RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM 602 105

CS23331 Design and Analysis of Algorithms

Laboratory Record Note Book

Name : A R KRISHNA

Year / Branch / Section:

University Register No. :

College Roll No. : 240701274

Semester : III

Academic Year : 2025-26

RAJALAKSHMI ENGINEERING COLLEGE

An Autonomous Institution

BONAFIDE CERTIFICATE

Name: ……

Academic Year: …… Semester: …… Branch: ……

Register No. | |

Certified that this is the bonafide record of work done by the above student in the.....Laboratory during the academic year 2025- 2026

Signature of Faculty in-charge

Submitted for the Practical Examination held on……

Internal Examiner External Examiner

INDEX

EX.NO	DATE	NAME OF THE EXPERIMENT	GITHUB QR
1		Basic C Programming	
2		Time Complexity	
3		Brute Force	
4		Divide and Conquer	
5		Greedy Technique	
6		Dynamic Programming	

Convert the following algorithm into a program and find its time complexity using counter method.

```
void reverse(int n)
{
   int rev = 0, remainder;
   while (n != 0)
    {
        remainder = n % 10;
        rev = rev * 10 + remainder;
        n/= 10;

    }
print(rev);
}
```

Note: No need of counter increment for declarations and scanf() and count variable printf() statements.

Input:

A positive Integer n

Output:

Print the value of the counter variable

Answer:

```c
#include <stdio.h>

void reverse(int n, int*c)
{
    int rev=0, rem;
    (*c)++;
    while(n!=0){
        (*c)++;
        rem=n%10;
        (*c)++;
        rev=rev*10+rem;
        (*c)++;
        n/=10;
        (*c)++;
    }
    (*c)++;
    (*c)++;
}

int main()
{
    int n, c=0;
    scanf("%d",&n);
    reverse(n,&c);
    printf("%d",c);
}
```

	Input	Expected	Got	
✔	12	11	11	✔
✔	1234	19	19	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Convert the following algorithm into a program and find its time

complexity using counter method.

```
void function(int n)
{
    int c= 0;
    for(int i=n/2; i<n; i++)
        for(int j=1; j<n; j = 2 * j)
            for(int k=1; k<n; k = k * 2)
                c++;
}
```

Note: No need of counter increment for declarations and scanf() and count variable printf() statements.

Input:

A positive Integer n

Output:

Print the value of the counter variable

Answer:

```c
#include<stdio.h>

void function(int n, int *c1)
{
    int c=0;
    (*c1)++;
    for(int i=n/2;i<n;i++){
        (*c1)++;
        for(int j=1;j<n;j=2*j){
            (*c1)++;
            for(int k=1; k<n; k=k*2){
                (*c1)++;
                c++;
                (*c1)++;
            }
            (*c1)++;
        }
        (*c1)++;
    }
    (*c1)++;
}

int main()
{
    int n, c1=0;
    scanf("%d",&n);
    function(n,&c1);
    printf("%d",c1);
}
```

	Input	Expected	Got	
✔	4	30	30	✔
✔	10	212	212	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Convert the following algorithm into a program and find its time complexity using counter method.

```
Factor(num) {
{
   for (i = 1; i <= num;++i)
   {
    if (num % i== 0)
       {
         printf("%d ", i);
       }
    }
 }
```

Note: No need of counter increment for declarations and scanf() and counter variable printf() statement.

Input:

A positive Integer n

Output:

Print the value of the counter variable

```c
#include <stdio.h>
void factor(int n, int *c)
{
    for (int i = 1; i <= n; i++)
    {
        (*c)++;
        (*c)++;
        if (n % i == 0)
        {
            // printf("%d ", i);
            (*c)++;
        }
    }
    (*c)++;
    // return ;
}
int main()
{
    int n, c = 0;
    scanf("%d", &n);
    factor(n, &c);
    printf("%d", c);
}
```

Open block

	Input	Expected	Got	
✔	12	31	31	✔
✔	25	54	54	✔
✔	4	12	12	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Convert the following algorithm into a program and find its time complexity using the counter method.

```
void func(int n)
{
    if(n==1)
    {
      printf("*");
    }
    else
    {
     for(int i=1; i<=n; i++)
     {
       for(int j=1; j<=n; j++)
       {
          printf("*");
          printf("*");
          break;
       }
     }
   }
 }
```

Note: No need of counter increment for declarations and scanf() and count variable printf() statements.

