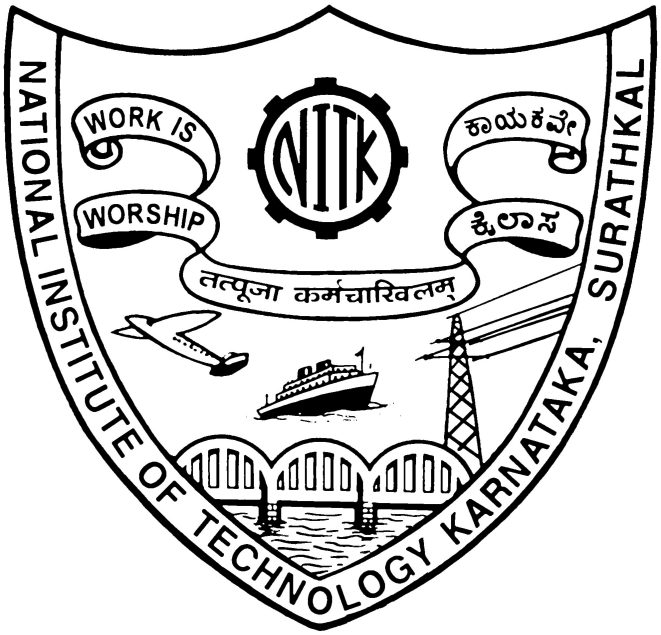
DSA PROJECT

Course Code: IT200



SUBMITTED TO

Mrs. Chaitra Bhat
Assistant Lecturer
Department of Information Technology
NITK Surathkal

SUBMITTED BY

Salman Shah- 15IT241
Vrishabh Sharma - 15IT242
Abhilekh Shrivastava - 15IT102
Neha Rajput – 15IT223

CONTENTS

PROBLEM NO.	CONTENT	PAGE
1	Problem Statement	2
1	Logic/Data Structure used	3
1	Code	4
1	Sample Outputs	6
2	Problem Statement	7
2	Logic/Data Structure used	9
2	Code	12
2	Sample Outputs	13

PROBLEM 1

There are N temples in a straight line and K monks who want to spread their enlightening power to the entire road of temples. All the monks have an enlightenment value, which denotes the range of enlightenment which they can spread in both the directions. Since, they do not want to waste their efficiency on trivial things of the world, they want to keep their range minimum.
Also, when we say that the N temples are in a straight line, we mean that that all the temples lie on something like an X-axis in a graph.
Find the minimum enlightenment value such that all the temples can receive it.

INPUT FORMAT

The first line contains two integers, N and K - denoting the number of temples and number of monks. The next line contains N integers denoting the position of the temples in the straight line.

OUTPUT FORMAT

Print the answer in a new line.

CONSTRAINTS

1 <= N <= 10^5
1 <= K < N
1 <= Position (i) <= 10^7
Update: The enlightenment value is an integer

SAMPLE INPUT

3 2
1 5 20

SAMPLE OUTPUT

2
5

EXPLANATION

The optimal answer is 2. A monk positioned at 3, can serve temples at 1 and 5. The other monk can be placed at 18, to serve temple at 20.

LOGIC/DATA STRUCTURE USED

Idea behind the question is to apply binary search to choose the minimum possible answer and then check whether it is possible or not. For this we will keep on decreasing the considered value until check fails. We all know whatever the minimum answer is, cannot be other than in between 0 to 10^ 7.So this is the range in which we apply binary search. Every time we will calculate the mid value of considered range [l, u] (taking l=1; u=10^7; initially) and will check whether this mid value (i.e. possible minimum enlightenment value or answer) is possible or not and will also check at mid-1. If the check at mid returns true, proceed to call check for mid-1. If the check for mid-1 is false that means mid is the minimum possible enlightenment value. Else, proceed to do binary search in the lower interval by setting u as mid-1. Also, if check at mid returns false, repeat the binary

search in the upper interval by setting l as mid+1.
Now how to check whether a solution at first sorts the input array taken from standard input, in ascending order. The main considered mid value is valid or not : In the given input array, take the first temple and keep a monk at a position at a distance of "mid" to the right of first temple and decrement the no. of monk by 1(Here this is denoted by temp--;(i.e. monk--)) Now if the effect of considered monk is up to a distance greater than the next temple, then keep on going to right (i.e. continue; i++;) if in between travelling from first to last temple monk decreases to less than zero, then check fails, otherwise keep on going right updating the position of recently put monk and decrementing the no. of monks. If we arrive to the last temple, with number of monks remaining greater than or equal to zero, then check succeeds.

