Stack

LIFO → last in, first out

semantics

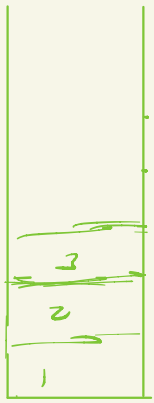
⇒ Syntax

```
Stack < Integer> st = new Stack <> ();
```

⇒ function

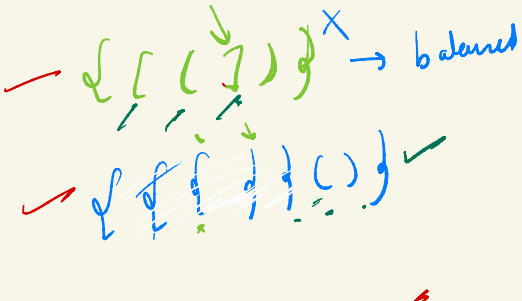
1) add → st.push(5)
2) remove → int a = st.pop() → remove and return the top element
3) int a = st.peek(); → returns the top element
4) size → st.size();

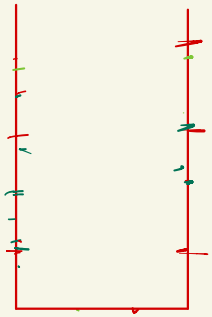
```java
public class Example {
    Run | Debug
    public static void main(String[] args) {
        Stack<Integer> st=new Stack<>();

        st.push(item: 1);
        st.push(item: 2);
        st.push(item: 3);

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

        st.push(item: 4);
        st.push(item: 5);

        while(st.size()>0){
            int a=st.pop();
            System.out.println(a);
        }
    }
```



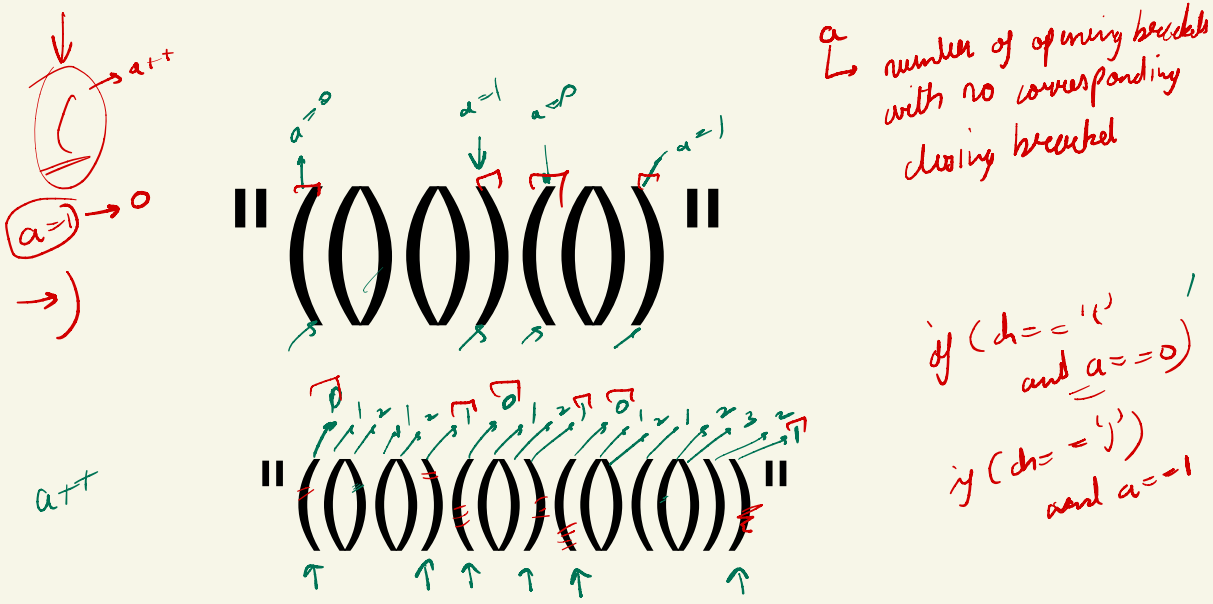
int a = st.pop()

