GeeksMan

Data Structure

Lesson 5

Trilok Kaushik

Founder of GeeksMan

Chirag Soni

Nupur Pahuja

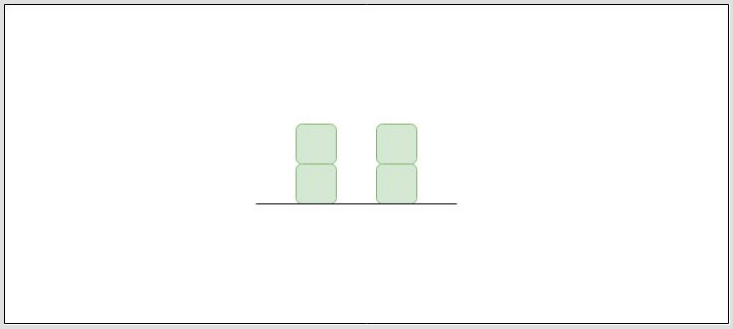
Jessica Mishra

Team Coordinators

TRAPPING RAIN WATER

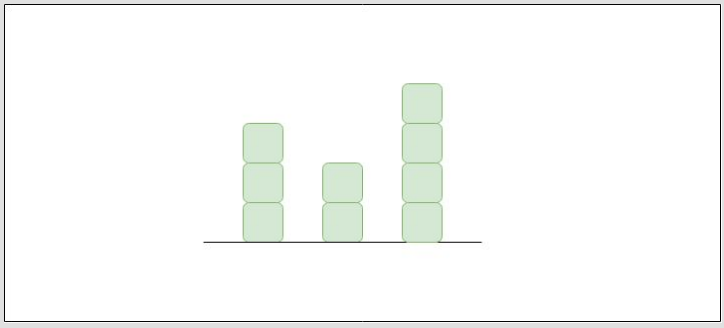
Given an array **arr[]** of **N** non-negative integers representing height of blocks at index **i** as **A_i** where the width of each block is 1. Compute how much water can be trapped in between blocks after raining.

Input: arr[]   = {2, 0, 2}
Output: 2
Explanation:
The structure is like below



We can trap 2 units of water in the middle gap.

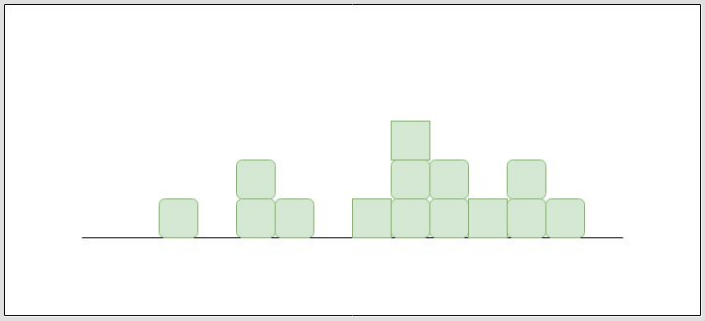
Input: arr[]   = {3, 0, 2, 0, 4}
Output: 7
Explanation:
Structure is like below



We can trap "3 units" of water between 3 and 2,
"1 unit" on top of bar 2 and "3 units" between 2
and 4. See below diagram also.

Input: arr[] = [0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1]
Output: 6

Explanation:
The structure is like below



Trap "1 unit" between first 1 and 2, "4 units" between first 2 and 3 and "1 unit" between second last 1 and last 2

Algorithm:

1. Traverse the array from start to end.
2. For every element (bar) , find the max value (bar with maximum height)on the left side as well as right side .
3. Take the minimum of these two values and subtract that particular element(height of the bar) for which the minimum of these maximums is calculated.This gives water stored by that bar.
4. Take the sum of water stored by each bar.

CODE :

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

vector<int> left(int a[] , int n)
{
    int i = 0;
    vector<int>v;
    int m = a[0];
```

```
10.      v.push_back(m);
11.      for(i=1;i<n;i++)
12.      {
13.          if(m < a[i])
14.          {
15.              m = a[i];
16.              v.push_back(m);
17.              continue;
18.          }
19.          v.push_back(m);
20.      }
21.      return v;
22.  }
23.  vector<int> right(int a[] , int n)
24.  {
25.      int i = 0;
26.      vector<int>v;
27.      int m = a[n-1];
28.      v.push_back(m);
29.      for(i=n-2;i>=0;i--)
30.      {
31.          if(m < a[i])
32.          {
33.              m = a[i];
34.              v.push_back(m);
35.              continue;
36.          }
37.          v.push_back(m);
38.      }
39.      vector<int>v1;
40.      for(i = 0 ; i<n ; i++)
41.          v1.push_back(v[n-i-1]);
42.      return v1;
43.  }
44.  int main()
45.  {
46.     int t;
```