Input:

A positive Integer n

Output:

Print the value of the counter variable

Answer: (penalty regime: 0 %)

```c
#include <stdio.h>

void func(int n)
{
    long long count = 0;

    count++; // if comparison
    if (n == 1)
    {
        count++; // printf("*")
    }
    else
    {
        // outer loop
        for (int i = 1; i <= n; i++)
        {
            count++; // outer loop condition check

            // inner loop
            for (int j = 1; j <= n; j++)
            {
                count++; // inner loop condition check
                count++; // printf("*")
                count++; // printf("*")
                count++; // break
                break;
            }
        }
        count++; // final outer loop condition check (false)
    }

    printf("%lld\n", count);
}

int main()
{
    int n;
    scanf("%d", &n);
    func(n);
    return 0;
}
```

	Input	Expected	Got	
✔	2	12	12	✔
✔	1000	5002	5002	✔
✔	143	717	717	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Convert the following algorithm into a program and find its time complexity using the counter method.

```
void function (int n)
{
    int i= 1;

    int s =1;

    while(s <= n)
    {
         i++;
         s += i;
     }
}
```

Note: No need of counter increment for declarations and scanf() and count variable printf() statements.

Input:

A positive Integer n

Output:

Print the value of the counter variable

For example:

Input	Result
9	12

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>

void function(int n,int* count){
    int i=1;
    ++(*count);
    int s=1;
    ++(*count);

    while(s<=n){
        ++(*count);
        i++;
        ++(*count);
        s+=i;
        ++(*count);
    }
    //printf("%d\n",counter);
}

int main()
{
    int count=0;
    int n;
    count++;
    scanf("%d",&n);
    function(n,&count);
    printf("%d\n",count);
    return 0;
}
```

	Input	Expected	Got	
✔	9	12	12	✔
✔	4	9	9	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Write a Program to Implement the Quick Sort Algorithm

Input Format:
The first line contains the no of elements in the list-n
The next n lines contain the elements.

Output:
Sorted list of elements

For example:

Input	Result
5 67 34 12 98 78	12 34 67 78 98

Answer:

```c
#include <stdio.h>

void swap(int* a, int* b) {
    int t = *a;
    *a = *b;
    *b = t;
}

int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

int main() {
    int n;
    scanf("%d", &n);
    int arr[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    quickSort(arr, 0, n - 1);

    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

	Input	Expected	Got	
✔	5 67 34 12 98 78	12 34 67 78 98	12 34 67 78 98	✔
✔	10 1 56 78 90 32 56 11 10 90 114	1 10 11 32 56 56 78 90 90 114	1 10 11 32 56 56 78 90 90 114	✔
✔	12 9 8 7 6 5 4 3 2 1 10 11 90	1 2 3 4 5 6 7 8 9 10 11 90	1 2 3 4 5 6 7 8 9 10 11 90	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Problem Statement:

Given a sorted array of integers say arr[] and a number x. Write a recursive program using divide and conquer strategy to check if there exist two elements in the array whose sum = x. If there exist such two elements then return the numbers, otherwise print as "No".

Note: Write a Divide and Conquer Solution

Input Format

First Line Contains Integer n – Size of array

Next n lines Contains n numbers – Elements of an array

Last Line Contains Integer x – Sum Value

Output Format

First Line Contains Integer – Element1

Second Line Contains Integer – Element2 (Element 1 and Elements 2 together sums to value "x")

Answer: (penalty regime: 0 %)

```c
#include <stdio.h>

void findPair(int arr[], int low, int high, int x) {
    if (low >= high) {
        printf("No\n");
        return;
    }

    int sum = arr[low] + arr[high];

    if (sum == x) {
        printf("%d\n%d\n", arr[low], arr[high]);
        return;
    } else if (sum < x) {
        findPair(arr, low + 1, high, x);
    } else {
        findPair(arr, low, high - 1, x);
    }
}

int main() {
    int n, x;
    scanf("%d", &n);

    int arr[n];
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    scanf("%d", &x);

    findPair(arr, 0, n - 1, x);

    return 0;
}
```

	Input	Expected	Got	
✔	4 2 4 8 10 14	4 10	4 10	✔
✔	5 2 4 6 8 10 100	No	No	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Problem Statement:

Given a sorted array and a value x, the floor of x is the largest element in array smaller than or equal to x. Write divide and conquer algorithm to find floor of x.

