FINDING TIME COMPLEXITY OF ALGORITHMS:

1.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

CODE:

```
#include<stdio.h>
int count=0;
void function (int n)
{
   int i= 1;
   count++;
   int s =1;
   count++;
   while(s <= n)
   {
      count++;
       i++;
       count++;
       s += i;
       count++;
    }
    count++;
    printf("%d",count);

}
```

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

2. 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

CODE:

```
#include<stdio.h>
int count=0;
void func(int n)
{
   if(n==1)
   {
      count++;
      //printf("*");
      count++;
   }
   else
   {
      count++;
    for(int i=1; i<=n; i++)
    {
       count++;
       for(int j=1; j<=n; j++)
```

```
        {
            count++;
            //printf("*");
            //printf("*");
            count=count+2;
            break;

        }
        count++;

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

3. 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

CODE:

```
#include<stdio.h>
int count=0;
void Factor(int num)
{
   int i;
```

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

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

4. 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

CODE:

```
#include<stdio.h>
int count=0;
void function(int n)
{
   int c=0;
```

```
    count++;
    for(int i=n/2; i<n; i++){
        count++;
        for(int j=1; j<n; j = 2*j){
        count++;
            for(int k=1; k<n; k = k*2){
                count++;//for K True
                c++;
                count++;
            }
            count++;//k false
        }count++;
    }count++;
    printf("%d",count);
}
int main(){
    int n;
    scanf("%d",&n);
    function(n);
    return 0;
}
```

5. 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

CODE:

```
#include<stdio.h>

int count=0;

void reverse(int n){
```

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

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

GREEDY ALGORITHM:

1.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.

CODE:

```
#include <stdio.h>
int minCoins(int V) {
    int denoms[] = {1, 2, 5, 10, 20, 50, 100, 500, 1000};
    int n = sizeof(denoms) / sizeof(denoms[0]);
```

```
    int count = 0;
    for (int i = n - 1; i >= 0; i--) {
        int currCount = V / denoms[i];
        count += currCount;
        V %= denoms[i];
    }
    return count;
}

int main() {
    int V;
    scanf("%d", &V);
    int minCount = minCoins(V);
    printf("%d\n", minCount);

    return 0;
}
```

2. 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

CODE:

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

// Function to compare two integers for qsort
int compare(const void* a, const void* b) {
    return (*(int*)a - *(int*)b);
}

int findContentChildren(int* g, int gSize, int* s, int sSize) {
    // Sort both arrays
    qsort(g, gSize, sizeof(int), compare);
    qsort(s, sSize, sizeof(int), compare);

    int i = 0, j = 0;
    int satisfied = 0;

    // Iterate through both arrays
    while (i < gSize && j < sSize) {
```

```
        if (s[j] >= g[i]) {
            // Cookie j can satisfy child i
            satisfied++;
            i++;
        }
        j++;
    }

    return satisfied;
}

int main() {
    // Example input
    int g[] = {1, 2, 3};
    int s[] = {1, 2};

    int gSize = sizeof(g) / sizeof(g[0]);
    int sSize = sizeof(s) / sizeof(s[0]);

    int result = findContentChildren(g, gSize, s, sSize);
    printf("%d\n", result);  // Output should be 1
  return 0;
}
```

3. 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

CODE:

```
#include <stdio.h>
#include<math.h>
int minDistance(int calories[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (calories[j] < calories[j + 1]) {
                int temp = calories[j];
                calories[j] = calories[j + 1];
                calories[j + 1] = temp;
            }
        }
    }

    int distance = 0;
    for (int i = 0; i < n; i++) {
        distance += pow(n,i)* calories[i];
    }
```

```
    return distance;
}

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

    int calories[n];

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

    int min = minDistance(calories, n);
    printf("%d\n", min);

    return 0;
}
```

4. 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

CODE:

```
#include <stdio.h>
#include <stdlib.h>
int compare(const void* a, const void* b) {
    return (*(int*)a - *(int*)b);
}

int maximizeSum(int* arr, int n) {
    qsort(arr, n, sizeof(int), compare);
```

```
    int max_sum = 0;
    for (int i = 0; i < n; i++) {
        max_sum += arr[i] * i;
    }
    return max_sum;
}

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

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

5. Given two arrays array_One[] and array_Two[] of same size N. We need to first rearrange the arrays such that the sum of the product of pairs(1 element from each) is minimum. That is SUM (A[i] * B[i]) for all i is minimum.

For example:

Input	Result
3 1 2 3 4 5 6	28

CODE:

