Maximum Sum Subarray – Kadane's Algorithm

> *Note: It is assumed that there must be atleast one element selected, based on the second test case.*

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

int main() {
    int n; cin>>n;
    int mxa=INT_MIN, mxb=0, g;
    for (int i=0; i<n; i++){
        cin>>g;
        mxb+=g;
        if (mxb<0) mxb=g;
        mxa = max(mxa,mxb);
    }
    cout<<mxa<<endl;
    return 1;
}
```

Input format

```
length of array
array elements
```

Test Case 1

```
7
2 3 -8 7 -1 2 3
```

Output

```
11
```

Test Case 2

```
2
-2 -4
```

Output

```
-2
```

Time Complexity: $O(N)$
Space Complexity: $O(1)$

Maximum Product Subarray

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

int main() {
    int n; cin>>n;
    int nums[n];
    for (int i=0; i<n; i++) cin>>nums[i];

    int max_l = INT_MIN, max_r = INT_MIN;

    int p=1;
    for (int i=0; i<n; i++){
        p*=nums[i];
        max_l = max(max_l,p);
        if (p==0) p=1;
    }

    p=1;
    for (int i=n-1; i>=0; i--){
        p*=nums[i];
        max_r = max(max_r,p);
        if (p==0) p=1;
    }

    cout<<max(max_l, max_r)<<endl;
    return 1;
}
```

Input format

```
length of array
array elements
```

Test Case 1

```
6
-2 6 -3 -10 0 2
```

Output

```
180
```

Test Case 2

```
5
-1 -3 -10 0 60
```

Output

```
60
```

Time Complexity: $O(N)$
Space Complexity: $O(1)$

Searching in a Sorted Rotated Array

```
#include<bits/stdc++.h>
using namespace std;
#define FASTIO ios_base::sync_with_stdio(false);cin.tie(0);cout.tie(0)

int binsearch(vector<int> &arr, int val, int offset){
    int n = arr.size(), res=0;
    bool flag=0;
    int l=0, r=n-1, m=0;
    while (l<=r){
        m=(l+r)/2;
        if (arr[(m+offset)%n]==val) {
            flag=1; break;
        }
        else if (arr[(m+offset)%n]<val) l = m+1;
        else r = m-1;
    return (flag?(m+offset)%n:-1);
}

int main() {
    FASTIO;
    int n; cin>>n;
    int res = 0;
    vector<int> arr (n,0);
    int pivot = 0;
    for (int i=0; i<n; i++){
        cin>>arr[i];
        if (i>0 && arr[pivot]<arr[i]) pivot++;
    }
    pivot = (pivot+1)%n;
    int val; cin>>val;
    cout<<binsearch(arr,val,pivot);
    return 1;
}
```

Input format

```
length of array
array elements
the target
```

Test Case 1

```
7
4 5 6 7 0 1 2
3
```

Output

```
-1
```

Test Case 2

```
5
50 10 20 30 40
10
```

Output

```
1
```

Time Complexity: $O(N)$
(worst case when pivot is in last: $O(N + logN) = O(N)$)
optimized in most cases.
Space Complexity: $O(1)$

Container with Most Water

```
#include<bits/stdc++.h>
using namespace std;
#define FASTIO ios_base::sync_with_stdio(false);cin.tie(0);cout.tie(0)
```

```
int main() {
    int n; cin>>n;
    int arr[n];
    for (int i=0; i<n; i++){
        cin>>arr[i];
    }
    int l = 0, r = n-1, res=0, dist=0;
    while (l<r){
        res = max(res, (min(arr[r],arr[l]))*(r-l) );
        if (arr[l]<arr[r]) l++;
        else r--;
    }
    cout<<res<<endl;
    return 1;
}
```