Input Format

First Line Contains Integer n – Size of array

Next n lines Contains n numbers – Elements of an array

Last Line Contains Integer x – Value for x

Output Format

First Line Contains Integer – Floor value for x

Answer: (penalty regime: 0 %)

```c
#include <stdio.h>

// Function to find floor of x in a sorted array
int findFloor(int arr[], int low, int high, int x)
{
    // If low and high cross each other
    if (low > high)
        return -1;

    // If last element is smaller than or equal to x
    if (x >= arr[high])
        return arr[high];

    // Find the middle point
    int mid = (low + high) / 2;

    // If mid element is floor of x
    if (arr[mid] == x)
        return arr[mid];

    // If x lies between mid-1 and mid
    if (mid > 0 && arr[mid - 1] <= x && x < arr[mid])
        return arr[mid - 1];

    // If x is smaller than mid element, then recur for left half
    if (x < arr[mid])
        return findFloor(arr, low, mid - 1, x);

    // Else recur for right half
    return findFloor(arr, mid + 1, high, x);
}
```

```c
// Driver program to test above functions
int main()
{
    int n;
    scanf("%d", &n);
    int arr[n];
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    int x;
    scanf("%d", &x);
    int index = findFloor(arr, 0, n - 1, x);
    if (index == -1)
        printf("Floor of %d doesn't exist in array", x);
    else
        printf("%d", index);
    return 0;
}
```

	Input	Expected	Got	
✔	6 1 2 8 10 12 19 5	2	2	✔
✔	5 10 22 85 108 129 100	85	85	✔
✔	7 3 5 7 9 11 13 15 10	9	9	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question 1 | Correct Mark 1.00 out of 1.00 Flag question

Given an array `nums` of size `n`, return *the majority element.*

The majority element is the element that appears more than `⌊n / 2⌋` times. You may assume that the majority element always exists in the array.

Example 1:

Input: nums = [3,2,3]

Output: 3

Example 2:

Input: nums = [2,2,1,1,1,2,2]

Output: 2

Constraints:

- `n == nums.length`
- `1 <= n <= 5 * 10^4`
- `-2^31 <= nums[i] <= 2^31 - 1`

For example:

Input	Result
3 3 2 3	3
7 2 2 1 1 1 2 2	2

Answer: (penalty regime: 0 %)

```c
#include <stdio.h>

int majorityElement(int* nums, int numsSize) {
    int count = 0;
    int candidate = 0;

    for (int i = 0; i < numsSize; i++) {
        if (count == 0) {
            candidate = nums[i];
        }
        count += (nums[i] == candidate) ? 1 : -1;
    }

    return candidate;
}

int main() {
    int n;
    scanf("%d", &n);

    int nums[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &nums[i]);
    }

    int result = majorityElement(nums, n);
    printf("%d\n", result);

    return 0;
}
```

	Input	Expected	Got	
✔	3 3 2 3	3	3	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Problem Statement

Given an array of 1s and 0s this has all 1s first followed by all 0s. Aim is to find the number of 0s. Write a program using Divide and Conquer to Count the number of zeroes in the given array.

Input Format

First Line Contains Integer m – Size of array

Next m lines Contains m numbers – Elements of an array

Output Format

First Line Contains Integer – Number of zeroes present in the given array.

Answer: (penalty regime: 0 %)

```c
#include <stdio.h>

// Function to find the first occurrence of 0 using binary search
int findFirstZero(int arr[], int low, int high) {
    if (high < low)
        return -1;

    int mid = low + (high - low) / 2;

    // Check if mid is the first zero
    if ((arr[mid] == 0) && (mid == 0 || arr[mid - 1] == 1))
        return mid;

    // If mid is 1, search in the right half
    if (arr[mid] == 1)
        return findFirstZero(arr, mid + 1, high);
    else
        return findFirstZero(arr, low, mid - 1);
}

int main() {
    int m;
    scanf("%d", &m);

    int arr[m];
    for (int i = 0; i < m; i++) {
        scanf("%d", &arr[i]);
    }

    int firstZeroIndex = findFirstZero(arr, 0, m - 1);

    int zeroCount = (firstZeroIndex == -1) ? 0 : m - firstZeroIndex;
    printf("%d\n", zeroCount);

    return 0;
}
```

	Input	Expected	Got	
✔	5 1 1 1 0 0	2	2	✔
✔	10 1 1 1 1 1 1 1 1 1 1	0	0	✔
✔	8 0 0 0 0 0 0 0 0	8	8	✔

✔	17 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0	2	2	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Write a program to take value V and we want to make change for V Rs, and we have infinite supply of each of the denominations in Indian currency, i.e., we have infinite supply of {1, 2, 5, 10, 20, 50, 100, 500, 1000} valued coins/notes, what is the minimum number of coins and/or notes needed to make the change.

Input Format:

Take an integer from stdin.

Output Format:

print the integer which is change of the number.