COMPLEXITY

O(n log n) as sorting takes O(n log n) and we apply binary search log n times, each taking O(n) to completely traverse through the whole position array. This fits into the constraints listed in the problem as n<=105 and number of operations will in no way exceed 108. (Standard number of operations performed by compilers used by online judges is one second).

CODE

```
#include <stdio.h>

int canDo(int pos[], int k, int range, int n)
{
        int covered[n],i;
        for (i = 0; i < n; i++)
                covered[i] = 0;
        int cur = 0;
        while (cur < n)
        {
                if (k == 0)
                        return 0;
                int max = pos[cur] + range + range;
                while (cur < n && pos[cur] <= max)
                        covered[cur++] = 1;
```

```
            k--;
        }
        return 1;
}

void merge(int a[], int start, int end)
{
        int copy[end - start + 1],i;
        int mid = (start + end) / 2;
        int p = start, q = mid + 1;
        for (i = 0; i < end - start + 1; i++)
                if (p > mid)
                        copy[i] = a[q++];
                else if (q > end)
                        copy[i] = a[p++];
                else if (a[p] < a[q])
                        copy[i] = a[p++];
                else
                        copy[i] = a[q++];
        for (i = start; i <= end; i++)
                a[i] = copy[i - start];
}

void mergeSort(int a[], int start, int end)
{
        if (start < end)
        {
                int mid = (start + end) / 2;
                mergeSort(a, start, mid);
                mergeSort(a, mid + 1, end);
                merge(a, start, end);
        }
}

int main()
{
        int n, k,i;
        scanf("%d %d", &n, &k);
        int pos[n];
```

```
        for (i = 0; i < n; i++)
                scanf("%d", &pos[i]);
        mergeSort(pos, 0, n-1);
    int prev = 999999999;
    int start = 0, end = 10000000;
    while (1)
    {
        if (start > end)
                break;
        int mid = (start + end) / 2;
        int done = start == end;
        if (canDo(pos, k, mid, n) == 1)
        {
                if (mid < prev)
                        prev = mid;
                end = mid - 1;
        }
        else
                start = mid + 1;
        if (done)
                break;
    }
    printf("%d\n", prev);
}
```

SAMPLE OUTPUTS

TESTCASE 1

INPUT	OUTPUT
6 3 4 8 3 1 20 7	2

TESTCASE 2

INPUT	OUTPUT
10 3 99 121 4 2 3 7 96 124 130 98	5

PROBLEM 2

Fatal Eagle has decided to do something to save his favorite city against the attack of Mr. XYZ, since no one else surprisingly seems bothered about it, and are just suffering through various attacks by various different creatures.

Seeing Fatal Eagle's passion, N members of the Bangalore City decided to come forward to try their best in saving their city. Now Fatal Eagle decided to strategize these N people into a formation of AT LEAST K people in a group. Otherwise, that group won't survive.

Let's demonstrate this by an example. Let's say that there were 10 people, and each group required at least 3 people in it for its survival. Then, the following 5 groups can be made:

10 - Single group of 10 members.

7, 3 - Two groups. One consists of 7 members, the other one of 3 members.

6, 4 - Two groups. One consists of 6 members, the other one of 4 members.

5, 5 - Two groups. One consists of 5 members, the other one of 5 members.

4, 3, 3 - Three groups. One consists of 4 members, the other two of 3 members.

Given the value of N, and K - help Fatal Eagle in finding out the number of ways he can form these groups (anti-squads) to save his city.

