SNo.	Problem Statement

1.	**Easy Level: Minimum Cost Tree From Leaf Values.**
	Code:
	
	Input: arr = [6,2,4]
	
	Output: 32
	
	Explanation: There are two possible trees shown.
	
	The first has a non-leaf node sum 36, and the second has non-leaf node sum 32.
	
	int mctFromLeafValues(vector<int>& arr) {
	
	stack<int>s;
	int sum=0;
	int t;
	for(int a:arr)
	{
	while(!s.empty() and a>s.top())
	{
	t=s.top();
	s.pop();
	if(s.empty())
	sum+=t*a;
	else
	sum+=t*min(s.top(),a);
	}
	s.push(a);
	}
	while(!s.empty())
	{
	t=s.top();
	s.pop();
	if(!s.empty())
	sum+=s.top()*t;
	}
	return sum;
	}

2.	**Easy Level: Daily Temperatures.**

Code:

```
Input: temperatures = [73,74,75,71,69,72,76,73]
Output: [1,1,4,2,1,1,0,0]
Input: temperatures = [30,40,50,60]
Output: [1,1,1,0]
```

```
vector<int> dailyTemperatures(vector<int>& temperatures) {

    int n=temperatures.size();
    stack<int>s;
    vector<int>ans(n,0);
    for(int i=0;i<n;i++)
    {
       while(s.size() and temperatures[s.top()]<temperatures[i])
       {
          ans[s.top()]=i-s.top();
          s.pop();
       }
       s.push(i);
    }
    return ans;
  }
```

3. **Medium Level: Distance of nearest cell having 1.**

Code:

```
Input: grid = {{0,1,1,0},{1,1,0,0},{0,0,1,1}}
Output: {{1,0,0,1},{0,0,1,1},{1,1,0,0}}
Explanation: The grid is-
0 1 1 0
1 1 0 0
```

```
0 0 1 1
0's at (0,0), (0,3), (1,2), (1,3), (2,0) and
(2,1) are at a distance of 1 from 1's at (0,1),
(0,2), (0,2), (2,3), (1,0) and (1,1)
respectively.
```

```
vector<vector<int>>nearest(vector<vector<int>>grid)
        {
            // Code here

            int n=grid.size();
            int m=grid[0].size();
            vector<vector<int>>ans(n,vector<int>(m,INT_MAX));
            queue<pair<int,int>>q;
            for(int i=0;i<n;i++)
            {
                for(int j=0;j<m;j++)
                {
                    if(grid[i][j]==1)
                    {
                        ans[i][j]=0;
                        q.push({i,j});
                    }
                }
            }
            while(!q.empty())
            {
                int i=q.front().first;
                int j=q.front().second;
                if((i-1)>=0 and ans[i][j]+1 < ans[i-1][j])
                {
                    ans[i-1][j]=ans[i][j]+1;
                    q.push({i-1,j});
                }
                if((j-1)>=0 and ans[i][j]+1 < ans[i][j-1])
                {
                    ans[i][j-1]=ans[i][j]+1;
```

```
                q.push({i,j-1});
            }
            if((i+1)<n and ans[i][j]+1 < ans[i+1][j])
            {
                ans[i+1][j]=ans[i][j]+1;
                q.push({i+1,j});
            }
            if((j+1)<m and ans[i][j]+1 < ans[i][j+1])
            {
                ans[i][j+1]=ans[i][j]+1;
                q.push({i,j+1});
            }
            q.pop();
        }
        return ans;
    }
```

4. **Medium Level : Online Stock Span.**

Code:

Input

```
["StockSpanner", "next", "next", "next", "next", "next", "next",
"next"]

[[], [100], [80], [60], [70], [60], [75], [85]]
```

Output

```
[null, 1, 1, 1, 2, 1, 4, 6]
```

Explanation

```
StockSpanner stockSpanner = new StockSpanner();

stockSpanner.next(100); // return 1

stockSpanner.next(80);  // return 1

stockSpanner.next(60);  // return 1

stockSpanner.next(70);  // return 2

stockSpanner.next(60);  // return 1
```

```
stockSpanner.next(75);  // return 4, because the last 4 prices
(including today's price of 75) were less than or equal to today's
price.

stockSpanner.next(85);  // return 6
```

```
stack<pair<int,int>>s;
   int index=-1;
   StockSpanner() {



   }

   int next(int price) {

      index+=1;
      while(!s.empty() and s.top().second<=price)//previous greater
element
         s.pop();
      //if no previous greater
      if(s.empty())
      {
         s.push({index,price});
         return index+1;
      }
         int res=s.top().first;
         s.push({index,price});
         return index-res;

   }
```