Input format

```
length of array
array elements
```

Test Case 1

```
4
1 5 4 3
```

Output

```
6
```

Test Case 2

```
5
3 1 2 4 5
```

Output

```
6
```

Time Complexity: $O(N)$
Space Complexity: $O(1)$

Factorial

```
#include<bits/stdc++.h>
using namespace std;
#define FASTIO ios_base::sync_with_stdio(false);cin.tie(0);cout.tie(0)

int main() {
    int inp; cin>>inp;
    list<int> digits;
    digits.push_back(1);

    int carry = 0;
    for (int n=2; n<=inp; n++){
        int val = n, rem=0; carry=0;
        auto c = digits.begin();
        for (auto i= digits.begin(); i!=digits.end(); i++){
            rem = (val)*(*i); rem+=carry;
            *i = (rem%10);
            carry = rem/10;
        }
        while (carry){
            digits.push_back(carry%10);
            carry/=10;
        }

    }
    for (auto i = digits.rbegin(); i!=digits.rend(); i++){
            cout<<(*i);
    }
    return 1;
}```
#### Input format
```

length of array
array elements

```
#### Test Case 1
```

100

```
#### Output
```

9332621544394415268169923885626670049071596826438162146859296389521759999
32299
1560894146397615651828625369792082722375825118521091686400000000000000000
00000 00

```
#### Test Case 2
```

50

```
#### Output
```

30414093201713378043612608166064768844377641568960512000000000000

~~~

Time Complexity: $O(N)$
Space Complexity: $O(N)$

[Trapping Rain Water](https://leetcode.com/problems/trapping-rain-
water/description/)
```cpp
#include<bits/stdc++.h>
using namespace std;
#define FASTIO ios_base::sync_with_stdio(false);cin.tie(0);cout.tie(0)

int main() {
   int n; cin>>n;
   int arr[n];
   for (int i=0; i<n; i++){
       cin>>arr[i];
   }
   int leftmax[n], rightmax[n];
   leftmax[0] = arr[0];
   for (int i=1; i<n; i++){
       leftmax[i] = max(leftmax[i-1],arr[i]);
   }
   rightmax[n-1] = arr[n-1];
   for (int i=n-2; i>=0; i--){
       rightmax[i] = max(rightmax[i+1],arr[i]);
   }

   int res = 0;
   for (int i=1; i<n-1; i++){
        res+=min(leftmax[i],rightmax[i])-arr[i];
   }
   cout<<res<<endl;
   return 1;
}
~~~

### Input format

```
length of array
array elements
```

Test Case 1

```
7
3 0 1 0 4 0 2
```

Output

```
10
```

Test Case 2

```
5
3 0 2 0 4
```

Output

```
7
```

Time Complexity: $O(N)$
Space Complexity: $O(N)$

Chocolate Distribution Problem

```
#include<bits/stdc++.h>
using namespace std;
#define FASTIO ios_base::sync_with_stdio(false);cin.tie(0);cout.tie(0)

int main() {
    int n,g; cin>>n;
    priority_queue<int,vector<int>, greater<int>> pq;
    for (int i=0; i<n; i++){
        cin>>g;
        pq.push(g);
    }
    int m; cin>>m;
    int res = pq.top();
    m--;
    while (m--) pq.pop();
    res = pq.top()-res;
    cout<<res<<endl;
    return 1;
}
```

Input format

```
length of array
array elements
m
```

Test Case 1

```
7
7 3 2 4 9 12 56
3
```

Output

```
2
```

Test Case 2

```
7
7 3 2 4 9 12 56
5
```

Output

```
7
```

Time Complexity: $O(Mlog(N))$
Space Complexity: $O(N)$

Merge Overlapping Intervals

```
#include<bits/stdc++.h>
using namespace std;
#define FASTIO ios_base::sync_with_stdio(false);cin.tie(0);cout.tie(0)

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

priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>>
pq;
    pair<int,int> pss;
    for (int i=0; i<n; i++){
        cin>>pss.first;
        cin>>pss.second;
        pq.push(pss);
    }
    vector<vector<int>> res; int m=-1;
    while (!pq.empty()){
        pair<int,int> cs = pq.top();
        pq.pop();
        if (m<0) {res.push_back({cs.first,cs.second});m++;}
        else if (res[m][1]>=cs.first && res[m][1]<cs.second){
            res[m][1] = cs.second;
        } else if (res[m][1]<cs.first){
            res.push_back({cs.first,cs.second});
            m++;
        }
    }
    for (auto i: res){
        for (auto j: i) cout<<j<<' ';
        cout<<endl;
    }
    return 1;