5
4
3
2
1

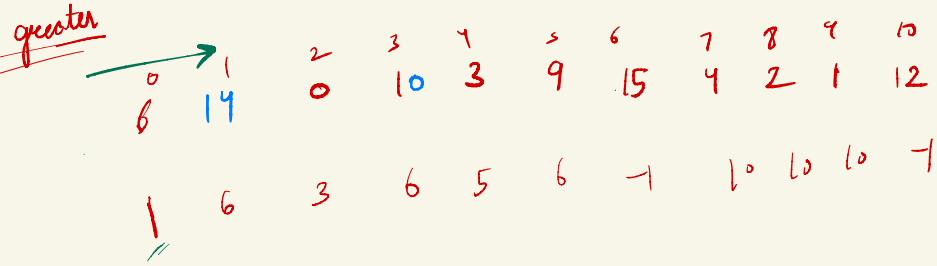




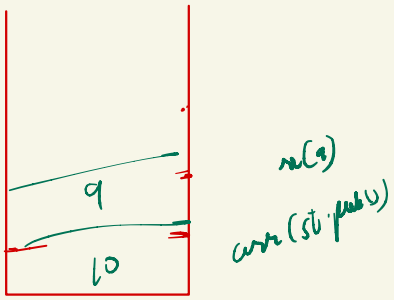
```java
public boolean isValid(String s) {
    Stack<Character> st=new Stack<>();

    for(int i=0; i<s.length(); i++){
        char ch=s.charAt(i);

        if(ch=='(' || ch=='{' || ch=='['){
            st.push(ch);
        } else if(ch==')'){
            if(st.size()==0 || st.peek()!='(') return false;

            st.pop();
        } else if(ch=='}'){
            if(st.size()==0 || st.peek()!='{') return false;

            st.pop();
        } else if(ch==']'){
            if(st.size()==0 || st.peek()!='[') return false;

            st.pop();
        }
    }

    if(st.size()==0) return true;

    return false;
}
```

