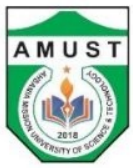

Ahsania Mission University of Science & Technology

Department of Computer Science and Engineering

1st Batch, 2nd Year 2st Semester, Fall 2025

Course Code: CSE 2202

Course Title:Computer Algorithms Sessional

Lab Report

Experiment No :05

Experiment Date :17/05/2025

Submission date:23/05/2025

Submitted To

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

Submitted By

Name : **MD:Rubyait Roman Rifat.**
ID : **1012320005101014**

Task No.: 01

Problem Statement: Merge Sort : Counting Inversions

Sample Input	STDIN Function
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 left, int mid, int right)
{
   int n1 = mid - left + 1;
   int n2 = right - mid;
   int leftArr[n1], rightArr[n2];
   for (int i = 0; i < n1; i++)
   {
      leftArr[i] = arr[left + i];
   }
   for (int i = 0; i < n2; i++)
   {
```

```
        rightArr[i] = arr[mid + 1 + i];
    }
    int i = 0, j = 0, k = left, invCount = 0;
    while (i < n1 && j < n2)
    {
        if (leftArr[i] <= rightArr[j])
        {
            arr[k] = leftArr[i];
            i++;
        }
        else
        {
            arr[k] = rightArr[j];
            invCount += (n1 - i);
            j++;
        }
        k++;
    }
    while (i < n1)
    {
        arr[k] = leftArr[i];
        i++;
        k++;
    }

    while (j < n2)
    {
        arr[k] = rightArr[j];
        j++;
        k++;
    }
    return invCount;
}
int mergeAndCount(int arr[], int left, int right)
{
    if (left >= right)
    {
        return 0;
    }
    int mid = left + (right - left) / 2;
    int invCount = 0;
```

```
    invCount += mergeAndCount(arr, left, mid);
    invCount += mergeAndCount(arr, mid + 1, right);
    invCount += Merge(arr, left, 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:

```
2
5
1 1 1 2 2
0
5
2 1 3 1 2
4

Process returned 0 (0x0)   execution time : 49.959 s
Press any key to continue.
```

Task No.: 02

Problem Statement:

D. Merge Sort

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

Merge sort is a well-known sorting algorithm. The main function 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 $l \le i < r - 1$ $a[i] \le a[i + 1]$), then end the function 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 functions.

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 function *mergesort* is very important, so Ivan has decided to calculate it while sorting 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 permutation 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 sorting 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 permutation and the number of *mergesort* calls required to sort it.

Output

If a permutation of size n such that there will be exactly k calls of *mergesort* while sorting it doesn't exist, output -1. Otherwise output n integer numbers $a[0], a[1], ..., a[n - 1]$ — the elements of a permutation that would meet the required conditions. If there are multiple 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++)
      {
         printf("%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 permutation p of size n, and a number xx that needs to be found. A permutation of length n is an array consisting of n distinct integers from 1 to n in arbitrary order. For example, $[2,3,1,5,4]$ is a permutation, but $[1,2,2]$is not a permutation (2 appears twice in the array), and $[1,3,4]$ is also not a permutation ($n=3$ 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 operation **no more than** 2 times before running the algorithm: choose the indices i, j ($1 \le i, j \le n$) and swap the elements at positions i and j.

After 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. $m = \lfloor \frac{r+l}{2} \rfloor$
3. If $p_m \le x$, assign $l = m$, otherwise $r = m$.

The goal is to rearrange the numbers in the permutation before the algorithm so that after the algorithm is executed, p_l is equal to x. It can be shown that 2 operations are always sufficient.

Input

Each test consists of multiple test cases. The first line contains a single integer t ($1 \le t \le 2 \cdot 10$) — the number of test cases. Then follow the descriptions of the test cases.

The first line of each test case contains two integers n and x ($1 \le x \le n \le 2 \cdot 10$) — the length of the permutation and the number to be found. The second line contains the permutation p separated by spaces ($1 \le p_i \le n$).It is guaranteed that the sum of the values of n for all test cases does not exceed $2 \cdot 10$.

Output

For each test case, output an integer k ($0 \le k \le 2$) on the first line — the number of operations performed by you. In the next k lines, output 2 integers i, j ($1 \le i, j \le n$) separated by a space, indicating that you are swapping the elements at positions i and j.

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

Example

Input

```
5
6 3
1 2 3 4 5 6
6 5
3 1 6 5 2 4
5 1
3 5 4 2 1
6 3
4 3 1 5 2 6
3 2
3 2 1
```

Output

```
0
```

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

Source Code:

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

using namespace std;

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

   while (testCases--) {
      int n, x;
      cin >> n >> x;

      vector<int> arr(n);

      // Read 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;
         }
```

```
}

// Simulate binary search to find where x would be
int left = 0, right = n - 1;
int targetPosition = -1;

while (left <= right) {
   int mid = (left + right) / 2;

   if (arr[mid] == x) {
      targetPosition = mid;
      break;
   }
   if (arr[mid] < x) {
      left = mid + 1;
   } else {
      right = mid - 1;
   }
}

// Case 1: x is already at the correct position
if (indexOfX == targetPosition) {
   cout << 0 << "\n";
}
// Case 2: One swap can place x at the right spot
else if (arr[targetPosition] == x) {
   cout << 1 << "\n";
   cout << indexOfX + 1 << " " << targetPosition + 1 << "\n"; // +1 for 1-based index
}
// 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]);

      // Try to do the binary search again
      int l = 0, r = n - 1;
      int newPos = -1;
      while (l <= r) {
         int mid = (l + r) / 2;
```

```
            if (arr[mid] == x) {
                newPos = mid;
                break;
            }
            if (arr[mid] < x) {
                l = mid + 1;
            } else {
                r = mid - 1;
            }
        }

        if (newPos == targetPosition) {
            cout << 2 << "\n";
            cout << indexOfX + 1 << " " << i + 1 << "\n";
            cout << i + 1 << " " << targetPosition + 1 << "\n";
            foundTwoSwap = true;
            break;
        }

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

      if (!foundTwoSwap) {
        cout << -1 << "\n";
      }
    }
  }

  return 0;
}
```

Output:

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