```
#include <stdio.h>
void sortAscending(int arr[], int n) {
    int i, j, temp;
    for (i = 0; i < n-1; i++) {
        for (j = i+1; j < n; j++) {
            if (arr[i] > arr[j]) {
                temp = arr[i];
```

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

int main() {
    int n;

    scanf("%d", &n);
```

```
    int array_One[n], array_Two[n];

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

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

    sortAscending(array_One, n);
    sortDescending(array_Two, n);
    int minSum = 0;
    for(int i = 0; i < n; i++) {
        minSum += array_One[i] * array_Two[i];
    }

    printf("%d\n", minSum);

    return 0;
}
```

DIVIDE AND CONQUER

1. **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.

CODE:

```
#include <stdio.h>

int countZeroes(int arr[], int low, int high) {
    if (low == high) {
        return 1 - arr[low];
    }


int mid = (low + high) / 2;



    int leftZeroes = countZeroes(arr, low, mid);
    int rightZeroes = countZeroes(arr, mid + 1, high);
    return leftZeroes + rightZeroes;
```

```
}

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

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

    int zeroes = countZeroes(arr, 0, n - 1);
    printf("%d\n", zeroes);

    return 0;
}
```

2. 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

Input	Result
7 2 2 1 1 1 2 2	2

CODE:

```
#include <stdio.h>
int findMajority(int arr[], int left, int right) {
    if (left == right) {
        return arr[left];
    }
    int mid = left + (right - left) / 2;
    int leftMajority = findMajority(arr, left, mid);
    int rightMajority = findMajority(arr, mid + 1, right);
    if (leftMajority == rightMajority) {
        return leftMajority;
    }
    int leftCount = 0, rightCount = 0;
    for (int i = left; i <= right; i++) {
        if (arr[i] == leftMajority) {
            leftCount++;
        } else if (arr[i] == rightMajority) {
```

```
            rightCount++;
        }
    }
    return (leftCount > rightCount) ? leftMajority :
rightMajority;
}
int main() {
    int n;
    scanf("%d", &n);
    int arr[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    int majorityElement = findMajority(arr, 0, n - 1);
    printf("%d\n", majorityElement);
    return 0;
}
```

3. **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

CODE:

```
#include <stdio.h>

int findFloor(int arr[], int n, int x) {
    int left = 0, right = n - 1;
    int floorValue = -1; // Initialize floorValue to -1 (not found)

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

        // Check if mid element is less than or equal to x
        if (arr[mid] <= x) {
            floorValue = arr[mid]; // Update floorValue
```

```
            left = mid + 1; // Search in the right half
        } else {
            right = mid - 1; // Search in the left half
        }
    }

    return floorValue; // Return the floor value or -1 if not found
}

int main() {
    int n;
    scanf("%d", &n); // Read size of array

    int arr[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]); // Read elements of the array
    }

    int x;
    scanf("%d", &x); // Read value for x

    int result = findFloor(arr, n, x); // Find floor of x
```

```
    printf("%d\n", result); // Print the result
    return 0;
}
```

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”)

CODE:

```
#include <stdio.h>

void findPairWithSum(int arr[], int left, int right, int x) {

    if (left >= right) {
        printf("No\n");
        return;
    }

    int currentSum = arr[left] + arr[right];
```

```
    if (currentSum == x) {
        printf("%d\n", arr[left]);
        printf("%d\n", arr[right]);
        return;
    } else if (currentSum < x) {

        findPairWithSum(arr, left + 1, right, x);
    } else {

        findPairWithSum(arr, left, right - 1, x);
    }
}

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);

    findPairWithSum(arr, 0, n - 1, x);
    return 0;
}
```

5. 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

CODE:

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

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

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

    for (int j = low; j < high; 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 = (int*)malloc(n * sizeof(int));
    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");
    free(arr);

    return 0;
}
```

DYNAMIC PROGRAMMING:

1.

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

CODE:

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

long long countWays(int n) {
    // Create an array to store the number of ways to represent each number
    long long dp[n + 1];

    // Initialize base cases
    dp[0] = 1; // One way to represent 0
    dp[1] = 1; // One way to represent 1
    dp[2] = 1; // One way to represent 2
    dp[3] = 2; // Two ways to represent 3

    // Fill the dp array for all values from 4 to n
    for (int i = 4; i <= n; i++) {
        dp[i] = dp[i - 1] + dp[i - 3];
```

```
    }

    // The answer for n is stored in dp[n]
    return dp[n];
}

int main() {
    long long n;

    // Read the input number
    scanf("%lld", &n);

    // Check for negative input
    if (n < 0) {
        printf("0\n"); // or some error message
        return 0;
    }

    // Get the number of ways to represent n
    long long result = countWays(n);

    // Print the result
    printf("%lld\n", result);
```

```
    return 0;
}
```

