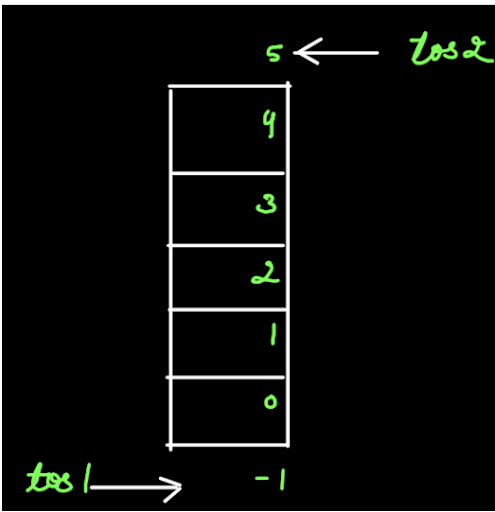
**1. Problem Discussion :**

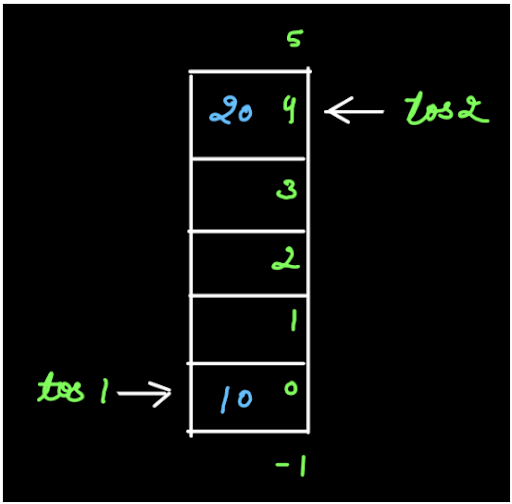
Here we want to achieve 2 stacks in a single array such that the given functions can be performed. The total size of the first stack and second stack would be hence equal to the size of the array. Reader, make sure that all the functions run in constant time. Also memory should be optimally used i.e. one of the stacks can use more elements in the array but overflow should not happen before the whole array is used.

**2. Approach :**

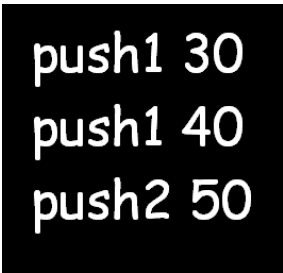
Let's start with the discussion of the solution now. In a normal stack, we had a "tos" (top of the stack) pointer which kept on increasing as we kept on pushing into the stack. Now that we have to form 2 stacks we will keep two tos. We keep tos1 at -1 index and tos2 at data.length index (see figure1).

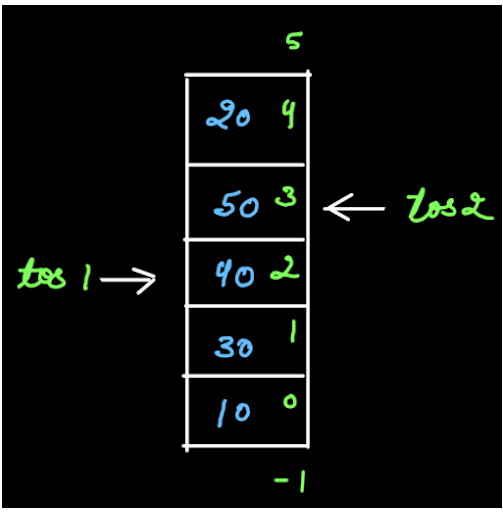
****

So, our first stack starts from the start of the stack and the second stack starts from the end of the stack. If we have to push2 20 and then push1 10 in the respective stacks then it would look like as shown in figure 2. Here tos2 will decrease by an index and tos1 will increase by an index.

****

After Figure 2, try to write how would the pushing take place if following commands are to be performed :

****

****

Now if we want to Push1 60 then we won't be able to do so because together, both the stacks have filled the entire array. From figure 4, we can also say if tos2=tos1+1 then no more elements can be pushed and stacks have been overflowed.

