Finding Time Complexity of Algorithms

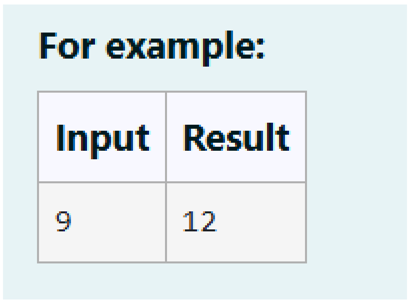
Problem 1: Finding Complexity using Counter Method

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

Program:

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

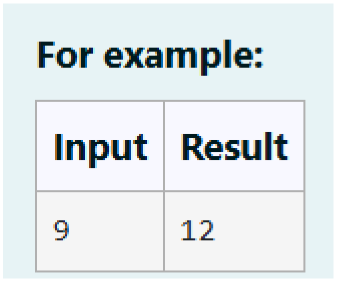
Problem 2: Finding Complexity using Counter method

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

Program:

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

Problem 3: Finding Complexity using Counter Method

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

Program:

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

count++;
printf("%d",count);
}
```

Problem 4: Finding Complexity using Counter Method

convert the following algorithm into a program and find its time complexity using a 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

Program:

```
#include<stdio.h>
int count=0;
void function(int n)
{
    count++;
    int c=0;
    count++;
    for(int i=n/2;i<n;i++)
    {
        count++;
        for(int j=1;j<n;j=j*2)
        {
            count++;
            for(int k=1;k<n;k=k*2)
            {
                count++;
                c++;
                count++;
            }
            count++;
        }
        count++;
    }
printf("%d",count);
}
int main()
{
    int n;
    scanf("%d",&n);
    function(n);
    return 0;
}
```

Problem 5: Finding Complexity using counter method

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

Program:

#include<stdio.h>
int reverse(int n)
{
    int count=0;
    int rev=0,remainder;
    count++;
    count++;
    while(n!=0)
    {
        count++;
        remainder=n%10;
        count++;
        rev=rev*10+remainder;
        count++;
        n/=10;
        count++;
    }
    count++;
    return count;
}
int main()
{
    int n,c;
    scanf("%d",&n);
    c=reverse(n);
    printf("%d",c);
}
```

Greedy Algorithms

1-G-Coin Problem

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.

Program:

```
#include<stdio.h>
int main()
{
    int n,count=0;
    scanf("%d",&n);
    int a[9]={1,2,5,10,20,50,100,500,1000};
    for(int i=8;i>=0;i--)
    {
        if(n>a[i])
        {
            n=n-a[i];
            count++;
        }
    }
    printf("%d",count);
}
```

2-G-Cookies Problem

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

Program:

```
#include<stdio.h>

#include<stdlib.h>


int main()

{

    int n,m;

    scanf("%d",&n);

    int a[n],c=0;
    for(int i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    scanf("%d",&m);
    int b[m];
    for(int i=0;i<m;i++)
    {
        scanf("%d",&b[i]);
    }
    for(int i=0;i<n;i++)
    {
        if(a[i]>=b[i])
        {
            c++;
        }
    }
    printf("%d",c);
}
```

3-G-Burger Problem

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

Program:
#include<stdio.h>
#include<math.h>
int main()
{
    int c,s=0;
    scanf("%d",&c);
    int a[c];
    for(int i=0;i<c;i++)
    {
        scanf("%d",&a[i]);
    }
    for(int i=0;i<c;i++)
    {
        for(int j=i+1;j<c;j++)
        {
            if(a[i]<a[j])
            {
                int t=a[i];
                a[i]=a[j];
                a[j]=t;
            }

        }
    }
    for(int i=0;i<c;i++)
    {
        int b=pow(c,i);
        s+=b*a[i];
    }
    printf("%d",s);
}
```

4-G-Array Sum max problem

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
```

