AHSANIA MISSION UNIVERSITY OF SCIENCE & TECHNOLOGY

Lab Report-5

Lab No: 05

Course Code: CSE 2202

Course Title: Computer Algorithm Sessional.

Submitted By:

Md Sakline Hossen
ID: 1012320005101027
1st Batch, 2nd Year, 2nd Semester
Department of Computer science and Engineering,
Ahsania Mission University of Science & Technology

Submitted To:

Md. Fahim Faisal
Lecturer,
Department of Computer science and Engineering,
Ahsania Mission University of Science & Technology

Task No.: 01

Problem Statement: Merge Sort : Coun ng
Inversions

Sample Input   STDIN Func on

-----   --------

2   d = 2

5   arr[] size n = 5 for the first dataset

1 1 1 2 2   arr = [1, 1, 1, 2, 2]

5   arr[] size n = 5 for the second dataset

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

Sample Output

0

4

Source Code:

```
#include
<iostream> using
namespace std;
int Merge(int arr[], int le , int mid, int right)
{
    int n1 = mid - le + 1;
int n2 = right - mid;
int le Arr[n1],
rightArr[n2];
    for (int i = 0; i < n1; i++)
    {
        le Arr[i] = arr[le + i];
    }
    for (int i = 0; i < n2; i++)
    {
        rightArr[i] = arr[mid + 1 + i];
    }
    int i = 0, j = 0, k = le , invCount = 0;
    while (i < n1 && j < n2)
    {
        if (le Arr[i] <= rightArr[j])
        {
            arr[k] = le
Arr[i];
```

```
i++;         }
else
        {
            arr[k] = rightArr[j];
            invCount += (n1
- i);          j++;

}
k++
;
    }
    while (i < n1)
    {
        arr[k] = le
Arr[i];
i++;         k++;
    }

    while (j < n2)
    {
        arr[k] = rightArr[j];

j++;
k++;
    }
    return invCount;
}
int mergeAndCount(int arr[], int le , int right)
{
    if (le >= right)
    {
        return 0;
    }
    int mid = le + (right - le ) / 2;
    int invCount = 0;     invCount +=
mergeAndCount(arr, le , mid);     invCount
+= mergeAndCount(arr, mid + 1, right);
invCount += Merge(arr, le , mid, right);
return invCount;
}
int main()
{
int t;
cin>
>t;
while
(t--)
```

```
{         int n;
cin>>n;         int
arr[n];
for(int i=0; i<n;
i++)
        {
cin>>arr[i];
        }

        cout<< mergeAndCount(arr,0,n-1)<<endl;
    }
}
```

Output:

Task No.: 02

Problem Statement:

D. Merge Sort me limit per test

2 seconds
memory limit per test
256 megabytes

Merge sort is a well-known sor ng algorithm. The main func on that sorts the elements of array a with indices from [l, r) can be implemented as follows:

1. If the segment [l, r) is already sorted in non-descending order (that is, for any i such that l ≤ i < r 1 a[i] ≤ a[i + 1]), then end the func on call;
2. Let ; $mid = \lfloor \frac{l+r}{2} \rfloor$
3. Call mergesort(a, l, mid);
4. Call mergesort(a, mid, r);
5. Merge segments [l, mid) and [mid, r), making the segment [l, r) sorted in non-descending order. The merge algorithm doesn't call any other func ons.

The array in this problem is 0-indexed, so to sort the whole array, you need to call mergesort(a, 0, n).

The number of calls of func on mergesort is very important, so Ivan has decided to calculate it while sor ng the array. For example, if a = {1, 2, 3, 4}, then there will be 1 call of mergesort — mergesort(0, 4), which will check that the array is sorted and then end. If a = {2, 1, 3}, then the number of calls is 3: first of all, you call mergesort(0, 3), which then sets mid = 1 and calls mergesort(0, 1) and mergesort(1, 3), which do not perform any recursive calls because segments (0, 1) and (1, 3) are sorted.

Ivan has implemented the program that counts the number of mergesort calls, but now he needs to test it. To do this, he needs to find an array a such that a is a permuta on of size n (that is, the number of elements in a is n, and every integer number from [1, n] can be found in this array), and the number of mergesort calls when sor ng the array is exactly k.

Help Ivan to find an array he wants!

Input
The first line contains two numbers n and k ($1 \le n \le 100000$, $1 \le k \le 200000$) — the size of a desired permuta on and the number of mergesort calls required to sort it.

Output
If a permuta on of size n such that there will be exactly k calls of mergesort while sor ng it doesn't exist, output - 1. Otherwise output n integer numbers a[0], a[1], ..., a[n - 1] — the elements of a permuta on that would meet the required condi ons. If there are mul ple answers, print any of them.