INPUT FORMAT

The first line would contain, T - denoting the number of test cases, followed by two integers, N and K denoting the number of people who're willing to help and size of the smallest possible group which can be formed.

OUTPUT FORMAT

You've to print the number of ways in which groups can be formed**.**

CONSTRAINTS

1 <= T <= 30

1 <= N, K <= 200

SAMPLE INPUT

2

10 3

20 5

SAMPLE OUTPUT

5

13

LOGIC/DATA STRUCTURE USED

You can think of this problem as a special instance of integer partition problem. A partition of integer N is a non-decreasing sequence of positive integers such that their sum is equal to N. For example 1 + 1 + 2 + 4 is a partition of 8. Here 1, 1, 2 and 4 are called components of the partition.

In this problem, you can think of N - the number of people - as of integer which we partitioning while groups are our partition components. Let's now reformulate the problem:

For a given N and K, your task is to compute the number of partitions of N such that the smallest number in each partition is not less than K. Let call any such partition valid.

We have at most 30 testcases to solve and we will deal with each of them separately.

Let F(n, m) be the number of valid partitions of integer n, such that all numbers in every partition are not greater than m.

Clearly F(N, N) is the number we are searching for.

A great method for computing F(n, m) for any n, m <= N is to define a recursive relation for it, then define values for some base case and finally solving the relation.

Recursive relation

Let's divide the set of all partitions included in F(n, m) into two disjoint sets: A, B. Each partition in set A does not contain number m while each partition in set B contain m. Clearly A and B are disjoint and they cover all possible valid partitions included in F(n, m). In order to compute the size of A, we just need to compute F(n, m - 1) and in order to compute the size of B, we need to compute F(n - m, m), so our recursive relation has the following form:

$$F(n, m) = F(n, m - 1) + F(n - m, m)$$

We will use memoization to solve this relation, because if not, we would be solving the same subproblems many times which hurts the performance a lot.

Base cases

Let's define bases cases as follows:

F(0, m) := 1, which is very useful for computation and of course is reasonable because the only partition of 0 is an empty one

F(n, 0) := 0, for n != 0, because there is no valid partition which do not use any positive numbers

F(-n, m) := 0, because there is no valid partition of negative integers

F(n, m) := 0, for m < K, because a valid partition has all components not less than K

F(n, m) := 0, for n < K, because there is no way to partition integers less than K in a valid way

Let dp[N][N] be our memoization table and let NIL denotes that dp has not been computed yet for the given n and k. All records in dp table are initially set to NIL. The following recursive function, written in pseudocode, computes F(n, m). Calling F(N, N) solves the problem:

```
solve(n, m):
    if n == 0: return 1
    if m == 0: return 0
    if n < 0 or n < k or m < k: return 0
    if dp[n][m] == NIL:
        dp[n][m] = solve(n, m - 1) + solve(n - m, m)
    return dp[n][m]
```

COMPLEXITY

Solving function takes constant time not counting the recursive calls. We will be filling dp[n][m] for n, m <= N at most once and we return immediately if its value was previously computed, so our total time complexity is O(N2). Remember to set all entries in dp table to NIL after each testcase - this also takes O(N2).

CODE

```
#include <stdio.h>


long long int M[205][205];
int main()
{
        int i,j,N,K,T;
   for(i=1;i<=200;i++)
   {
        for(j=1;j<=i;j++)
        {
                if(i==j)
                M[i][j]=1;
                else
                {
```

```
                    M[i][j]=M[i-j][j];
                }
        }
        for(j=i;j>=1;j--)
        {
                M[i][j]=M[i][j]+M[i][j+1];
        }

    }
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d %d",&N,&K);
        printf("%lld\n",M[N][K]);
    }
    return 0;
}
```

SAMPLE OUTPUTS

TESTCASE 1

INPUT	OUTPUT
3	
10 3	5
20 5	13
40 6	167

TESTCASE 2

INPUT	OUTPUT
4	
11 2	14
17 6	4
20 6	8
40 10	27