Codility_

## Candidate Report: trainingSBW5Q6-9ZN

Check out Codility training tasks

Test Name:

Summary Timeline

### Tasks summary

| Task | Time spent | Score |
|---|---|---|
| **Triangle**<br>C | 7 min | 100% |

### Total score

100%

## Tasks Details

Easy

### 1. Triangle

Determine whether a triangle can be built from a given set of edges.

| Task Score | Correctness | Performance |
|---|---|---|
| 100% | 100% | 100% |

### Task description

An array A consisting of N integers is given. A triplet (P, Q, R) is *triangular* if 0 ≤ P < Q < R < N and:

- A[P] + A[Q] > A[R],
- A[Q] + A[R] > A[P],
- A[R] + A[P] > A[Q].

For example, consider array A such that:

```
A[0] = 10    A[1] = 2     A[2] = 5
A[3] = 1     A[4] = 8     A[5] = 20
```

Triplet (0, 2, 4) is triangular.

Write a function:

```
int solution(int A[], int N);
```

that, given an array A consisting of N integers, returns 1 if there exists a triangular triplet for this array and returns 0 otherwise.

For example, given array A such that:

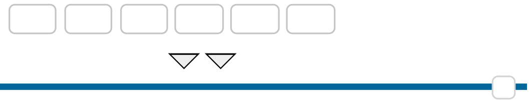
```
A[0] = 10    A[1] = 2     A[2] = 5
A[3] = 1     A[4] = 8     A[5] = 20
```

the function should return 1, as explained above. Given array A such that:

### Solution

| | |
|---|---|
| Programming language used: | C |
| Total time used: | 7 minutes |
| Effective time used: | 7 minutes |
| Notes: | *not defined yet* |

### Task timeline



08:01:04 08:07:20

Code: 08:07:20 UTC, c, final, score: **100**

show code in pop-up

```
A[0] = 10    A[1] = 50    A[2] = 5
A[3] = 1
```

the function should return 0.

Write an **efficient** algorithm for the following assumptions:

- N is an integer within the range [0..100,000];
- each element of array A is an integer within the range [−2,147,483,648..2,147,483,647].

```
// you can write to stdout for debugging purposes, e.g.
// printf("this is a debug message\n");

// Function to Merge Arrays L and R into A.
// lefCount = number of elements in L
// rightCount = number of elements in R.
void Merge(int *A,int *L,int leftCount,int *R,int rightCoun
        int i,j,k;

        // i - to mark the index of left aubarray (L)
        // j - to mark the index of right sub-raay (R)
        // k - to mark the index of merged subarray (A)
        i = 0; j = 0; k =0;

        while(i<leftCount && j< rightCount) {
                if(L[i]  < R[j]) A[k++] = L[i++];
                else A[k++] = R[j++];
        }
        while(i < leftCount) A[k++] = L[i++];
        while(j < rightCount) A[k++] = R[j++];
}

// Recursive function to sort an array of integers.
void MergeSort(int *A,int n) {
        int mid,i, *L, *R;
        if(n < 2) return; // base condition. If the array h

        mid = n/2;  // find the mid index.

        // create left and right subarrays
        // mid elements (from index 0 till mid-1) should be
        // and (n-mid) elements (from mid to n-1) will be p
        L = (int*)malloc(mid*sizeof(int));
        R = (int*)malloc((n- mid)*sizeof(int));

        for(i = 0;i<mid;i++) L[i] = A[i]; // creating left
        for(i = mid;i<n;i++) R[i-mid] = A[i]; // creating r

        MergeSort(L,mid);  // sorting the left subarray
        MergeSort(R,n-mid);  // sorting the right subarray
        Merge(A,L,mid,R,n-mid);  // Merging L and R into A
        free(L);
        free(R);
}

int solution(int A[], int N) {
    // write your code in C99 (gcc 6.2.0)
    MergeSort(A, N);
    for (int i = N-1; i>=2; i--){
        if ((A[i]-A[i-1])<A[i-2])
            return 1;
    }
    return 0;
}
```

## Analysis summary

The solution obtained perfect score.

