Algorithmic Thinking

Activity: Brushing the PF Contents

Peak Finder: Consider an array of numbers (either positive or negative), we have to find **a peak** (there could be more than 1 peak), such that the **left** and **right** element of the **current element** are less than or equal to the current element.

0	1	2	3	4	5	6	7
a	b	c	d	e	f	g	h

$\uparrow$ b is a **peak** if b >= a && b >= c

Activity: Brushing the PF Contents

Peak Finder: Consider an array of numbers (either positive or negative), we have to find **a peak** (there could be more than 1 peak), such that the **left** and **right** element of the **current element** are less than or equal to the current element.

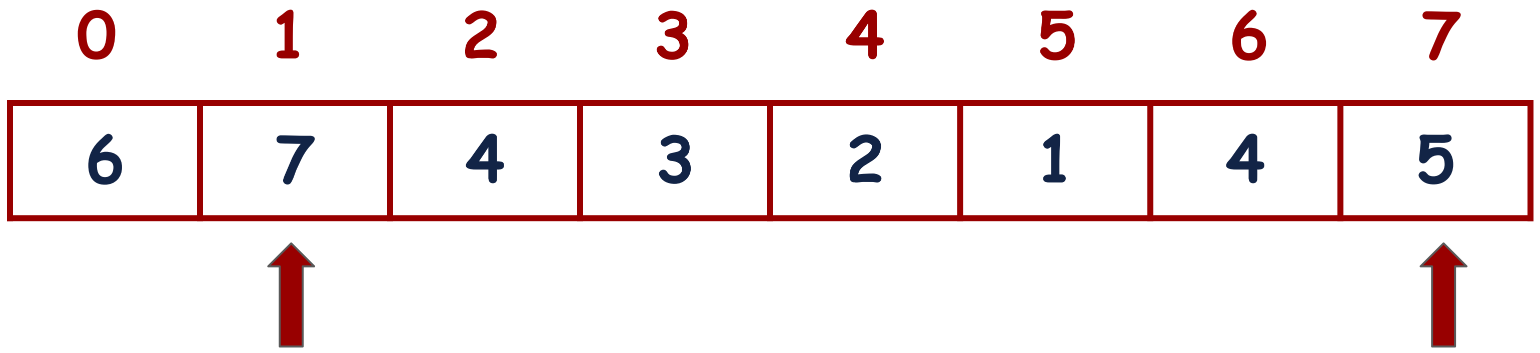
0	1	2	3	4	5	6	7
a	b	c	d	e	f	g	h

↑

a is a **peak** if a >= b

Activity: Peak Finder

For Example: There are 2 peaks in this example. Your goal is to find either 7 or 5.



7 > 6 && 7 > 4

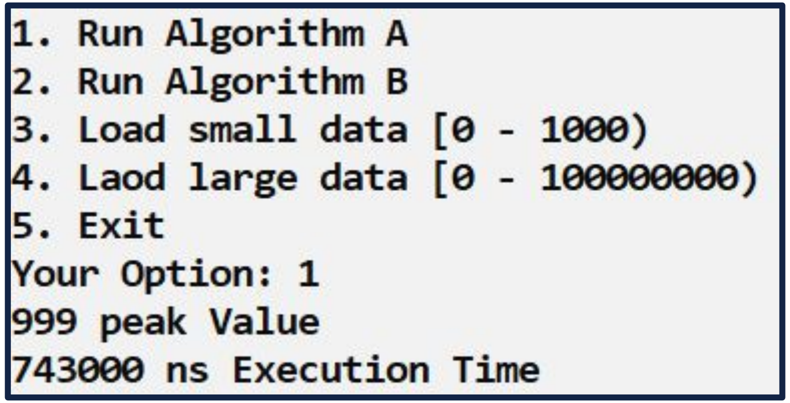
5 > 4

Activity: Brushing the PF Contents

Write the code in any programming language (C++, C# or python) you are comfortable with.

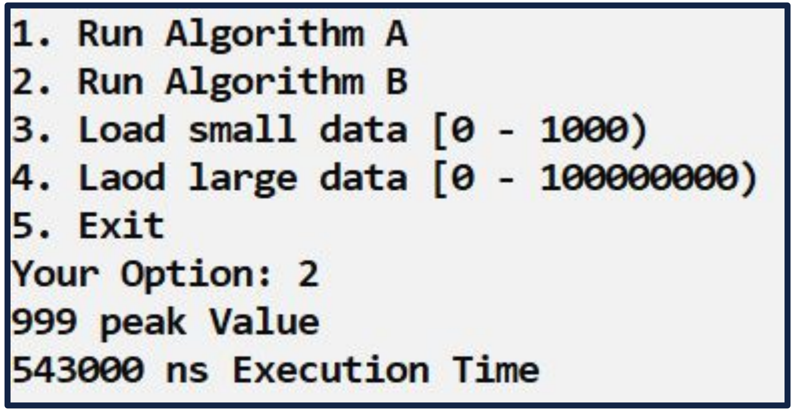
Output: Peak Finder

Output of 2 different Algorithms on 1000 elements



```
1. Run Algorithm A
2. Run Algorithm B
3. Load small data [0 - 1000)
4. Laod large data [0 - 100000000)
5. Exit
Your Option: 1
999 peak Value
743000 ns Execution Time
```