}
```

Input format

```
length of ranges
range 1
...
range length
```

Test Case 1

```
4
1 3
2 4
2 6
7 8
```

Output

```
1 6
7 8
```

Test Case 2

```
4
7 8
1 5
2 4
4 6
```

Output

```
1 6
7 8
```

Time Complexity: $O(Nlog(N))$
Space Complexity: $O(N)$

Boolean Matrix

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

int main() {
    int m,n; cin>>m>>n;
    vector<vector<int>> matrix (m,vector<int>(n,0));
    for (int i=0; i<m; i++) for (int j=0; j<n; j++) cin>>matrix[i][j];
    vector<bool> rs (m,0);
    vector<bool> cs (n,0);
    for (int i=0; i<m; i++){
        for (int j=0; j<n; j++){
            if (matrix[i][j]==1) {rs[i]=1;cs[j]=1;}
        }
    }
    for (int i=0; i<m; i++){
        for (int j=0; j<n; j++){
            if (rs[i] || cs[j]) matrix[i][j]=1;
            cout<<matrix[i][j]<<' ';
        }
        cout<<endl;
    }
}
```

Input format

```
dimensions of matrix
matrix elements
```

Test Case 1

```
2 2
1 0
0 0
```

Output

```
1 1
1 0
```

Test Case 2

```
3 4
1 0 0 1
0 0 1 0
0 0 0 0
```

Output

```
1 1 1 1
1 1 1 1
1 0 1 1
```

Time Complexity: $O(m * n)$
Space Complexity: $O(N)$

Spiral Matrix

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

#define FASTIO ios_base::sync_with_stdio(false); cin.tie(0); cout.tie(0)

int main(){
    FASTIO;
    int m,n; cin>>m>>n;
    vector<vector<int>> matrix (m,vector(n,0));
    for (int i=0; i<m; i++){
        for (int j=0; j<n; j++){
            cin>>matrix[i][j];
        }
    }
    vector<int> res;

    int i=0,j=0;

    while (i<=m/2 && j<=n/2){
        int flag=0;
        for (int k=j;k<=(n-j-1);k++){
            flag=1;
            res.push_back(matrix[i][k]);
        }
        if (flag!=1) break;
        for (int k=i+1;k<=(m-i-1);k++){
            flag=2;
            res.push_back(matrix[k][n-j-1]);
        }
        if (flag!=2 || j == (n-j-1)) break;
        for (int k=(n-j-1)-1;k>=j;k--){
            flag=3;
            res.push_back(matrix[m-i-1][k]);
        }
        if (flag!=3 || i == (m-i-1)) break;
        for (int k=(m-i-1)-1;k>i;k--){
            flag=4;
            res.push_back(matrix[k][j]);
        }
        if (flag!=4) break;
        i++;j++;
    }

    for (int i: res) cout<<i<<' ';
    return 1;
}
```

Input format

```
dimensions of matrix
matrix elements
```

Test Case 1

```
4 4
1 2 3 4
5 6 7 8
```

```
9 10 11 12
13 14 15 16
```

Output

```
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
```

Test Case 2

```
3 6
1 2 3 4 5 6
7 8 9 10 11 12
13 14 15 16 17 18
```

Output

```
1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11
```

Time Complexity: $O(M * N)$
Space Complexity: $O(1)$

Valid Paranthesis Expression

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

int main(){
    int n1, n2; string s1,s2;
    cin>>n1>>s1>>n2>>s2;
    if (n1!=n2) {cout<<"False"<<endl; return;}
    vector<int> v1(26,0), v2(26,0);
    for (char i: s1) v1[i-97]++;
    for (char i: s2) v2[i-97]++;

    bool flag=0;
    for (int i=0; i<26; i++){
        if (v1[i]!=v2[i]){
            flag=1; break;
        }
    }
    cout<<(flag?"False":"True")<<endl;
}
```

