Full Name:	Keerthana S
Email:	keerthisrinu2003@gmail.com
Test Name:	**Mock Test**
Taken On:	19 Aug 2025 09:56:25 IST
Time Taken:	44 min 15 sec/ 90 min
Invited by:	Ankush
Invited on:	19 Aug 2025 09:56:07 IST
Skills Score:	
Tags Score:	Algorithms 290/290 Arrays 95/95 Core CS 290/290 Data Structures 215/215 Easy 95/95 Medium 75/75 Queues 120/120 Search 75/75 Sorting 95/95 Strings 95/95 problem-solving 170/170

100%

290/290

scored in **Mock Test** in 44 min 15 sec on 19 Aug 2025 09:56:25 IST

Recruiter/Team Comments:

No Comments.

Plagiarism flagged

We have marked questions with suspected plagiarism below. Please review it in detail here -

	Question Description	Time Taken	Score	Status
Q1	Truck Tour > Coding	10 min 58 sec	120/ 120	
Q2	Pairs > Coding	19 min 45 sec	75/ 75	
Q3	Big Sorting > Coding	8 min 42 sec	95/ 95	

QUESTION 1

Truck Tour > Coding Algorithms Data Structures Queues Core CS

Needs Review

Score 120

QUESTION DESCRIPTION

Suppose there is a circle. There are N petrol pumps on that circle. Petrol pumps are numbered 0 to $(N-1)$ (both inclusive). You have two pieces of information corresponding to each of the petrol pump: (1) the amount of petrol that particular petrol pump will give, and (2) the distance from that petrol pump to the next petrol pump.

Initially, you have a tank of infinite capacity carrying no petrol. You can start the tour at any of the petrol pumps. Calculate the first point from where the truck will be able to complete the circle. Consider that the truck will stop at each of the petrol pumps. The truck will move one kilometer for each litre of the petrol.

Input Format

The first line will contain the value of N.
The next N lines will contain a pair of integers each, i.e. the amount of petrol that petrol pump will give and the distance between that petrol pump and the next petrol pump.

Constraints:
$1 \leq N \leq 10^5$
$1 \leq \text{amount of petrol, distance} \leq 10^9$

Output Format

An integer which will be the smallest index of the petrol pump from which we can start the tour.

Sample Input

```
3
1 5
10 3
3 4
```

Sample Output

```
1
```

Explanation

We can start the tour from the second petrol pump.

CANDIDATE ANSWER

Language used: **C**

```c

/*
 * Complete the 'truckTour' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts 2D_INTEGER_ARRAY petrolpumps as parameter.
 */

int truckTour(int petrolpumps_rows, int petrolpumps_columns, int**
petrolpumps) {
    int s = 0;
    int f = 0;
    int o = 0;
    for (int i =0 ; i < petrolpumps_rows ; i++){

        int p = petrolpumps[i][0];
        int d = petrolpumps[i][1];
```

```
        int diff = p - d;

        f += diff;
        o += diff;
        if (f <0){
            s = i + 1;
            f = 0;
            }
 }
    if(o < 0){
    return -1;
 }
    return s;

}
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 1	Easy	Sample case	Success	0	0.0077 sec	7.25 KB
Testcase 2	Easy	Hidden case	Success	10	0.0077 sec	7.13 KB
Testcase 3	Easy	Hidden case	Success	10	0.0103 sec	7.25 KB
Testcase 4	Easy	Hidden case	Success	10	0.01 sec	7.13 KB
Testcase 5	Easy	Hidden case	Success	10	0.0318 sec	17.3 KB
Testcase 6	Easy	Hidden case	Success	10	0.0379 sec	17.3 KB
Testcase 7	Easy	Hidden case	Success	10	0.0475 sec	17.4 KB
Testcase 8	Easy	Hidden case	Success	10	0.0449 sec	17 KB
Testcase 9	Easy	Hidden case	Success	10	0.0349 sec	17 KB
Testcase 10	Easy	Hidden case	Success	10	0.0388 sec	17.1 KB
Testcase 11	Easy	Hidden case	Success	10	0.0441 sec	16.9 KB
Testcase 12	Easy	Hidden case	Success	10	0.0454 sec	17 KB
Testcase 13	Easy	Hidden case	Success	10	0.0328 sec	16.9 KB

No Comments

QUESTION 2

Correct Answer

Score 75

Pairs › Coding | Search | Algorithms | Medium | problem-solving | Core CS

QUESTION DESCRIPTION

Given an array of integers and a target value, determine the number of pairs of array elements that have a difference equal to the target value.

Example

$k = 1$

$arr = [1, 2, 3, 4]$

There are three values that differ by $k = 1$: $2 - 1 = 1$, $3 - 2 = 1$, and $4 - 3 = 1$. Return 3.

Function Description

Complete the *pairs* function below.

pairs has the following parameter(s):

- *int k:* an integer, the target difference
- *int arr[n]:* an array of integers

Returns

- *int:* the number of pairs that satisfy the criterion

Input Format

The first line contains two space-separated integers n and k, the size of arr and the target value.
The second line contains n space-separated integers of the array arr.

Constraints

- $2 \leq n \leq 10^5$
- $0 < k < 10^9$
- $0 < arr[i] < 2^{31} - 1$
- each integer $arr[i]$ will be unique

Sample Input

