Easy 1

Given a string s consisting of words and spaces, return the length of the last word in the string.
A word is a maximal
substring consisting of non-space characters only.

Solution: --(using Java because it has direct trim function to eliminates spaces from front and back)

//Explaination:

//In Java there is function called trim() which eliminates space from front of the first word and from back
//of the last word in a sentence.
//we have initialize len variable whose value is initially equal to zero
//suppose for example sentence is Anjali Bhardwaj
//then loop will iterate through each character and value of len will get incremented
//if at any point of time character is equal to space then value of len will be equal to zero
//since it specifies that it is not the last word
//at the end we will return the len that specifies the length of last word.

```
public class Main
{
    int findLength(String s)
    {
        int len = 0;

        String x = s.trim();

        for (int i = 0; i < x.length(); i++) {
            if (x.charAt(i) == ' ')
                len = 0;
            else
                len++;
        }

        return len;
    }
    public static void main(String[] args)
    {
        String input = "Anjali Bhardwaj  ";
        Main solution = new Main();
        System.out.println("The length of last word is "
                           + solution.findLength(input));
    }
}
```

Medium 3

Given an m x n binary matrix filled with 0’s and 1’s, find the largest square containing only 1’s
and return its area.

Solution --(using c++)

// Explanation

//since we need the value of previously computed ans that is why we are solving it using dynamic
//programming.
//we have created a 2 D matrix name dp whose row size is eqaul to n and col size is equal to m
//since in first row and first column the maximum size square we can create is equal to 1
//so we have intialized first row and first column of our dp matrix eqaul to given matrix mat
//we have created a variable name maxi to store our maximum result
//we have iterated through the matrix from row 1 and column 1 to entire matrix
//if matrix[i][j]==0 sqaure cannot be formed so in our dp matrix we have placed the
//corresponding row and col with 0
//else we have check the row above it and the column before it and diagonal cell to get the miniminal
//element and it is equal to zero and the corresponding cell in our dp matrix would be same
//else we will add 1 to minimal element we got the size of sqaure will be increased and corresponding
//we will update our maxi
//at last we will return our maxi

```
#include<bits/stdc++.h>
using namespace std;
class FindSquare{
public:
    int maxSquare(int n, int m, vector<vector<int>> mat){
        vector<vector<int>>dp(n,vector<int>(m));
        int maxi=0;
        for(int i=0;i<n;i++){
            dp[i][0]=mat[i][0];
            maxi=max(maxi,dp[i][0]);
        }
        for(int j=0;j<m;j++){
            dp[0][j]=mat[0][j];
            maxi=max(maxi,dp[0][j]);
        }
        for(int i=1;i<n;i++){
            for(int j=1;j<m;j++){
                if(mat[i][j]==1){
                    dp[i][j]=1+min(dp[i-1][j],min(dp[i-1][j-1],dp[i][j-1]));
                }
                else{
                    dp[i][j]=0;
                }
                maxi=max(maxi,dp[i][j]);
            }
        }
        return maxi;

    }
};
int main(){
    int n,m;
    cin>>n>>m;
    vector<vector<int>>mat(n,vector<int>(m,0));
    for(int i=0;i<n*m;i++){
```

```
        cin>>mat[i/m][i%m];
    }
    FindSquare fs;
    cout<<fs.maxSquare(n,m,mat)<<endl;
    return 0;
}
```

Hard 1

Return the max sliding window.
You are given an array of integers nums, there is a sliding window of size k which is moving from the very

left of the array to the very right. You can only see the k numbers in the window.
Each time the sliding window moves right by one position.

Solution --(using c++)

```
//explanation

//we have make use of sliding window concept
//we have created a window size of k
//initially i=0 and j=0
//we will iterate the loop till j<n where n is size of an array
//we have taken a queue for storing the maximum element in each window
//initially if q is empty we will push the first element
//we will check whether i-j+1 i.e the window tranversed is less than k
//if it is less than k then we simply increment the value of j
//then we will check whether the queue is not empty and value of element
//that is in front of queue is smaller than a[j]
//if it is smaller that means element in the queue is not the larget element in the window of size k
//so we remove it from queue
//if the element that the variable j is pointing is smaller than the element that is present at the front
//of queue ,we will simply push it in the queue because maybe in future that could be largest element in wi
ndow
//of size k
// whenever the i-j+1==k that is equal to window size,we push our answer in vector and if the element is e
qual to element
//i is pointing to then we will remove it from queue because that answer will not be part of upcoming windo
w

#include <bits/stdc++.h>
using namespace std;
vector<int> maxofSubA(int a[],int k,int n){
    int i=0,j=0;
    vector<int> v;
    queue<int> q;
    while(j<n){
        while(q.size()>0 && q.front()<a[j]){
            q.pop();
        }
        q.push(a[j]);
        if(j-i+1<k){
```

```
            j++;
        }
        else if(j-i+1==k){
            v.push_back(q.front());
            if(q.front()==a[i]){
                q.pop();
            }
            i++;
            j++;
        }
    }
    return v;
}
int main()
{
    int a[]={5,2,10,1,6,20};
    vector<int> v;
    v=maxofSubA(a,3,6);
    for (auto& it : v) {

        cout << it << ' ';
    }
}
```