## Analysis ?

Detected time complexity: **O(N*log(N))**

collapse all Example tests

▼ example ✓ OK

example, positive answer, length=6

1. 0.001 s **OK**

▼ example1 ✓ OK

example, answer is zero, length=4

1. 0.001 s **OK**

collapse all **Correctness tests**

▼ extreme_empty ✓ OK

empty sequence

1. 0.001 s **OK**
2. 0.001 s **OK**
3. 0.001 s **OK**
4. 0.001 s **OK**
5. 0.001 s **OK**
6. 0.001 s **OK**

▼ extreme_single ✓ OK

1-element sequence

1. 0.001 s **OK**
2. 0.001 s **OK**
3. 0.001 s **OK**
4. 0.001 s **OK**
5. 0.001 s **OK**
6. 0.001 s **OK**

▼ extreme_two_elems ✓ OK

2-element sequence

1. 0.001 s **OK**
2. 0.001 s **OK**
3. 0.001 s **OK**
4. 0.001 s **OK**
5. 0.001 s **OK**
6. 0.001 s **OK**

▼ extreme_negative1 ✓ OK

three equal negative numbers

1. 0.001 s **OK**
2. 0.001 s **OK**
3. 0.001 s **OK**
4. 0.001 s **OK**
5. 0.001 s **OK**
6. 0.001 s **OK**

▼ extreme_arith_overflow1 ✓ OK

overflow test, 3 MAXINTs

1. 0.001 s **OK**
2. 0.001 s **OK**
3. 0.001 s **OK**
4. 0.001 s **OK**

5. 0.001 s **OK**

6. 0.001 s **OK**

▼ extreme_arith_overflow2 ✓ **OK**

overflow test, 10 and 2 MININTs

1. 0.001 s **OK**

2. 0.001 s **OK**

3. 0.001 s **OK**

4. 0.001 s **OK**

5. 0.001 s **OK**

6. 0.001 s **OK**

▼ extreme_arith_overflow3 ✓ **OK**

overflow test, 0 and 2 MAXINTs

1. 0.001 s **OK**

2. 0.001 s **OK**

3. 0.001 s **OK**

4. 0.001 s **OK**

5. 0.001 s **OK**

6. 0.001 s **OK**

▼ medium1 ✓ **OK**

chaotic sequence of values from [0..100K], length=30

1. 0.001 s **OK**

2. 0.001 s **OK**

3. 0.001 s **OK**

4. 0.001 s **OK**

5. 0.001 s **OK**

6. 0.001 s **OK**

▼ medium2 ✓ **OK**

chaotic sequence of values from [0..1K], length=50

1. 0.001 s **OK**

2. 0.001 s **OK**

3. 0.001 s **OK**

4. 0.001 s **OK**

5. 0.001 s **OK**

6. 0.001 s **OK**

▼ medium3 ✓ **OK**

chaotic sequence of values from [0..1K], length=100

1. 0.001 s **OK**

2. 0.001 s **OK**

3. 0.001 s **OK**

4. 0.001 s **OK**

5. 0.001 s **OK**

6. 0.001 s **OK**

collapse all

Performance tests

▼ large1 ✓ **OK**

chaotic sequence with values from [0..100K], length=10K

1. 0.001 s **OK**
2. 0.001 s **OK**
3. 0.001 s **OK**
4. 0.001 s **OK**
5. 0.001 s **OK**
6. 0.001 s **OK**

▼ large2 ✓ **OK**

1 followed by an ascending sequence of ~50K elements from [0..100K], length=~50K

1. 0.012 s **OK**
2. 0.001 s **OK**
3. 0.001 s **OK**
4. 0.001 s **OK**
5. 0.001 s **OK**
6. 0.001 s **OK**

▼ large_random ✓ **OK**

chaotic sequence of values from [0..1M], length=100K

1. 0.024 s **OK**
2. 0.001 s **OK**
3. 0.001 s **OK**
4. 0.001 s **OK**
5. 0.001 s **OK**
6. 0.001 s **OK**

▼ large_negative ✓ **OK**

chaotic sequence of negative values from [-1M..-1], length=100K

1. 0.024 s **OK**
2. 0.001 s **OK**
3. 0.001 s **OK**
4. 0.001 s **OK**
5. 0.001 s **OK**
6. 0.001 s **OK**

▼ large_negative2 ✓ **OK**

chaotic sequence of negative values from [-10..-1], length=100K

1. 0.016 s **OK**
2. 0.001 s **OK**
3. 0.001 s **OK**

4. 0.001 s **OK**

5. 0.001 s **OK**

6. 0.001 s **OK**

▼ large_negative3 ✓ **OK**

sequence of -1 value, length=100K

1. 0.012 s **OK**

2. 0.001 s **OK**

3. 0.001 s **OK**

4. 0.001 s **OK**

5. 0.001 s **OK**

6. 0.001 s **OK**

The PDF version of this report that may be downloaded on top of this site may contain sensitive data including personal information. For security purposes, we recommend you remove it from your system once reviewed.