```
STDIN       Function
-----       --------
5 2         arr[] size n = 5, k =2
1 5 3 4 2   arr = [1, 5, 3, 4, 2]
```

Sample Output

```
3
```

Explanation

There are 3 pairs of integers in the set with a difference of 2: [5,3], [4,2] and [3,1]. .

CANDIDATE ANSWER

Language used: **C**

```c
int parse_int(char*);
int compare(const void *a, const void *b) {
    int x = *(int*)a;
    int y = *(int*)b;
    return x - y;
}

/*
 * Complete the 'pairs' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts following parameters:
 *  1. INTEGER k
 *  2. INTEGER_ARRAY arr
 */

int pairs(int k, int arr_count, int* arr) {
    qsort(arr, arr_count, sizeof(int), compare);
    int i = 0, j = 1;
    int count = 0;
 // two-pointer approach
    while (i < arr_count && j < arr_count) {
        int diff = arr[j] - arr[i];
        if (diff == k) {
            count++;
            i++;
            j++;
 }
        else if (diff < k) {
```

```
            j++;
 }
        else {
            i++;
            if (i == j) {
                j++;
 }
 }
 }
    return count;

}

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 1	Easy	Hidden case	Success	5	0.0075 sec	7.25 KB
Testcase 2	Easy	Hidden case	Success	5	0.007 sec	7.38 KB
Testcase 3	Easy	Hidden case	Success	5	0.0103 sec	7.25 KB
Testcase 4	Easy	Hidden case	Success	5	0.0074 sec	7.13 KB
Testcase 5	Easy	Hidden case	Success	5	0.0074 sec	7.25 KB
Testcase 6	Easy	Hidden case	Success	5	0.0088 sec	7.38 KB
Testcase 7	Easy	Hidden case	Success	5	0.0086 sec	7.25 KB
Testcase 8	Easy	Hidden case	Success	5	0.0111 sec	7.13 KB
Testcase 9	Easy	Hidden case	Success	5	0.011 sec	7.38 KB
Testcase 10	Easy	Hidden case	Success	5	0.0086 sec	7.63 KB
Testcase 11	Easy	Hidden case	Success	5	0.0311 sec	9.15 KB
Testcase 12	Easy	Hidden case	Success	5	0.0404 sec	9.31 KB
Testcase 13	Easy	Hidden case	Success	5	0.0503 sec	9.38 KB
Testcase 14	Easy	Hidden case	Success	5	0.0257 sec	9.39 KB
Testcase 15	Easy	Hidden case	Success	5	0.0272 sec	9.25 KB
Testcase 16	Easy	Sample case	Success	0	0.0075 sec	7.25 KB
Testcase 17	Easy	Sample case	Success	0	0.0092 sec	7.13 KB
Testcase 18	Easy	Sample case	Success	0	0.0078 sec	7.38 KB

No Comments

QUESTION 3

Correct Answer

Score 95

Big Sorting › Coding

Sorting | Strings | Algorithms | Easy | Data Structures | Arrays | problem-solving | Core CS

QUESTION DESCRIPTION

Consider an array of numeric strings where each string is a positive number with anywhere from 1 to 10^6 digits. Sort the array's elements in *non-decreasing*, or ascending order of their integer values and return the sorted array.

Example

$unsorted = ['1', '200', '150', '3']$

Return the array ['1', '3', '150', '200'].

Function Description

Complete the *bigSorting* function in the editor below.

bigSorting has the following parameter(s):

- *string unsorted[n]:* an unsorted array of integers as strings

Returns

- *string[n]:* the array sorted in numerical order

Input Format

The first line contains an integer, n, the number of strings in $unsorted$.
Each of the n subsequent lines contains an integer string, $unsorted[i]$.

Constraints

- $1 \leq n \leq 2 \times 10^5$
- Each string is guaranteed to represent a positive integer.
- There will be no leading zeros.
- The total number of digits across all strings in $unsorted$ is between 1 and 10^6 (inclusive).

Sample Input 0

```
6
31415926535897932384626433832795
1
3
10
3
5
```

Sample Output 0

```
1
3
3
5
10
31415926535897932384626433832795
```

Explanation 0

The initial array of strings is
$unsorted = [31415926535897932384626433832795, 1, 3, 10, 3, 5]$. When we order each string by the real-world integer value it represents, we get:

$$1 \leq 3 \leq 3 \leq 5 \leq 10 \leq 31415926535897932384626433832795$$

We then print each value on a new line, from smallest to largest.