Example Input :

64

Output:

4

Explanaton:

We need a 50 Rs note and a 10 Rs note and two 2 rupee coins.

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>

int main()
{
    int n, ans=0;
    scanf("%d",&n);
    int arr[] = {1000, 500, 100, 50, 20, 10, 5, 2, 1};
    for(int i=0;i<9;i++){
        if((arr[i] % n) == arr[i])
        {
            n-=arr[i];
            i=0;
            ans++;
        }
        if(n==0)
        {
            break;
        }
    }
    printf("%d",ans);
    return 0;
}
```

	Input	Expected	Got	
✔	49	5	5	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question 1 Correct Mark 1.00 out of 1.00 Flag question

Assume you are an awesome parent and want to give your children some cookies. But, you should give each child at most one cookie.

Each child i has a greed factor g[i], which is the minimum size of a cookie that the child will be content with; and each cookie j has a size s[j]. If s[j] >= g[i], we can assign the cookie j to the child i, and the child i will be content. Your goal is to maximize the number of your content children and output the maximum number.

Example 1:

Input:

3

1 2 3

2

1 1

Output:

1

Explanation: You have 3 children and 2 cookies. The greed factors of 3 children are 1, 2, 3.

And even though you have 2 cookies, since their size is both 1, you could only make the child whose greed factor is 1 content.

You need to output 1.

Constraints:

1 <= g.length <= 3 * 10^4

0 <= s.length <= 3 * 10^4

1 <= g[i], s[j] <= 2^31 - 1

Answer: (penalty regime: 0 %)

```c
#include <stdio.h>

void sort(int *arr, int n){
    for(int i=0;i<n-1;i++){
        for(int j=0;j<n-i-1;j++){
            if(arr[j]>arr[j+1]){
                int temp=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
            }
        }
    }
}

int main()
{
    int x,y,ans=0,k=0;
    scanf("%d",&x);
    int child[x];
    for(int i=0;i<x;i++)
    {
        scanf("%d",&child[i]);
    }
    scanf("%d",&y);
    int cookies[y];
    for(int i=0;i<y;i++)
    {
        scanf("%d",&cookies[i]);
    }
    sort(child, x);
    sort(cookies, y);
    for(int i=0;i<x;i++)
    {
        for(int j=k;j<y;j++)
        {
            if(child[i] <= cookies[j])
            {
                ans++;
                k=j+1;
                break;
            }
        }
    }
    printf("%d",ans);
    return 0;
}
```

	Input	Expected	Got	
✔	2 1 2 3 1 2 3	2	2	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

A person needs to eat burgers. Each burger contains a count of calorie. After eating the burger, the person needs to run a distance to burn out his calories. If he has eaten i burgers with c calories each, then he has to run at least $3^i * c$ kilometers to burn out the calories. For example, if he ate 3 burgers with the count of calorie in the order: [1, 3, 2], the kilometers he needs to run are $(3^0 * 1) + (3^1 * 3) + (3^2 * 2) = 1 + 9 + 18 = 28$. But this is not the minimum, so need to try out other orders of consumption and choose the minimum value. Determine the minimum distance he needs to run. Note: He can eat burger in any order and use an efficient sorting algorithm.Apply greedy approach to solve the problem.

Input Format

First Line contains the number of burgers

Second line contains calories of each burger which is n space-separate integers

Output Format

Print: Minimum number of kilometers needed to run to burn out the calories

Sample Input

3

5 10 7

Sample Output

76

For example:

Test	Input	Result
Test Case 1	3 1 3 2	18

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

int compare(const void* a,const void* b){
    int a1= *(const int*)a, b1=*(const int*)b;
    return b1-a1;
}

int main(){
    int n;//no of burgers
    scanf("%d",&n);
    int calories[n];//calories of each burger
    for(int i=0;i<n;i++) scanf("%d",&calories[i]);

    qsort(calories,n,sizeof(calories[0]),compare);

    int kilometers=0;

    for(int i=0;i<n;i++) kilometers+=pow(n,i)*calories[i];

    printf("%d\n",kilometers);
    return 0;
}
```

	Test	Input	Expected	Got	
✔	Test Case 1	3 1 3 2	18	18	✔
✔	Test Case 2	4 7 4 9 6	389	389	✔
✔	Test Case 3	3 5 10 7	76	76	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Given an array of N integer, we have to maximize the sum of arr[i] * i, where i is the index of the element (i = 0, 1, 2, ..., N).Write an algorithm based on Greedy technique with a Complexity O(nlogn).

Input Format:

First line specifies the number of elements-n

The next n lines contain the array elements.