Algorithm A

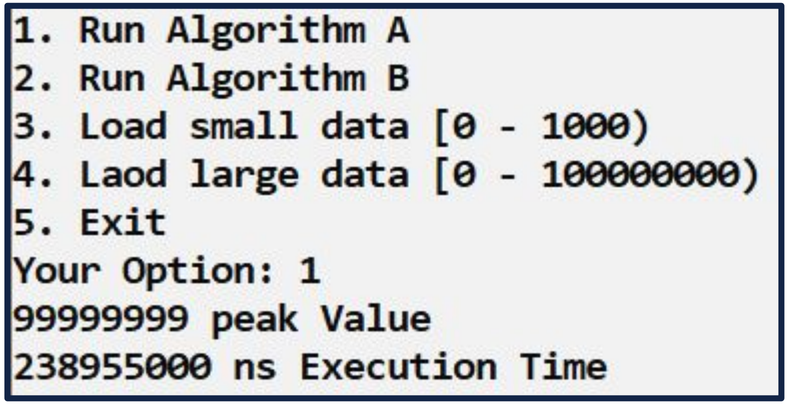


```
1. Run Algorithm A
2. Run Algorithm B
3. Load small data [0 - 1000)
4. Laod large data [0 - 100000000)
5. Exit
Your Option: 2
999 peak Value
543000 ns Execution Time
```

Algorithm B

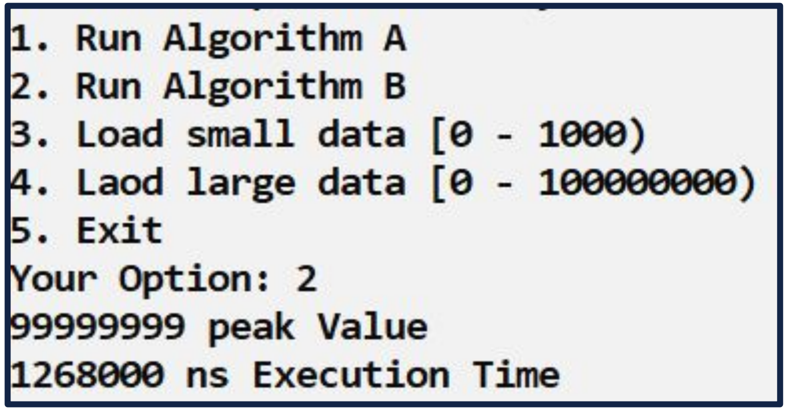
Output: Peak Finder

Output of 2 different Algorithms on One Hundred Million elements



```
1. Run Algorithm A
2. Run Algorithm B
3. Load small data [0 - 1000)
4. Laod large data [0 - 100000000)
5. Exit
Your Option: 1
99999999 peak Value
238955000 ns Execution Time
```

Algorithm A



```
1. Run Algorithm A
2. Run Algorithm B
3. Load small data [0 - 1000)
4. Laod large data [0 - 100000000)
5. Exit
Your Option: 2
99999999 peak Value
1268000 ns Execution Time
```

Algorithm B

Algorithm: Correctness

In the previous programming courses, we were only interested in the Correctness of the Algorithms.

Output sahi aa rahi hai na..!!
Bas Kafi hai..!!

Algorithm: Efficiency

But, there is also another important property of the algorithm which relates to the amount of Computational Resources used by the algorithm

Algorithm: Efficiency

Whenever we have to measure how efficient an algorithm is, we can do it in the following 2 ways.

Determine the

1. Time Complexity
2. Space Complexity

Will you Use?

Facebook has around 2.9 billion users. If facebook took too much time in loading or processing data will you have used it?

How Data is stored Efficiently?

Let’s say i want to see the list of all my friends on Facebook.
Can you think of the way how they have stored the information of Friends?

How Data is stored Efficiently?

Let's say i want to see all the posts of my friends of friends.
Can you think of the way how they have stored that information so that it can be retrieved in an efficient manner?

We have come a long way

Programming Fundamentals

Procedural Programming (Solving Problems and Writing Programs)

↓

Object Oriented Programming

Solving Problems with Object Oriented Approach

We have to go a long way

Programming Fundamentals

Procedural Programming (Solving Problems and Writing Programs)

↓

Object Oriented Programming

Solving Problems with Object Oriented Approach

↓

Data Structures and Algorithms

Solving Problems with efficient Data Structures and Algorithms with respect to Time and Space Complexity

Learning Objective

Students should be able to understand the efficiency of the Algorithms in terms of time and space complexity

Self Assessment

Given a sequence of non-negative integers $a_0,\dots,a_{n-1}$, find the **maximum pairwise product**, that is, the largest integer that can be obtained by multiplying two different elements from the sequence (or, more formally, $\max(a_i a_j)$ such that i and j are not equal. Different elements here mean a_i and a_j with $i \neq j$ (it can be the case that $a_i = a_j$).

Input Format:

The first line of the input contains an integer n. The next line contains n non-negative integers $a_0,\dots,a_{n-1}$ (separated by spaces).

Output Format:

Output a single number — the maximum pairwise product.

Constraint:

$2 \leq n \leq 2 \cdot 10^5$;

$0 \leq a_0,\dots,a_{n-1} \leq 10^5$.