```
    cin >> t;
    while(t--)
    {
        int n;
        cin >> n;
        int a[n];
        for(int i=0;i<n;i++)
            cin >> a[i];
        vector<int>vl = left(a,n);
        vector<int>vr = right(a,n);
        int m = 0;
        for(int i=0;i<n;i++)
        {
            m = m + min(vl[i] , vr[i]) - a[i];
        }
        cout << m << endl;
    }
  return 0;
}
```

Link for the code : https://sapphireengine.com/@/v63tpo

Longest valid Parentheses

Given a string S consisting of opening and closing parenthesis '(' and ')'.

Find length of the longest valid parentheses substring.

Input: (((**Output:** 2

Input:)()()) **Output:** 4

Algorithm:

1. Create an empty stack and push -1 to it. The first element of the stack is used to provide a base for the next valid string.
2. Initialize the result as 0.
3. If the character is '(' i.e. str[i] == '(', push index 'i' to the stack.

4. Else (if the character is ')')

   a) Pop an item from the stack (Most of the time an opening bracket)

   b) If the stack is not empty, then find the length of current valid substring by taking the difference between the current index and top of the stack. If current length is more than the result , then update the result.

   c) If the stack is empty, push the current index as a base for the next valid substring.

5. Return result.

CODE :

```
#include <iostream>
#include<bits/stdc++.h>
using namespace std;

int main() {
    int t;
    cin>>t;

    while(t--)
      {
          string s;
          stack<int> st;
          st.push(-1);
          cin>>s;
          int m=0,res,c=0,p;
          for(int i=0;i<s.length();i++)
          {
              if(s[i]=='(')
                  st.push(i);
              else
              {
                  if(st.size()==1)
                  {
                      st.push(i);
                  }
```

```
                    else
                    {
                        p=st.top();
                        st.pop();
                        if(s[p]=='(')
                            {
                                res=i-st.top();
                                m=max(m,res);
                            }
                        else
                            st.push(i);
                    }

                }
            }
            cout<<m<<endl;

        }
    }
```

Link for the code : https://sapphireengine.com/@/9bbgtc

SORT A STACK

There can be many solutions to this question. But let us focus on two methods for now , one by just traversing the stack and comparing each element with all the elements , and the other through recursion .Lets discuss both of them one by one.

SOLUTION 1

CODE :

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;
```

```
4.
5. class SortedStack
6. {
7.    public:
8.    stack<int>s;
9.    void sort();
10.   };
11.
12.  void printStack(stack<int>s)
13.  {
14.     while(!s.empty())
15.     {
16.         cout << s.top();
17.         s.pop();
18.     }
19.     cout << endl;
20.  }
21.  int main()
22.  {
23.    int t;
24.     cin >>t;
25.     while(t--)
26.     {
27.         SortedStack *ss = new SortedStack();
28.         int n;
29.         cin >> n;
30.         for(int i = 0;i<n;i++)
31.         {
32.             int k;
33.             cin >> k;
34.             ss->sort();
35.             printStack(ss->s);
36.         }
37.     }
38.    return 0;
39.  }
40.
41.
42.  void SortedStack :: sort()
43.  {
```

```
    stack<int> sorted;
    int t;
    while(!s.empty())
    {
        t=s.top();
        s.pop();
        if(sorted.empty())
        {
            sorted.push(t);

        }
        else if(sorted.top()<t)
        {
            while(!sorted.empty()&&sorted.top()<t)
            {
                s.push(sorted.top());
                sorted.pop();
            }
            sorted.push(t);
        }
        else
             sorted.push(t);
    }
    while(!sorted.empty())
    {
        s.push(sorted.top());
        sorted.pop();
    }
  }
```

https://sapphireengine.com/@/mzxh8i

SOLUTION 2

CODE:

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

class SortedStack
```

```
6. {
7.     public:
8.     stack<int>s;
9.     void sort();
10.    };
11.
12.   void printStack(stack<int>s)
13.   {
14.       while(!s.empty())
15.       {
16.           cout << s.top();
17.           s.pop();
18.       }
19.       cout << endl;
20.   }
21.   int main()
22.   {
23.      int t;
24.       cin >>t;
25.       while(t--)
26.       {
27.           SortedStack *ss = new SortedStack();
28.           int n;
29.           cin >> n;
30.           for(int i = 0;i<n;i++)
31.           {
32.               int k;
33.               cin >> k;
34.               ss->sort();
35.               printStack(ss->s);
36.           }
37.       }
38.      return 0;
39.   }
40.
41.
42.   void insert(stack<int>&s, int temp)
43.   {
44.       if(s.size() == 0 || s.top() <= temp)
45.       {
```

```
46.           s.push(temp);
47.           return ;
48.       }
49.       else
50.       {
51.               int x = s.top();
52.               s.pop();
53.               insert(s,temp);
54.               s.push(x);
55.       }
56.   }
57.   void SortedStack :: sort()
58.   {
59.       if(s.size() == 1)
60.           return ;
61.       int temp = s.top();
62.       s.pop();
63.       sort(s);
64.       insert(s ,temp);
65.   }
```