2. 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

CODE:

```
#include <stdio.h>

#define MAX_N 100

int main() {
    int n;
    int board[MAX_N][MAX_N];
    long long dp[MAX_N][MAX_N];

    // Read the size of the chessboard
    scanf("%d", &n);

    // Read the chessboard values
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &board[i][j]);
        }
    }

    // Initialize the DP table
    dp[0][0] = board[0][0];
```

```
    // Fill the first row
    for (int j = 1; j < n; j++) {
        dp[0][j] = dp[0][j - 1] + board[0][j];
    }

    // Fill the first column
    for (int i = 1; i < n; i++) {
        dp[i][0] = dp[i - 1][0] + board[i][0];
    }

    // Fill the rest of the DP table
    for (int i = 1; i < n; i++) {
        for (int j = 1; j < n; j++) {
            dp[i][j] = board[i][j] + (dp[i - 1][j] > dp[i][j - 1] ? dp[i
- 1][j] : dp[i][j - 1]);
        }
    }

    // The result is in the bottom-right corner
    printf("%lld\n", dp[n - 1][n - 1]);

    return 0;
}
```

3.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

CODE:

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

#define MAX_LENGTH 100

int main() {
    char s1[MAX_LENGTH], s2[MAX_LENGTH];
    int dp[MAX_LENGTH][MAX_LENGTH];
    int m, n;

    // Input the two strings
    scanf("%s", s1);
    scanf("%s", s2);

    m = strlen(s1);
    n = strlen(s2);

    // Initialize the DP table
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0 || j == 0) {
```

```
            dp[i][j] = 0; // LCS of any string with an empty string is 0
        } else if (s1[i - 1] == s2[j - 1]) {
            dp[i][j] = dp[i - 1][j - 1] + 1; // Characters match
        } else {
            dp[i][j] = (dp[i - 1][j] > dp[i][j - 1]) ? dp[i - 1][j] : dp[i][j - 1]; // Characters do not match
        }
    }
}

// The length of the longest common subsequence
printf("%d\n", dp[m][n]);

return 0;
}
```

4. 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

CODE:

```
#include <stdio.h>

#define MAX_LENGTH 100

int main() {
    int n;
    int arr[MAX_LENGTH];
    int dp[MAX_LENGTH];

    // Input the size of the sequence
    scanf("%d", &n);
```

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

// Initialize the DP table
for (int i = 0; i < n; i++) {
    dp[i] = 1; // Each element is a subsequence of length 1
}

// Fill the DP table
for (int i = 1; i < n; i++) {
    for (int j = 0; j < i; j++) {
        if (arr[j] <= arr[i]) {
            dp[i] = (dp[i] > dp[j] + 1) ? dp[i] : (dp[j] + 1);
        }
    }
}

// Find the maximum value in the DP table
int max_length = 0;
for (int i = 0; i < n; i++) {
    if (dp[i] > max_length) {
```

```
            max_length = dp[i];
        }
    }

    // Output the length of the longest non-decreasing subsequence
    printf("%d\n", max_length);

    return 0;
}
```

COMPETITIVE PROGRAMMING:

1. 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

CODE:

```
#include <stdio.h>


int findDuplicate(int arr[], int n) {
    int slow = arr[0];
    int fast = arr[0];
    do {
```

```
        slow = arr[slow];
        fast = arr[arr[fast]];
    } while (slow != fast);
    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\n", duplicate);
return 0;
}
```

2. 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

CODE:

```
#include <stdio.h>


void findDuplicate(int arr[], int n) {
    int hash[n];
    for (int i = 0; i < n; i++) {
        hash[i] = 0;
    }
```

```
    for (int i = 0; i < n; i++) {
        if (hash[arr[i]] == 1) {
            printf("%d\n", arr[i]);
            return;
        }
        hash[arr[i]] = 1;
    }
}

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

    findDuplicate(arr,n);

    return 0;
}
```

3.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 5

CODE:

