Stack Queue practice sheet

1. We have a stack. The stack is maintaining the ascending order always. Now, you would need to insert a new value into this stack using a **pushStack(**in value**)** function. Write the **pushStack()** function so that the function always maintains the ascending order in the stack. You can use one additional stack.
2. Create a **Stack** and populate it by integers. Write a program in a way that the largest value in that **Stack** would be in the TOP position.

**Hints**: You should use an additional helpingStack.

After your Program: Stack looks like below:

|  |
| --- |
| 99 |
| 23 |
| 53 |
| 56 |
| 19 |
| 44 |
| 22 |
| 44 |

**TOP**

**TOP**

|  |
| --- |
| 23 |
| 53 |
| 56 |
| 19 |
| 44 |
| 22 |
| 99 |
| 44 |

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**helperStack**

**Stack**

1. Create a stack S of ten elements. Take a number x from the user and delete the elements smaller than or equal to X from the stack.

Example: Suppose your initial stack is S where you pushed 10 elements as below-

Top {9, 3, 4, 15, 20, 1, 6, 4, 9, 10} Bottom

X is taken as input and X= 6. Then pop out and remove the elements smaller than or equal to X. Finally, the stack will look like-

Top {9, 15, 20, 9, 10} Bottom

**You can use only one additional stack to solve this problem**.

1. Create a stack S of ten elements. Take a number x from the user and delete the elements smaller than or equal to X from the stack.

Example: Suppose your initial stack is S where you pushed 10 elements as below-

Top {9, 3, 4, 15, 20, 1, 6, 4, 9, 10} Bottom

X is taken as input and X= 6. Then pop out and remove the elements smaller than or equal to X. Finally, the stack will look like-

Top {9, 15, 20, 9, 10} Bottom

**You can use only one additional queue to solve this problem**.