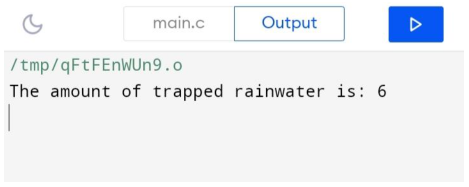
Date: 12.02.2024.   day: Monday

Trapping rainwater:

```
#include <stdio.h>

int trapRainWater(int height[], int size) {
    if (size <= 2) {
        return 0; // Not enough height to trap water
    }

    int leftMax[size];
    int rightMax[size];
    int waterTrapped = 0;
    leftMax[0] = height[0];
    for (int i = 1; i < size; i++) {
        leftMax[i] = (height[i] > leftMax[i - 1]) ? height[i] : leftMax[i - 1];
    }
    rightMax[size - 1] = height[size - 1];
    for (int i = size - 2; i >= 0; i--) {
        rightMax[i] = (height[i] > rightMax[i + 1]) ? height[i] : rightMax[i + 1];
    }


    for (int i = 0; i < size; i++) {
        int minHeight = (leftMax[i] < rightMax[i]) ? leftMax[i] : rightMax[i];
        waterTrapped += (minHeight - height[i]);
    }

    return waterTrapped;
}

int main() {
    int height[] = {0,1,0,2,1,0,1,3,2,1,2,1};
    int size = sizeof(height) / sizeof(height[0]);

    int trappedWater = trapRainWater(height, size);

    printf("The amount of trapped rainwater is: %d\n", trappedWater);

    return 0;
}
```

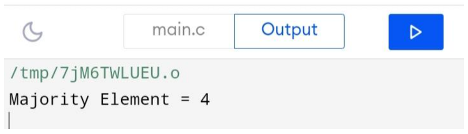
Output:.



```
/tmp/qFtFEnWUn9.o
The amount of trapped rainwater is: 6
```

<u>Date: 13.02.2024.   Tuesday</u>

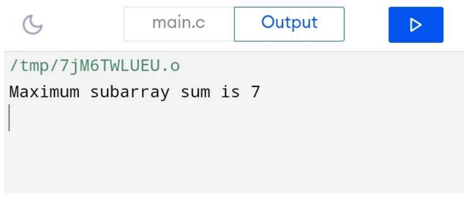
```
#include<stdio.h>

void getMajorityElement(int arr[], int n)
{
    int maxCount = 0;
    int index = -1; // sentinels
    for (int i = 0; i < n; i++) {
        int count = 0;
        for (int j = 0; j < n; j++) {
            if (arr[i] == arr[j])
                count++;
        }


        if (count > maxCount) {
            maxCount = count;
            index = i;
        }
    }

    if (maxCount > n / 2)
        printf("Majority Element = %d\n",arr[index]);

    else
        printf("No Majority Element\n");
}


int main()
{
    int arr[] = { 3, 3, 4, 2, 4, 4, 2, 4, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    getMajorityElement(arr, n);

    return 0;
}
```
Output:



```
/tmp/7jM6TWLUEU.o
Majority Element = 4
```

<u>Date:14.02.2024.   Wednesday</u>

```
#include <stdio.h>

int maxSubArraySum(int arr[], int n) {
    int maxEndingHere = arr[0];
    int maxSoFar = arr[0];

    for (int i = 1; i < n; i++) {
        maxEndingHere = (maxEndingHere + arr[i] > arr[i]) ? maxEndingHere + arr[i] : arr[i];
        maxSoFar = (maxEndingHere > maxSoFar) ? maxEndingHere : maxSoFar;
    }

    return maxSoFar;
}

int main() {
    int arr[] = {-2, -3, 4, -1, -2, 1, 5, -3};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Maximum subarray sum is %d\n", maxSubArraySum(arr, n));

    return 0;
}
```
Output:



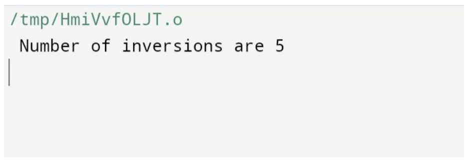
```
/tmp/7jM6TWLUEU.o
Maximum subarray sum is 7
```

