

Technique: Stack with Max

Level: Easy

Implement a Stack with a max() function. This function runs in O(1) time and returns the value of the maximum number on the stack.

Questions to Clarify:

Q. Can I use extra space apart from the stack data?

A. Yes, you can use extra space.

Q. What to return if the stack is empty?

A. Throw an exception.

Solution:

There are two solutions that we recommend. Both have the same time and space complexity, but #2 uses space more efficiently.

Solution #1:

When you insert a number A, Store a Pair of numbers on the stack. The first is A. The second is the Max element on the stack. It is simple to calculate this max.

insert(A)

```
max = Maximum of A and the current max
push(Pair<A, max>)
```

max()

return the max on the top of the stack

pop()

pop the pair on top, return value

Solution #2

Every time you encounter a new max, put it on a separate Max Stack. For example:

Main: 3,1,4,2,4,6,3

 $Max: \quad \underline{3}, \underline{4}, \underline{4}, \underline{6}$

Here, the current max is 6 - the top of the Max stack. If you insert 8, that is the new max. Now it looks as follows:

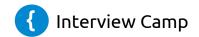
Main: 3,1,4,2,4,6,3,8

Max: 3,4,4,6,8

If we do a pop(), we remove 8 from both Main and Max.

Main: 3,1,4,2,4,6,3

Max: 3,4,4,6



If we do a pop() again, we remove 3 from the Main stack, but since it is not the max, we don't touch the Max stack.

Main: 3,1,4,2,4,6 Max: 3,4,4,6

If you do another pop(), we will remove 6 from both the stacks.

This implementation is a bit more space efficient because we don't store a new value until a new max is found. It has the same space complexity though - O(n).

For an interview, both solutions are good enough. You can ask the interviewer which one they prefer. We will implement #2 below.

Pseudocode:

```
(Note: Never write pseudocode in an actual interview. Unless you're writing a few lines quickly to plan out your solution. Your actual solution should be in a real language and use good syntax.)
```

```
Class variables:
main-stack
max-stack

insert(A)
    if A is >= max-stack.top()
        max-stack.push(A)

main-stack.push(A)

max()
    max-stack.top()

pop()
    popped-value = main-stack.pop()
    if popped-value == max
        max-stack.pop()
```

Test Cases:

Edge Cases: Empty stack Base Cases: Stack with 1 value

Regular Cases: insert smaller value, insert equal to max, insert larger value

Time Complexity: O(1) for push and pop

Space Complexity: O(n) for n insertions

```
public class StackWithMax {
   Stack<Integer> main;
   Stack<Integer> max;
   public StackWithMax() {
       main = new Stack<>();
       max = new Stack<>();
    }
   public void push(int a) {
       main.push(a);
       if (max.isEmpty() || a >= max.peek())
           max.push(a);
    }
   public int max() throws EmptyStackException {
       if (max.isEmpty())
           throw new EmptyStackException();
       return max.peek();
   }
   public int pop() throws EmptyStackException {
       if (main.isEmpty())
           throw new EmptyStackException();
       int item = main.pop();
       if (max.peek() == item)
          max.pop();
       return item;
   }
```