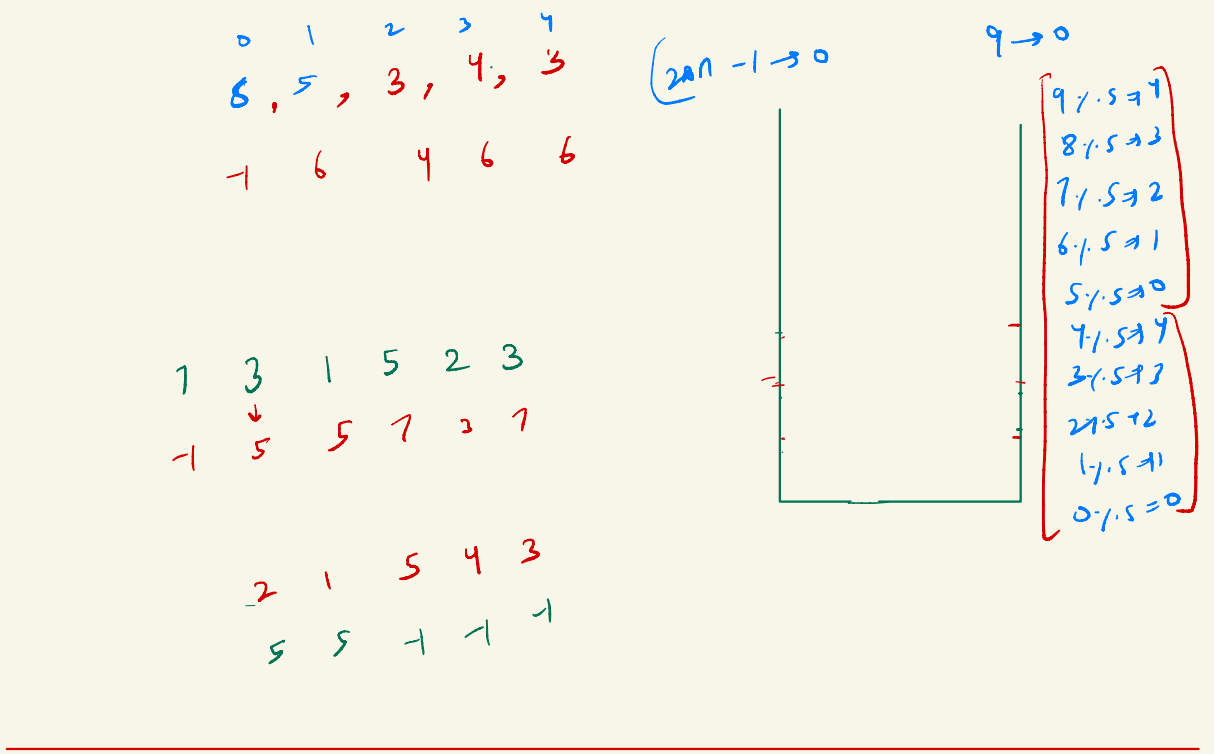


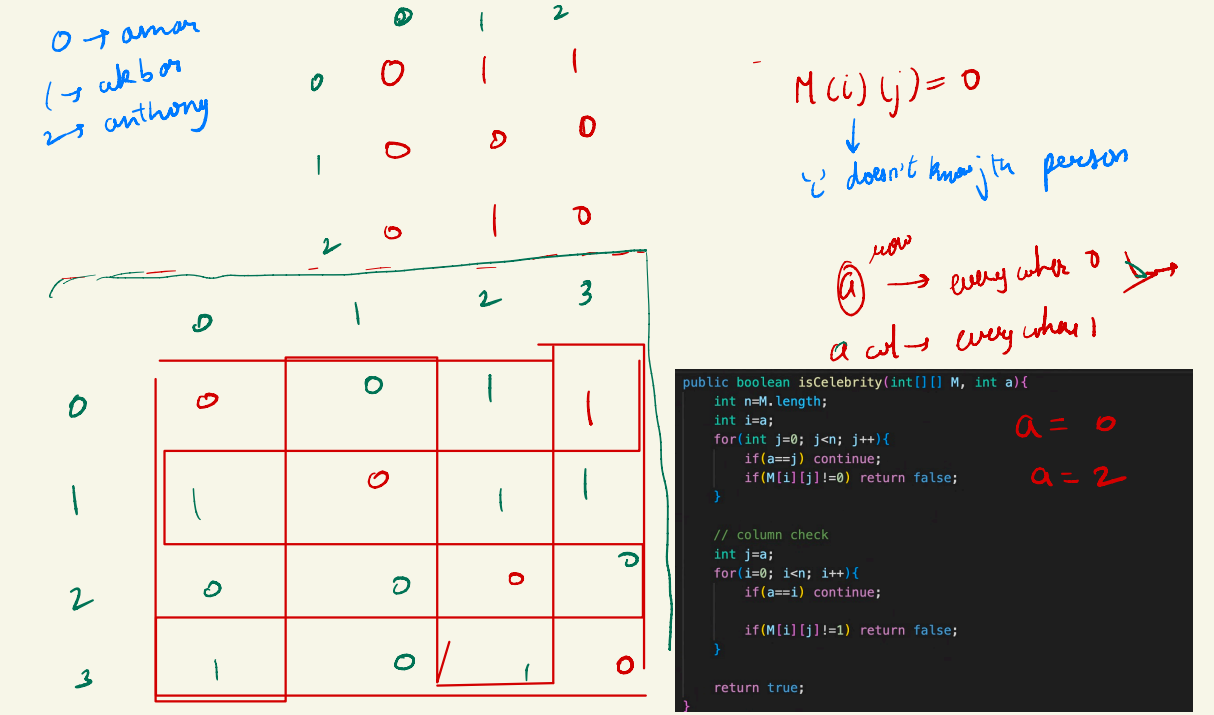
Next greater




```java
public static long[] nextLargerElement(long[] arr, int n){
    long[] ngr=new long[n];

    Stack<Long> st=new Stack<>();

    for(int i=n-1; i>=0; i--){
        long ele=arr[i];

        while(st.size()>0 && st.peek()<=ele){
            st.pop();
        }

        if(st.size()==0){
            ngr[i]=-1;
        } else {
            ngr[i]=st.peek();
        }

        st.push(ele);
    }

    return ngr;
}
```