15.02.2024.   Thursday

Count inversions:

```
#include <stdio.h>
#include <stdlib.h>
int getInvCount(int arr[], int n)
{
    int inv_count = 0;
    for (int i = 0; i < n - 1; i++)
        for (int j = i + 1; j < n; j++)
            if (arr[i] > arr[j])
                inv_count++;

    return inv_count;
}

int main()
{
    int arr[] = { 1, 20, 6, 4, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    printf(" Number of inversions are %d \n",
           getInvCount(arr, n));
    return 0;
}
```

Output:



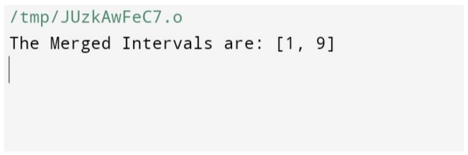
```
/tmp/HmiVvfOLJT.o
 Number of inversions are 5
```

16.02.2023   Friday

```
include <iostream>
#include <vector>
#include <stack>
#include <algorithm>
```

```
using namespace std;
 vector<vector<int>> overlappedInterval(vector<vector<int>>& intervals) {

    if (intervals.empty()) {

        return vector<vector<int>>();

    }

    sort(intervals.begin(), intervals.end(), [](const vector<int>& a, const vector<int>& b) {

        return a[0] < b[0];

    });


    stack<vector<int>> mergedStack;

    mergedStack.push(intervals[0]);


    for (int i = 1; i < intervals.size(); i++) {

        vector<int> current = intervals[i];

        vector<int>& top = mergedStack.top();


        if (current[0] <= top[1]) {



            top[1] = max(top[1], current[1]);

        } else {

            // If no overlap, push the current interval onto the stack

            mergedStack.push(current);

        }

    }
    vector<vector<int>> mergedIntervals;

    while (!mergedStack.empty()) {

        mergedIntervals.insert(mergedIntervals.begin(), mergedStack.top());

        mergedStack.pop();

    }


    return mergedIntervals;
}

int main() {

    vector<vector<int>> intervals = {{6, 8}, {1, 9}, {2, 4}, {4, 7}};

    vector<vector<int>> merged = overlappedInterval(intervals);


    cout << "The Merged Intervals are: ";

    for (const vector<int>& interval : merged) {

        cout << "[" << interval[0] << ", " << interval[1] << "] ";

    }

    cout << endl;


    return 0;
}
```

Output **:**

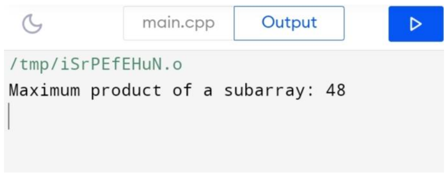


```
/tmp/JUzkAwFeC7.o
The Merged Intervals are: [1, 9]
```

17.02.2024.   Saturday

```
#include <iostream>
#include <vector>

using namespace std;

int maxProductSubarray(vector<int>& nums) {
    if (nums.empty()) {
        return 0;
    }

    int max_prod = nums[0];
    int min_prod = nums[0];
    int result = nums[0];

    for (int i = 1; i < nums.size(); i++) {
        if (nums[i] < 0) {
            swap(max_prod, min_prod);
        }

        max_prod = max(nums[i], max_prod * nums[i]);
        min_prod = min(nums[i], min_prod * nums[i]);

        result = max(result, max_prod);
    }

    return result;
}

int main() {
    vector<int> nums = {2, 3, -2, 4, -1};

    int result = maxProductSubarray(nums);

    cout << "Maximum product of a subarray: " << result << endl;

    return 0;
}
```

Output :

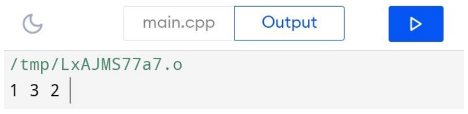


```
/tmp/iSrPEfEHuN.o
Maximum product of a subarray: 48
```

18.02.2024. Sunday

```
#include <iostream>
#include <algorithm>
#include <vector>
int main() {
    std::vector<int> nums = {1, 2, 3}; // example array
    std::next_permutation(nums.begin(), nums.end()); // find the next permutation
    // print the next permutation
    for (int num : nums) {
        std::cout << num << " ";
    }
    // expected output: 1 3 2
    return 0;
}
```

Output:



```
/tmp/LxAJMS77a7.o
1 3 2
```