5. **Medium Level: Rotten Oranges.**

Code:

```
Input: grid = {{0,1,2},{0,1,2},{2,1,1}}

Output: 1

Explanation: The grid is-

0 1 2

0 1 2
```

```
2 1 1
Oranges at positions (0,2), (1,2), (2,0)
will rot oranges at (0,1), (1,1), (2,2) and
(2,1) in unit time.
```

```
int orangesRotting(vector<vector<int>>& grid) {
        // Code here
        queue<pair<int, int>> rotten;
        int r = grid.size(), c = grid[0].size(), fresh = 0, t = 0;
        for(int i = 0; i < r; ++i){
            for(int j = 0; j < c; ++j){
                if(grid[i][j] == 2) rotten.push({i, j});
                else if(grid[i][j] == 1) fresh++;
            }
        }

        while(!rotten.empty()){
            int num = rotten.size();
            for(int i = 0; i < num; ++i){
                int x = rotten.front().first, y = rotten.front().second;
                rotten.pop();
                if(x > 0 && grid[x-1][y] == 1)
                {
                    grid[x-1][y] = 2;
                    fresh--;
                    rotten.push({x-1, y});
                };
                if(y > 0 && grid[x][y-1] == 1)
                {
                    grid[x][y-1] = 2;
                    fresh--;
                    rotten.push({x, y-1});
                };
                if(x < r-1 && grid[x+1][y] == 1)
                {
                    grid[x+1][y] = 2;
                    fresh--;
```

```
                rotten.push({x+1, y});
            };
            if(y < c-1 && grid[x][y+1] == 1)
            {
                grid[x][y+1] = 2;
                fresh--;
                rotten.push({x, y+1});
            };
        }
        if(!rotten.empty()) t++;
    }
    return (fresh == 0) ? t : -1;
  }
```

6. **Medium Level: sum-of-subarray-minimums.**

Code:

```
Input: arr = [3,1,2,4]

Output: 17

Explanation:

Subarrays are [3], [1], [2], [4], [3,1], [1,2], [2,4], [3,1,2],
[1,2,4], [3,1,2,4].

Minimums are 3, 1, 2, 4, 1, 1, 2, 1, 1, 1.

Sum is 17.
```

```
int sumSubarrayMins(vector<int>& arr) {


    int n = arr.size(), mod = 1e9+7;
    long sum = 0;

    stack<pair<int,long>> st;


    for(int i=n-1; i>=0; i--){

        while(!st.empty() && arr[i] <= arr[st.top().first]){
            st.pop();
        }
```

```
            if(st.empty()){
                st.push({i, (arr[i] * (n-i) % mod)});
            }



            else {
                st.push({i, (arr[i] * (st.top().first - i) % mod +
st.top().second)});
            }


            sum = (sum + st.top().second) % mod;
        }
        return sum;
    }
```

7. **Medium Level: Evaluate Reverse Polish Notation.**

Code:

```
Input: tokens = ["2","1","+","3","*"]

Output: 9

Explanation: ((2 + 1) * 3) = 9
```

```
int evalRPN(vector<string>& tokens) {

        stack<int>s;
        int i=0;
        while(i<tokens.size())
        {
            if(tokens[i]=="+" || tokens[i]=="-" || tokens[i]=="*" ||
tokens[i]=="/" )
            {
                int a=s.top();
                s.pop();
                int b=s.top();
                s.pop();
                if(tokens[i]=="+")
```

```
            {
                s.push(a+b);
            }
            if(tokens[i]=="-")
            {
                s.push(b-a);
            }
            if(tokens[i]=="*")
            {
                s.push(a*b);
            }
            if(tokens[i]=="/")
            {
                int x=b/a;
                s.push(x);
            }
            i++;
        }
        else
        {
            s.push(stoi(tokens[i]));
            i++;
        }
    }
    return s.top();
}
```

8. **Medium Level: Circular tour .**

Code:

```
Input:
N = 4
Petrol = 4 6 7 4
Distance = 6 5 3 5
Output: 1
Explanation: There are 4 petrol pumps with
amount of petrol and distance to next
petrol pump value pairs as {4, 6}, {6, 5},
```

	{7, 3} and {4, 5}. The first point from where truck can make a circular tour is 2nd petrol pump. Output in this case is 1 (index of 2nd petrol pump).
	
	int tour(petrolPump p[],int n)
	{
	//Your code here
	int totSum=0,currSum=0,j=0;
	for(int i=0;i<n;i++)
	{
	totSum+=p[i].petrol-p[i].distance;
	currSum+=p[i].petrol-p[i].distance;
	if(currSum<0)
	{
	j=i+1;
	currSum=0;
	}
	}
	return totSum<0?-1:j;
	}

9.	**Medium Level: Flatten Nested List Iterator.**
	Code:
	Input: nestedList = [[1,1],2,[1,1]]
	
	Output: [1,1,2,1,1]
	
	Explanation: By calling next repeatedly until hasNext returns false, the order of elements returned by next should be: [1,1,2,1,1].
	
	vector<int> flattenList;
	int index;
	
	NestedIterator(vector<NestedInteger> &nestedList)
	{
	index = 0;
	doDFS(nestedList);
	}

```
int next()
{
   return flattenList[index++];
}

bool hasNext()
{
   if (index < flattenList.size())
      return true;

   return false;
}

void doDFS(vector<NestedInteger> &nestedList)
{

   for (auto nestedInt : nestedList)
   {
      if (nestedInt.isInteger())
      {
         flattenList.push_back(nestedInt.getInteger());
      }
      else
      {
         auto list = nestedInt.getList();
         doDFS(list);
      }
   }
}
```