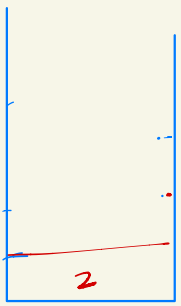





```
public boolean isCelebrity(int[][] M, int a){
    int n=M.length;
    int i=a;
    for(int j=0; j<n; j++){
        if(a==j) continue;
        if(M[i][j]!=0) return false;
    }

    // column check
    int j=a;
    for(i=0; i<n; i++){
        if(a==i) continue;

        if(M[i][j]!=1) return false;
    }

    return true;
}
```

Elimination method

- M(a)(b) = 0 → b can't be a celebrity
- M(a)(b) = 1 → a can't be celebrity



Sliding window maximum

k = 3

O(n)

nums = [1,3,-1,-3,5,3,6,7]

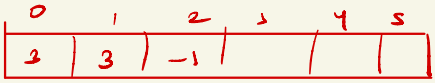
	0	1	2	3	4	5	6	7
nge[] ⇒	1	4	4	4	6	6	7	8

idx = 0 + 2

n-k

0, 1, 2 - - - n-k

n-k+1



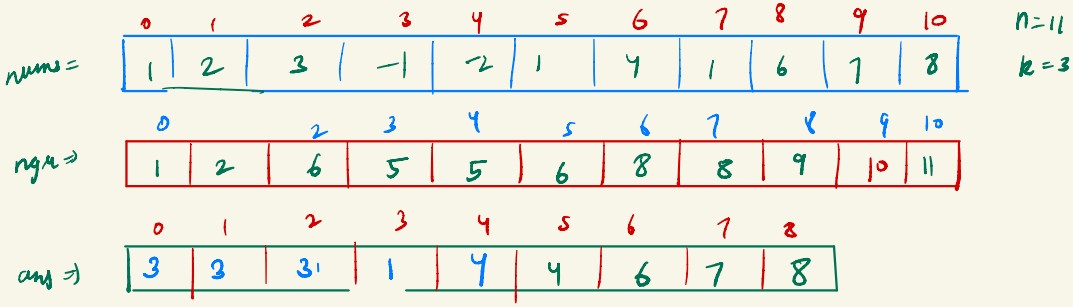
sp = 0 + 2

ep = 2 + 4

(2)

sp = 0 ;

sp <= n-k




```
int idx=0; // stores the maximum element index of current window
for(int sp=0; sp<=(n-k); sp++){
    if(idx<sp){
        idx=sp;
    }

    int ep=sp+k-1;

    while(ngr[idx]<=ep){
        idx=ngr[idx];
    }

    ans[sp]=nums[idx];
}

return ans;
```

```
int celebrity(int M[][], int n){
    Stack<Integer> st=new Stack<>();

    for(int i=0; i<n; i++){
        st.push(i);
    }

    while(st.size()>1){
        int a=st.pop();
        int b=st.pop();

        if(M[a][b]==0){
            st.push(a); // a can be celebrity
        } else {
            st.push(b);
        }
    }

    int a=st.pop();
    if(isCelebrity(M, a)) return a;

    return -1;
}
```