Output Format:

Maximum Array Sum to be printed.

Sample Input:

5

2 5 3 4 0

Sample output:

40

Answer: (penalty regime: 0 %)

```c
#include <stdio.h>
#include <stdlib.h>

int compare(const void *a, const void *b)
{
    return (*(int *)a - *(int *)b);
}

int main()
{
    int n;
    scanf("%d", &n);
    int arr[n];
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }
    qsort(arr, n, sizeof(int), compare);
    int maxSum = 0;
    for (int i = 0; i < n; i++)
    {
        maxSum += arr[i] * i;
    }
    printf("%d\n", maxSum);
    return 0;
}

```

	Input	Expected	Got	
✔	5 2 5 3 4 0	40	40	✔
✔	10 2 2 2 4 4 3 3 5 5 5	191	191	✔
✔	2 45 3	45	45	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Answer: (penalty regime: 0 %)

```c
#include <stdio.h>
#include <stdlib.h>

int ascending(const void *a, const void *b) {
    return (*(int *)a - *(int *)b);
}

int descending(const void *a, const void *b) {
    return (*(int *)b - *(int *)a);
}

int main() {
    int n;
    scanf("%d", &n);

    int array_One[n], array_Two[n];

    // Input array_One
    for (int i = 0; i < n; i++) {
        scanf("%d", &array_One[i]);
    }

    // Input array_Two
    for (int i = 0; i < n; i++) {
        scanf("%d", &array_Two[i]);
    }

    // Sort array_One ascending
    qsort(array_One, n, sizeof(int), ascending);

    // Sort array_Two descending
    qsort(array_Two, n, sizeof(int), descending);

    // Calculate minimum sum of products
    int min_sum = 0;
    for (int i = 0; i < n; i++) {
        min_sum += array_One[i] * array_Two[i];
    }

    printf("%d\n", min_sum);
    return 0;
}
```

	Input	Expected	Got	
✔	3 1 2 3 4 5 6	28	28	✔
✔	4 7 5 1 2 1 3 4 1	22	22	✔
✔	5 20 10 30 10 40 8 9 4 3 10	590	590	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Playing with Numbers:

Ram and Sita are playing with numbers by giving puzzles to each other. Now it was Ram term, so he gave Sita a positive integer 'n' and two numbers 1 and 3. He asked her to find the possible ways by which the number n can be represented using 1 and 3.Write any efficient algorithm to find the possible ways.

Example 1:

Input: *6*
Output:*6*
Explanation: *There are 6 ways to 6 represent number with 1 and 3*

1+1+1+1+1+1
3+3
1+1+1+3
1+1+3+1
1+3+1+1
3+1+1+1

Input Format

First Line contains the number n

Output Format

Print: The number of possible ways 'n' can be represented using 1 and 3

Sample Input

6

Sample Output

6

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>

long long countWays(int n)
{
    long long dp[n+1];

    dp[0]=1;
    for(int i=1;i<=n;i++){
        dp[i]=0;
        if(i>=1)
           dp[i]+=dp[i-1];
        if(i>=3)
            dp[i]+=dp[i-3];
    }
    return dp[n];
}

int main() {
    int n;
    scanf("%d", &n);

    printf("%lld\n", countWays(n));
    return 0;
}
```

	Input	Expected	Got	
✔	6	6	6	✔
✔	25	8641	8641	✔
✔	100	24382819596721629	24382819596721629	✔

Passed all tests! ✔

Correct

Marks for this submission: 10.00/10.00.

Playing with Chessboard:

Ram is given with an n*n chessboard with each cell with a monetary value. Ram stands at the (0,0), that the position of the top left white rook. He is been given a task to reach the bottom right black rook position (n-1, n-1) constrained that he needs to reach the position by traveling the maximum monetary path under the condition that he can only travel one step right or one step down the board. Help ram to achieve it by providing an efficient DP algorithm.

Example:

Input

3

1 2 4

2 3 4

8 7 1

Output:

19

Explanation:

Totally there will be 6 paths among that the optimal is

Optimal path value:1+2+8+7+1=19

Input Format

First Line contains the integer n

The next n lines contain the n*n chessboard values

Output Format

Print Maximum monetary value of the path

Answer: (penalty regime: 0 %)