**3. Code Discussion:**

We try to discuss each function one by one.

****

We have already been provided the above information in our question.

CONSTRUCTOR-

****

We make an array "data" of the given length "cap" and assign tos1 and tos2 with -1 and data.length respectively.

PUSH-

push1-

****

As already discussed in "What" if tos2==tos1+1 then we print Stack overflowed. Else we increase "tos1" by an index and store the given value to that index in the array. push2-

****

Similarly, if tos2==tos1+1 then we print Stack overflowed. Else we decrease "tos2" by an index and store the given value to that index in the array.

SIZE-

size1-

****

As seen from figure 4, the first stack is from index 0 to index 2 hence the size of the stack1 is 3 i.e. tos1+1. size2-

****

Similarly, as seen from figure 4, the second stack is from index 4 to index 3 hence the size of the stack2 is 2 i.e. 5-3 or data.length-tos2.

TOP-

top1-

****

As seen from figure 4, if stack1 is empty then there is no top value. Else we return the value at tos1. top2-

****

Similarly, if stack2 is empty then there is no top value. Else we return the value at tos2.

POP-

pop1-

****

If stack1 is empty then no value can be popped. Else, as we have seen in the "What" section, the value at tos1 is returned and tos1 is decreased by an index. pop2-

****

Similarly, if stack2 is empty then no value can be popped. Else, the value at tos2 is returned and tos2 is increased by an index.

**4. Code:**

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static class TwoStack {

int[] data;

int tos1;

int tos2;

public TwoStack(int cap) {

data = new int[cap];

tos1 = -1;

tos2 = data.length;

}

int size1() {

return tos1 + 1;

}

int size2() {

return data.length - tos2;

}

void push1(int val) {

if (tos2 == tos1 + 1) {

System.out.println("Stack overflow");

} else {

tos1++;

data[tos1] = val;

}

}

void push2(int val) {

if (tos2 == tos1 + 1) {

System.out.println("Stack overflow");

} else {

tos2--;

data[tos2] = val;

}

}

int pop1() {

if (size1() == 0) {

System.out.println("Stack underflow");

return -1;

} else {

int val = data[tos1];

tos1--;

return val;

}

}

int pop2() {

if (size2() == 0) {

System.out.println("Stack underflow");

return -1;

} else {

int val = data[tos2];

tos2++;

return val;

}

}

int top1() {

if (size1() == 0) {

System.out.println("Stack underflow");

return -1;

} else {

int val = data[tos1];

return val;

}

}

int top2() {

if (size2() == 0) {

System.out.println("Stack underflow");

return -1;

} else {

int val = data[tos2];

return val;

}

}

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

TwoStack st = new TwoStack(n);

String str = br.readLine();

while (str.equals("quit") == false) {

if (str.startsWith("push1")) {

int val = Integer.parseInt(str.split(" ")[1]);

st.push1(val);

} else if (str.startsWith("pop1")) {

int val = st.pop1();

if (val != -1) {

System.out.println(val);

}

} else if (str.startsWith("top1")) {

int val = st.top1();

if (val != -1) {

System.out.println(val);

}

} else if (str.startsWith("size1")) {

System.out.println(st.size1());

} else if (str.startsWith("push2")) {

int val = Integer.parseInt(str.split(" ")[1]);

st.push2(val);

} else if (str.startsWith("pop2")) {

int val = st.pop2();

if (val != -1) {

System.out.println(val);

}

} else if (str.startsWith("top2")) {

int val = st.top2();

if (val != -1) {

System.out.println(val);

}

} else if (str.startsWith("size2")) {

System.out.println(st.size2());

}

str = br.readLine();

}

}

}

**5. Analysis**

Time Complexity:

O(1) According to the question, all the functions are coded in constant time.

Space Complexity:

O(n) Since space is used optimally and stacks require n space, therefore, the complexity is linear.

We have written all the required functions. Now, we suggest you watch the solution video if there are any gaps in your understanding.