Examples

Input
```
3 3
```
Output
```
2 1 3
```

Input
```
4 1
```
Output
```
1 2 3 4
```

Input
```
5 6
```
Output
```
-1
```

Source Code:

```
#include
<bits/stdc++.h> using
namespace std;
int n, k;
int a[100010];
```

```
void dfs(int l, int r)
{     if (k == 1 || l
+ 1 == r)
    {
        return;
    }     k -= 2;     int
mid = (l + r)/2;
swap(a[mid], a[mid
- 1]);     dfs(l, mid);
    dfs(mid, r);
}

int main()
{
    cin >> n
>> k;     if
(k % 2 ==
0)
    {
        cout << -1;
        return 0;
    }
    for (int i = 0; i < n; i++)
    {
a[i] = i +
1;
    }

dfs(0,
n);
if (k !=
1)
    {
        cout << -1;
    }
    else
    {
        for (int i = 0; i < n; i++)
        {
            prin ("%d ", a[i]);
        }
    }
}
```

Output:

```
3  3
2 1 3
Process returned 0 (0x0)   execution time : 16.709 s
Press any key to continue.
```

Task No.: 03

Problem Statement:

E. Binary Search

time limit per
test
2 seconds
memory limit
per test
256 megabytes

Anton got bored during the hike and wanted to solve something. He asked Kirill if he had any new problems, and of course, Kirill had one.

You are given a permuta on p of size n, and a number xx that needs to be found. A permuta on of length n is an array consis ng of n dis nct integers from 1 to n in arbitrary order. For example, [2,3,1,5,4] is a permuta on, but [1,2,2]is not a permuta on (2 appears twice in the array), and [1,3,4] is also not a permuta on (n=3 but there is 4 in the array).

You decided that you are a cool programmer, so you will use an advanced algorithm for the search — binary search. However, you forgot that for binary search, the array must be sorted.
You did not give up and decided to apply this algorithm anyway, and in order to get the correct answer, you can perform the following opera on no more than 22 mes before running the algorithm: choose the indices i, j (1≤i,j≤n) and swap the elements at posi ons i and j.
A er that, the binary search is performed. At the beginning of the algorithm, two variables l=1 and r=n+1 are declared. Then the following loop is executed:

1. If r−l, end the loop
2. m=⌊r+l2⌋
3. If pm≤x, assign l=m, otherwise r=m.

The goal is to rearrange the numbers in the permuta on before the algorithm so that a er the algorithm is executed, pl is equal to x. It can be shown that 2 opera ons are always sufficient.
Input
Each test consists of mul ple test cases. The first line contains a single integer (1≤t≤2·10) — the number of test cases. Then follow the descrip ons of the test cases.
The first line of each test case contains two integers n and x (1≤x≤n≤2·10) — the length of the permuta on and the number to be found. The second line contains the permuta on pp separated by spaces (1≤pi≤n).It is guaranteed that the sum of the values of nn for all test cases does not exceed 2·10.
Output
For each test case, output an integer kk (0≤k≤2) on the first line — the number of opera ons performed by you. In the next k lines, output 2 integers i, j (1≤i,j≤n) separated by a space, indica ng that you are swapping the elements at posi ons ii and j.

Note that you do not need to minimize the number of opera ons.

Source Code:

```
#include <iostream>
#include <vector>
#include <algorithm> // For sort()

using namespace std;

int main() {
    int testCases;
    cout << "Enter number of test cases: ";
    cin >> testCases;

    while (testCases--) {
        int n, x;
        cout << "\nEnter size of array (n) and the target number (x): ";
        cin >> n >> x;

        vector<int> arr(n);
        cout << "Enter " << n << " elements of the array: ";
        for (int i = 0; i < n; i++) {
            cin >> arr[i];
        }

        // Find the index of x in the array
        int indexOfX = -1;
        for (int i = 0; i < n; i++) {
            if (arr[i] == x) {
                indexOfX = i;
                break;
            }
        }

        if (indexOfX == -1) {
            cout << "Output: -1 (x is not in the array)\n";
            continue;
        }

        // Sorted array to find correct position of x
        vector<int> sortedArr = arr;
        sort(sortedArr.begin(), sortedArr.end());

        int targetPosition = -1;
        for (int i = 0; i < n; i++) {
            if (sortedArr[i] == x) {
                targetPosition = i;
                break;
```

```
        }
    }

    // Case 1: x is already at correct position
    if (arr[targetPosition] == x && indexOfX == targetPosition) {
        cout << "Output: 0 (x is already at the correct position)\n";
    }
    // Case 2: One swap can fix it
    else if (arr[targetPosition] == x) {
        cout << "Output:\n1\n";
        cout << indexOfX + 1 << " " << targetPosition + 1 << "\n";
    }
    // Case 3: Try two swaps
    else {
        bool foundTwoSwap = false;
        for (int i = 0; i < n; i++) {
            if (arr[i] == x) continue;

            swap(arr[i], arr[indexOfX]);

            if (arr[targetPosition] == x) {
                cout << "Output:\n2\n";
                cout << indexOfX + 1 << " " << i + 1 << "\n";
                cout << i + 1 << " " << targetPosition + 1 << "\n";
                foundTwoSwap = true;
                break;
            }

            swap(arr[i], arr[indexOfX]); // Undo
        }

        if (!foundTwoSwap) {
            cout << "Output: -1 (Not possible even with two swaps)\n";
        }
    }
}

return 0;
}
```
Output:

```
Enter number of test cases: 2

Enter size of array (n) and the target number (x): 5 2
Enter 5 elements of the array: 1 2 3 5 4
Output: 0 (x is already at the correct position)

Enter size of array (n) and the target number (x): 3 10
Enter 3 elements of the array: 1 2 3
Output: -1 (x is not in the array)

Process returned 0 (0x0)   execution time : 55.921 s
Press any key to continue.
```