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

void find_intersection(int arr1[], int n1, int arr2[], int n2) {
    int i = 0, j = 0;
    int first = 1; // Flag to manage spacing in output

    while (i < n1 && j < n2) {
        if (arr1[i] == arr2[j]) {
            // Print the intersection element
```

```
            if (!first) {
                printf(" "); // Print space before next number
            }
            printf("%d", arr1[i]);
            first = 0; // Set flag to false after first element is
printed
            i++;
            j++;
        } else if (arr1[i] < arr2[j]) {
            i++;
        } else {
            j++;
        }
    }
    printf("\n"); // New line after each test case output
}

int main() {
    int T;
    scanf("%d", &T); // Read number of test cases

    while (T--) {
        int n1;
```

```
        scanf("%d", &n1); // Read size of first array
        int arr1[n1]; // Declare the first array
        for (int i = 0; i < n1; i++) {
            scanf("%d", &arr1[i]); // Read elements of first array
        }

        int n2;
        scanf("%d", &n2); // Read size of second array
        int arr2[n2]; // Declare the second array
        for (int i = 0; i < n2; i++) {
            scanf("%d", &arr2[i]); // Read elements of second
array
        }

        // Find and print the intersection of the two arrays
        find_intersection(arr1, n1, arr2, n2);
    }

    return 0;
}
```

4.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

CODE:

```
#include <stdio.h>

void findIntersection(int arr1[], int n1, int arr2[], int n2) {
    int i = 0, j = 0;
    int found = 0; // To check if we found any intersection

    while (i < n1 && j < n2) {
        if (arr1[i] == arr2[j]) {
            // Print the element if it's part of the intersection
            if (found == 0 || arr1[i] != arr1[i - 1]) { // Avoid
duplicates
```

```
            printf("%d ", arr1[i]);
            found = 1; // Mark that we found at least one
intersection
        }
        i++;
        j++;
    } else if (arr1[i] < arr2[j]) {
        i++;
    } else {
        j++;
    }
  }

  if (found) {
    printf("\n"); // New line after each test case result
  } else {
    printf("No intersection\n"); // If no common elements
were found
  }
}

int main() {
  int T; // Number of test cases
```

```
    scanf("%d", &T);

    for (int t = 0; t < T; t++) {
        int n1;
        scanf("%d", &n1); // Size of the first array
        int arr1[n1];
        for (int i = 0; i < n1; i++) {
            scanf("%d", &arr1[i]); // Elements of the first array
        }

        int n2;
        scanf("%d", &n2); // Size of the second array
        int arr2[n2];
        for (int i = 0; i < n2; i++) {
            scanf("%d", &arr2[i]); // Elements of the second array
        }

        // Find and print intersection
        findIntersection(arr1, n1, arr2, n2);
    }

    return 0;
}
```

5.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

CODE:

```
#include <stdio.h>

int has_pair_with_difference(int A[], int n, int k) {
    int i = 0, j = 0;

    while (j < n) {
        int diff = A[j] - A[i];

        if (diff == k && i != j) {
            return 1; // Pair found
        } else if (diff < k) {
            j++; // Increase the larger index
        } else {
            i++; // Increase the smaller index
        }

        // Ensure i is always less than j
        if (i == j) {
            j++;
        }
    }
```

```
    return 0; // No pair found
}

int main() {
    int n;
    scanf("%d", &n); // Read number of elements in the array

    int A[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &A[i]); // Read elements of the array
    }

    int k;
    scanf("%d", &k); // Read the non-negative integer k

    // Check if a pair exists
    int result = has_pair_with_difference(A, n, k);
    printf("%d\n", result);

    return 0;
}
```

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

CODE:

```
#include <stdio.h>

int has_pair_with_difference(int A[], int n, int k) {
    int i = 0, j = 0;

    while (j < n) {
        int diff = A[j] - A[i];

        if (diff == k && i != j) {
            return 1; // Pair found
        } else if (diff < k) {
            j++; // Increase the larger index to find a larger difference
        } else {
            i++; // Increase the smaller index to find a smaller difference
        }

        // Ensure i is always less than j
        if (i == j) {
            j++;
        }
```

```
    }

    return 0; // No pair found
}

int main() {
    int n;
    scanf("%d", &n); // Read number of elements in the array

    int A[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &A[i]); // Read elements of the array
    }

    int k;
    scanf("%d", &k); // Read the non-negative integer k

    // Check if a pair exists
    int result = has_pair_with_difference(A, n, k);
    printf("%d\n", result);

    return 0;
}
```