Sample Input 1

```
8
1
2
100
123034798498573417183401923 71
3084193741082937
3084193741082938
111
200
```

Sample Output 1

```
1
2
100
111
200
3084193741082937
3084193741082938
1230347984985734171834019237 1
```

CANDIDATE ANSWER

Language used: **C**

```c
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);




/*
 * Complete the 'bigSorting' function below.
 *
 * The function is expected to return a STRING_ARRAY.
 * The function accepts STRING_ARRAY unsorted as parameter.
 */

/*
 * To return the string array from the function, you should:
 *     - Store the size of the array to be returned in the result_count
variable
 *     - Allocate the array statically or dynamically
 *
 * For example,
 * char** return_string_array_using_static_allocation(int* result_count) {
 *     *result_count = 5;
 *
 *     static char* a[5] = {"static", "allocation", "of", "string", "array"};
 *
 *     return a;
 * }
 *
 * char** return_string_array_using_dynamic_allocation(int* result_count) {
 *     *result_count = 5;
 *
 *     char** a = malloc(5 * sizeof(char*));
 *
 *     for (int i = 0; i < 5; i++) {
 *         *(a + i) = malloc(20 * sizeof(char));
 *     }
```

```
 *
 *     *(a + 0) = "dynamic";
 *     *(a + 1) = "allocation";
 *     *(a + 2) = "of";
 *     *(a + 3) = "string";
 *     *(a + 4) = "array";
 *
 *     return a;
 * }
 *
 */
char** bigSorting(int unsorted_count, char** unsorted, int* result_count) {
    int smaller(char *a, char *b) {
        int la = strlen(a);
        int lb = strlen(b);

        if (la != lb) return la < lb;   // shorter length = smaller number

        for (int i = 0; i < la; i++) {
            if (a[i] != b[i]) return a[i] < b[i];
        }
        return 0; // equal
    }

    // merge two sorted halves
    void merge(char **arr, int left, int mid, int right) {
        int n1 = mid - left + 1;
        int n2 = right - mid;

        char **L = malloc(n1 * sizeof(char*));
        char **R = malloc(n2 * sizeof(char*));

        for (int i = 0; i < n1; i++) L[i] = arr[left + i];
        for (int j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];

        int i = 0, j = 0, k = left;
        while (i < n1 && j < n2) {
            if (smaller(L[i], R[j])) arr[k++] = L[i++];
            else arr[k++] = R[j++];
        }
        while (i < n1) arr[k++] = L[i++];
        while (j < n2) arr[k++] = R[j++];

        free(L);
        free(R);
    }

    // recursive merge sort
    void mergeSort(char **arr, int l, int r) {
        if (l < r) {
            int m = (l + r) / 2;
            mergeSort(arr, l, m);
            mergeSort(arr, m + 1, r);
            merge(arr, l, m, r);
        }
    }

    // call mergesort
    mergeSort(unsorted, 0, unsorted_count - 1);

    *result_count = unsorted_count;
    return unsorted;

```

```c
}

int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    int n = parse_int(ltrim(rtrim(readline())));

    char** unsorted = malloc(n * sizeof(char*));

    for (int i = 0; i < n; i++) {
        char* unsorted_item = readline();

        *(unsorted + i) = unsorted_item;
    }

    int result_count;
    char** result = bigSorting(n, unsorted, &result_count);

    for (int i = 0; i < result_count; i++) {
        fprintf(fptr, "%s", *(result + i));

        if (i != result_count - 1) {
            fprintf(fptr, "\n");
        }
    }

    fprintf(fptr, "\n");

    fclose(fptr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;

    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) {
            break;
        }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n')
{
            break;
        }

        alloc_length <<= 1;

        data = realloc(data, alloc_length);

        if (!data) {
            data = '\0';

            break;
```

```
        }
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';

        data = realloc(data, data_length);

        if (!data) {
            data = '\0';
        }
    } else {
        data = realloc(data, data_length + 1);

        if (!data) {
            data = '\0';
        } else {
            data[data_length] = '\0';
        }
    }

    return data;
}

char* ltrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    while (*str != '\0' && isspace(*str)) {
        str++;
    }

    return str;
}

char* rtrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    char* end = str + strlen(str) - 1;

    while (end >= str && isspace(*end)) {
        end--;
    }

    *(end + 1) = '\0';

    return str;
}

int parse_int(char* str) {
    char* endptr;
    int value = strtol(str, &endptr, 10);
```

```
    if (endptr == str || *endptr != '\0') {
        exit(EXIT_FAILURE);
    }

    return value;
}
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 1	Easy	Sample case	Success	0	0.01 sec	7.38 KB
Testcase 2	Medium	Hidden case	Success	10	0.0079 sec	7.13 KB
Testcase 3	Medium	Hidden case	Success	10	0.0167 sec	7.75 KB
Testcase 4	Hard	Hidden case	Success	15	0.0287 sec	8.25 KB
Testcase 5	Hard	Hidden case	Success	15	0.0328 sec	8.5 KB
Testcase 6	Hard	Hidden case	Success	15	0.0319 sec	8.25 KB
Testcase 7	Hard	Hidden case	Success	15	0.0471 sec	9.64 KB
Testcase 8	Hard	Hidden case	Success	15	0.1757 sec	15.7 KB
Testcase 9	Easy	Sample case	Success	0	0.0089 sec	7.25 KB

No Comments

PDF generated at: 19 Aug 2025 05:13:06 UTC