```c
#include <stdio.h>
#define MAX 100

int max(int a, int b){
    return (a>b) ? a:b;
}

int main(){
    int n, board[MAX][MAX], dp[MAX][MAX];
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++)
        scanf("%d",&board[i][j]);
    }
    dp[0][0]=board[0][0];
    for(int j=1;j<n;j++)
        dp[0][j] = dp[0][j - 1] + board[0][j];

    for (int i = 1; i < n; i++)
        dp[i][0] = dp[i - 1][0] + board[i][0];

    for (int i = 1; i < n; i++) {
        for (int j = 1; j < n; j++) {
            dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]) + board[i][j];
        }
    }

    printf("%d\n", dp[n - 1][n - 1]);
    return 0;
}
```

	Input	Expected	Got	
✔	3 1 2 4 2 3 4 8 7 1	19	19	✔
✔	3 1 3 1 1 5 1 4 2 1	12	12	✔
✔	4 1 1 3 4 1 5 7 8 2 3 4 6 1 6 9 8	28	28	✔

Passed all tests! ✔

Correct

Marks for this submission: 10.00/10.00.

Given two strings find the length of the common longest subsequence(need not be contiguous) between the two.

Example:

s1: ggtabe

s2: tgatasb

s1	a	g	**g**	**t**	**a**	**b**	
s2	**g**	x	**t**	**x**	**a**	**y**	**b**

The length is 4

Solveing it using Dynamic Programming

For example:

Input	Result
aab azb	2

Answer: (penalty regime: 0 %)

```c
#include <stdio.h>
#include <string.h>

int max(int a, int b)
{
    if (a > b)
        return a;
    return b;
}
int help(int a, int b, char *x, char *y, int dp[][b + 1])
{
    if (a < 0 || b < 0)
        return 0;
    if (dp[a][b] != -1)
        return dp[a][b];
    if (x[a] == y[b])
        return 1 + help(a - 1, b - 1, x, y, dp);
    return dp[a][b] = max(help(a - 1, b, x, y, dp), help(a, b - 1, x, y, dp));
}
int main()
{
    char x[100], y[100];
    scanf("%s %s", x, y);
    int a = strlen(x), b = strlen(y);
    int dp[a][b];
    memset(dp, -1, sizeof(dp));
    printf("%d", help(strlen(x) - 1, strlen(y) - 1, x, y, dp));
    return 0;
}
```

	Input	Expected	Got	
✔	aab azb	2	2	✔
✔	ABCD ABCD	4	4	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Problem statement:

Find the length of the Longest Non-decreasing Subsequence in a given Sequence.

Eg:

Input:9

Sequence:[-1,3,4,5,2,2,2,2,3]

the subsequence is [-1,2,2,2,2,3]

Output:6

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
#include<limits.h>
#include<string.h>

int max(int a, int b){
    if(a>b)
       return a;
    return b;
}

int help(int *arr, int n, int i, int prev, int dp[][n+1])
{
    if(i==n)
       return 0;
    if(dp[i][prev+1]!= -1)
       return dp[i][prev+1];
    int l = help(arr, n, i+1, prev, dp);
    int r = INT_MIN;
    if (prev == -1 || arr[i] >= arr[prev])
        r=1+help(arr, n, i+1, i, dp);
    return dp[i][prev+1] = max(l,r);
}

int main()
{
    int n;
    scanf("%d",&n);
    int arr[n];
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    int dp[n][n + 1];
    memset(dp, -1, sizeof(dp));
    int ans = help(arr, n, 0, -1, dp);
    printf("%d", ans);
    return 0;
}
```

	Input	Expected	Got	
✔	9 -1 3 4 5 2 2 2 2 3	6	6	✔
✔	7 1 2 2 4 5 7 6	6	6	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Find Duplicate in Array.

Given a read only array of n integers between 1 and n, find one number that repeats.

Input Format:

First Line - Number of elements

n Lines - n Elements

Output Format:

Element x - That is repeated

For example:

Input	Result
5 1 1 2 3 4	1

Answer: (penalty regime: 0 %)

```c
#include <stdio.h>

int findDuplicate(const int *arr, int n) {
    int slow = arr[0];
    int fast = arr[0];

    // Phase 1: Finding the intersection point
    do {
        slow = arr[slow];
        fast = arr[arr[fast]];
    } while (slow != fast);

    // Phase 2: Finding the entrance to the cycle
    slow = arr[0];
    while (slow != fast) {
        slow = arr[slow];
        fast = arr[fast];
    }

    return slow;
}

int main() {
    int n;
    scanf("%d", &n);

    int arr[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    int duplicate = findDuplicate(arr, n);
    printf("%d", duplicate);

    return 0;
}
```

	Input	Expected	Got	
✔	11 10 9 7 6 5 1 2 3 8 4 7	7	7	✔
✔	5 1 2 3 4 4	4	4	✔
✔	5 1 1 2 3 4	1	1	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Find Duplicate in Array.

