number
    // n-1 comparisons are made at this line
    else if number > preMax then:
        // n-1 assignments are made at this line
        preMax ← number
    // n-2 comparisons are made at this line
    else if number > prePreMax then:
        // n-2 assignments are made at this line
        prePreMax ← number

```

Grand Total Time complexity for the worst case of the algorithm is:  $10n + c$  where c represents number of constant operations

c) You can find Algorithm.java file along with Main.java file. You can validate Algorithm.java with basic input included without any additional library. However, to run Main.java, you will need to install JFreeChart through maven since the library was used to draw the graph along with Swing. Although, the screenshot of the graph is available at d) below.

d) Comparison graph

