Two Stack in an Array

Algorithm:

S1: Maintain two pointers for both the stack top element.
top1 for stack1 (start from 0 index)
top2 for stack2 (start from n-1 index)

S2: Since stack cannot overlap in one single array, we will have to see whether top1 < top2.

Code: (c++)

```
#include<iostream>
#include<stdlib>
class twoStack{
    int* arr;
    int size;
    int top1, top2;
  public:
    twoStacks(int n)
    {
      size=n;
      arr = new int[n];
      top1 = -1;
      top2 = n-1;
    }
void push1(int x){
    if( top1 < (top2-1))
    {
       top1++;
       arr[top1] = x;
    }
    else{
      cout<<"stack1 Overflow";
      exit(0);
    }
}
Void push2(int x)
{
   if(top1<(top2-1))
   {
     top2--;
     arr[top2] = x;
   }
  else{
    cout<<"stack2 overflow";
    exit(0);
  }
}

int pop1()
{
   if(top1>=0)
   {
     int x = arr[top1];
     top1--; //decrement
     return x;
   }
```

```
    else {
        cout<<"stack1 underflow";
        exit(0);
    }
}

int pop2(){
    if(top2 <=(size-1)) {
        int x = arr[top2];
        top2++; //increment
        return x;
    }
    else {
        cout<<"stack2 underflow";
        exit(0);
    }
};
int main()
{
    twoStack ts(5);
    ts.push1(5);
    ts.push2(10);
    ts.push1(15);
    ts.push2(11);
    ts.push2(7);
    cout<<"Popped element from stack1:"<<ts.pop1();
    cout<<"Popped element from stack2:"<<ts.pop2();
    return 0;
}
```

Output:

```
popped element from stack1: 15
popped element from stack2: 7
```

Complexities:

Time: O(1) → for both push and pop

Space: O(n) → where n is array size.