Given a read only array of n integers between 1 and n, find one number that repeats.

Input Format:

First Line - Number of elements

n Lines - n Elements

Output Format:

Element x - That is repeated

For example:

Input	Result
5 1 1 2 3 4	1

Answer: (penalty regime: 0 %)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int findDuplicate(int* arr, int n) {
    int* freq = (int*)calloc(n + 1, sizeof(int));
    if (freq == NULL) {
        return -1;
    }

    for (int i = 0; i < n; i++) {
        int current_num = arr[i];
        if (freq[current_num] == 1) {
            free(freq);
            return current_num;
        }
        freq[current_num] = 1;
    }

    free(freq);
    return -1;
}

int main() {
    int n;
    if (scanf("%d", &n) != 1 || n <= 0) {
        return 1;
    }
    int* arr = (int*)malloc(n * sizeof(int));
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    int duplicate = findDuplicate(arr, n);

    if (duplicate != -1) {
        printf("%d", duplicate);
    } else {
        printf("No duplicate found or error occurred.\n");
    }

    free(arr);

    return 0;
}
```

	Input	Expected	Got	
✔	11 10 9 7 6 5 1 2 3 8 4 7	7	7	✔
✔	5 1 2 3 4 4	4	4	✔
✔	5 1 1 2 3 4	1	1	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Find the intersection of two sorted arrays.

OR in other words,

Given 2 sorted arrays, find all the elements which occur in both the arrays.

Input Format

- The first line contains T, the number of test cases. Following T lines contain:

1. Line 1 contains N1, followed by N1 integers of the first array
2. Line 2 contains N2, followed by N2 integers of the second array

Output Format

The intersection of the arrays in a single line

Example

Input:

1

3 10 17 57

6 2 7 10 15 57 246

Output:

10 57

Input:

1

6 1 2 3 4 5 6

2 1 6

Output:

1 6

For example:

Input	Result
1 3 10 17 57 6 2 7 10 15 57 246	10 57

Answer: (penalty regime: 0 %)

```c
#include <stdio.h>

void findIntersection(int arr1[], int n1, int arr2[], int n2) {
    int i = 0, j = 0;
    while (i < n1 && j < n2) {
        if (arr1[i] < arr2[j])
            i++;
        else if (arr1[i] > arr2[j])
            j++;
        else {
            printf("%d ", arr1[i]);
            i++;
            j++;
        }
    }
    printf("\n");
}

int main() {
    int T;
    scanf("%d", &T);

    while (T--) {
        int n1;
        scanf("%d", &n1);
        int arr1[n1];
        for (int i = 0; i < n1; i++)
            scanf("%d", &arr1[i]);

        int n2;
        scanf("%d", &n2);
        int arr2[n2];
        for (int i = 0; i < n2; i++)
            scanf("%d", &arr2[i]);

        findIntersection(arr1, n1, arr2, n2);
    }

    return 0;
}
```

	Input	Expected	Got	
✔	1 3 10 17 57 6 2 7 10 15 57 246	10 57	10 57	✔
✔	1 6 1 2 3 4 5 6 2 1 6	1 6	1 6	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Find the intersection of two sorted arrays.

OR in other words,

Given 2 sorted arrays, find all the elements which occur in both the arrays.

Input Format

- The first line contains T, the number of test cases. Following T lines contain:

1. Line 1 contains N1, followed by N1 integers of the first array
2. Line 2 contains N2, followed by N2 integers of the second array

Output Format

The intersection of the arrays in a single line

Example

Input:

1

3 10 17 57

6 2 7 10 15 57 246

Output:

10 57

Input:

1

6 1 2 3 4 5 6

2 1 6

Output:

1 6

For example:

Input	Result
1 3 10 17 57 6 2 7 10 15 57 246	10 57

Answer: (penalty regime: 0 %)

```c
#include <stdio.h>
#include <stdlib.h>

void printIntersection(int* arr1, int n1, int* arr2, int n2) {
    int i = 0;
    int j = 0;
    int found = 0;

    while (i < n1 && j < n2) {
        if (arr1[i] < arr2[j]) {
            i++;
        }
        else if (arr2[j] < arr1[i]) {
            j++;
        }
        else {
            if (found) {
                printf(" ");
            }
            printf("%d", arr1[i]);
            found = 1;
            i++;
            j++;
        }
    }
    printf("\n");
}

```