Input format

```
expression length
expression sequence
```

Test Case 1

```
10
((()))()()
```

Output

```
Balanced
```

Test Case 2

```
8
()))(())
```

Output

```
Not Balanced
```

Time Complexity: $O(N^2)$
Space Complexity: $O(N^2)$

Longest Palindromic Substring

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

int main(){
    int n; cin>>n;
    string s; cin>>s;
    int res = 1;

    vector<vector<bool>> dp (n, vector<bool>(n,0));

    for (int i=0; i<n; i++){
        dp[i][i]=1;
    }

    int si=0;

    for (int i=0; i<n-1; i++){
        if (s[i]==s[i+1]) {
            dp[i][i+1]=1;
            si=i; res=2;
        }
    }
    for (int i=3; i<n; i++){
        for (int j=0; j<n-i+1; j++){
            int v = i+j-1;
            if (dp[j+1][v-1] && s[j]==s[v]) {
                dp[j][v]=1;
                if (i>res){
                    si=j;
                    res = i;
                }
            }

        }
    }
    for (int i=si; i<res; i++){
        cout<<s[i]<<' ';
    }
    return 1;
}
```

Test Case 1

```
5
geeks
```

Output

```
ee
```

Test Case 2

```
3
abc
```

Output

```
a
```

Time Complexity: $O(N)$
Space Complexity: $O(1)$

Longest Common Prefix using Sorting

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

#define FASTIO ios_base::sync_with_stdio(false); cin.tie(0); cout.tie(0)

void main(){
    FASTIO;
    int n, m=INT_MAX; cin>>n;
    vector<string> vs(n,""); string s;
    for (int i=0; i<n; i++){
        cin>>s;
        m = min(m,(int)s.size());
        vs[i] = s;
    }
    string res="";

    for (int i=0; i<m; i++){
        int flag=1;
        for(int j=1; j<n; j++){
            if (vs[j][i]!=vs[j-1][i]) {
                flag=0; break;
            }
        }
        if (flag) res+=vs[0][i];
        else break;
    }
    cout<<res<<endl;
}
```

Input format

```
expression length
expression sequence
```

Test Case 1

```
10
((()))()()
```

Output

```
Balanced
```

Test Case 2

```
8
()))((())
```

Output

```
Not Balanced
```

Time Complexity: $O(N)$
Space Complexity: $O(1)$

Maximum Depth or Height of Binary Tree

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

struct Node {
    int val;
    Node *left = nullptr;
    Node *right = nullptr;
};

int maxDepth(struct Node *root){
    if (root==nullptr) return 0;
    else if (root->left==nullptr && root->right==nullptr) return 1;
    int ldepth = (root->left)?maxDepth(root->left):0;
    int rdepth = (root->right)?maxDepth(root->right):0;
    return max(ldepth,rdepth)+1;
}
```

Test Case 1

```
int main() {
    struct Node *root = new Node; root->val = 12;
    root->left = new Node; root->left->val = 8;
    root->right = new Node; root->right->val = 18;
    root->left->left = new Node; root->left->left->val = 5;
    root->left->right = new Node; root->left->right->val = 11;
    cout<<maxDepth(root)<<endl;
    return 1;
}
```

Output

```
3
```

Test Case 2

```
int main() {
    struct Node *root = new Node; root->val = 1;
    root->left = new Node; root->left->val = 2;
    root->right = new Node; root->right->val = 3;
    root->left->left = new Node; root->left->left->val = 4;
    root->right->right = new Node; root->right->right->val = 5;
    root->right->right->left = new Node; root->right->right->left->val =
6;
    root->right->right->right = new Node; root->right->right->right->val =
7;
    cout<<maxDepth(root)<<endl;
    return 1;
}
```

Output

```
4
```