```c
int main() {
    int T;
    scanf("%d", &T);

    while (T--) {
        int n1, n2;
        scanf("%d", &n1);
        int* arr1 = (int*)malloc(n1 * sizeof(int));
        if (arr1 == NULL) return 1;
        for (int i = 0; i < n1; i++) {
            scanf("%d", &arr1[i]);
        }

        // Read the second array
        scanf("%d", &n2);
        int* arr2 = (int*)malloc(n2 * sizeof(int));
        if (arr2 == NULL) {
            free(arr1);
            return 1;
        }
        for (int i = 0; i < n2; i++) {
            scanf("%d", &arr2[i]);
        }
        printIntersection(arr1, n1, arr2, n2);
        free(arr1);
        free(arr2);
    }

    return 0;
}
```

	Input	Expected	Got	
✔	1 3 10 17 57 6 2 7 10 15 57 246	10 57	10 57	✔
✔	1 6 1 2 3 4 5 6 2 1 6	1 6	1 6	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Given an array A of sorted integers and another non negative integer k, find if there exists 2 indices i and j such that A[j] - A[i] = k, i != j.

Input Format:

First Line n - Number of elements in an array

Next n Lines - N elements in the array

k - Non - Negative Integer

Output Format:

1 - If pair exists

0 - If no pair exists

Explanation for the given Sample Testcase:

YES as 5 - 1 = 4

So Return 1.

For example:

Input	Result
3 1 3 5 4	1

Answer: (penalty regime: 0 %)

```c
#include <stdio.h>
#include <stdlib.h>

int findPairWithDifference(int* arr, int n, int k) {
    if (n < 2) {
        return 0;
    }

    int i = 0;
    int j = 1;
    while (i < n && j < n) {
        int diff = arr[j] - arr[i];

        if (diff == k && i != j) {
            return 1;
        }
        else if (diff < k) {
            j++;
        }
        else {
            i++;
        }
        if (i == j) {
            j++;
        }
    }
    return 0;
}

```

```c
int main() {
    int n;
    int k;
    if (scanf("%d", &n) != 1) {
        return 1;
    }
    int* arr = (int*)malloc(n * sizeof(int));
    if (arr == NULL) {
        perror("Failed to allocate memory");
        return 1;
    }
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    scanf("%d", &k);

    int result = findPairWithDifference(arr, n, k);
    printf("%d\n", result);
    free(arr);

    return 0;
}
```

	Input	Expected	Got	
✔	3 1 3 5 4	1	1	✔
✔	10 1 4 6 8 12 14 15 20 21 25 1	1	1	✔
✔	10 1 2 3 5 11 14 16 24 28 29 0	0	0	✔
✔	10 0 2 3 7 13 14 15 20 24 25 10	1	1	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Given an array A of sorted integers and another non negative integer k, find if there exists 2 indices i and j such that A[j] - A[i] = k, i != j.

Input Format:

First Line n - Number of elements in an array

Next n Lines - N elements in the array

k - Non - Negative Integer

Output Format:

1 - If pair exists

0 - If no pair exists

Explanation for the given Sample Testcase:

YES as 5 - 1 = 4

So Return 1.

For example:

Input	Result
3 1 3 5 4	1

Answer: (penalty regime: 0 %)

```c
#include <stdio.h>
#include <stdlib.h>

int binarySearch(int* arr, int start, int end, int target) {
    while (start <= end) {
        int mid = start + (end - start) / 2;
        if (arr[mid] == target) {
            return 1;
        }
        if (arr[mid] < target) {
            start = mid + 1;
        }
        else {
            end = mid - 1;
        }
    }
    return 0;
}

int findPairWithDifference(int* arr, int n, int k) {
    for (int i = 0; i < n; i++) {
        int target = arr[i] + k;
        if (binarySearch(arr, i + 1, n - 1, target)) {
            return 1;
        }
    }
    return 0;
}

```

```c
int main() {
    int n;
    int k;
    if (scanf("%d", &n) != 1) {
        return 1;
    }
    int* arr = (int*)malloc(n * sizeof(int));
    if (arr == NULL) {
        perror("Failed to allocate memory");
        return 1;
    }
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    scanf("%d", &k);
    int result = findPairWithDifference(arr, n, k);
    printf("%d\n", result);
    free(arr);

    return 0;
}
```

	Input	Expected	Got	
✔	3 1 3 5 4	1	1	✔
✔	10 1 4 6 8 12 14 15 20 21 25 1	1	1	✔
✔	10 1 2 3 5 11 14 16 24 28 29 0	0	0	✔
✔	10 0 2 3 7 13 14 15 20 24 25 10	1	1	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.