Program:

```
#include<stdio.h>
#include<math.h>
int main()
{
    int a;
    scanf("%d",&a);
    int b[a],s=0;
    for(int i=0;i<a;i++)
    {
        scanf("%d",&b[i]);
    }
    for(int i=0;i<a;i++)
    {
        for(int j=i+1;j<a;j++)
        {
            if(b[i]>b[j])
            {
                int t=b[i];
                b[i]=b[j];
                b[j]=t;
            }
        }
    }
    for(int i=0;i<a;i++)
    {
        int u=(b[i]*i);
        s+=u;
    }
    printf("%d",s);
}
```

5-G-Product of Array elements-Minimum

Given two arrays array_One[] and array_Two[] of the 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	28
1	
2	
3	
4	
5	
6	

```
Program:
#include<stdio.h>
#include<math.h>
int main()
{
    int n;
    scanf("%d",&n);
    int a[n],b[n],s=0;
    for(int i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    for(int i=0;i<n;i++)
    {
        scanf("%d",&b[i]);
    }
    for(int i=0;i<n;i++)
    {


    for(int j=i+1;j<n;j++)
    {
        if(a[i]>a[j])
        {
            int t=a[i];
            a[i]=a[j];
            a[j]=t;
        }
    }
    }
    for(int i=0;i<n;i++)
    {
        for(int j=i+1;j<n;j++)
        {
            if(b[i]<b[j])
            {
                int t=b[i];
                b[i]=b[j];
                b[j]=t;
            }
        }
    }
    for(int i=0;i<n;i++)
    {
        s+=a[i]*b[i];
    }
    printf("%d",s);
}
```

Divide and Conquer

1-Number of Zeros in a Given Array

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.

Program:

```
#include<stdio.h>
int dac(int l,int u);
int a[100];
int c=0;
int main()
{
    int m;
    scanf("%d",&m);
    for(int i=0;i<m;i++)
    {
        scanf("%d",&a[i]);
    }
    int u=m-1;
    int l=0;
    int k=(dac(l,u));
    printf("%d",k);
}
int dac(int l,int u)
{
    if(l==u)
    {
        if(a[l]==0)
        c++;
    }
    else
    {
        int mid=(l+u)/2;
        dac(l,mid);
        dac(mid+1,u);
    }
    return c;
}
```

2-Majority Element

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 2 1 1 1 2 2	

Program:

```
#include<stdio.h>
int a[100];
int maj(int l,int h);
int main()
{
  int n;
  scanf("%d",&n);
  for(int i=0;i<n;i++)
  {
    scanf("%d",&a[i]);
  }
  int l=0;
  int h=n-1;
  printf("%d",maj(l,h));
}
int maj(int l,int h)
{
  if(l==h)
  {
    return a[l];
  }
  else
  {
    int m=(l+h)/2;
```

```
        int left=maj(l,m);
        int right=maj(m+1,h);
        int c1=0,c2=0;
        for(int i=l;i<h;i++)
        {
            if(a[i]==l)
            {
                c1++;
            }
            else
            c2++;
        }
        if(c1>c2)
        {
            return left;
        }
        else
        {
            return right;
        }

    }
}
```

3-Finding Floor Value

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

Program:

```
#include<stdio.h>
int a[100];
void find(int l,int h,int k)
{
   if(l>h)
   {
      printf("%d",a[h]);
      return;
   }
   int m=(l+h)/2;
   if(a[m]==k)
   {
      printf("%d",a[m]);
      return;
   }
   else if(a[m]<k)
   {
      find(m+1,h,k);
   }
   else
   {
      find(l,m-1,k);
   }
}
```

```
int main()
{
   int n;
   scanf("%d",&n);
   for(int i=0;i<n;i++)
   {
      scanf("%d",&a[i]);
   }
   int k;
   scanf("%d",&k);
   int l=0,h=n-1;
   find(l,h,k);
}
```

4-Two Elements sum to x

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”)

Program:

```
#include<stdio.h>
int a[100];
int main()
{
    int n;
    scanf("%d",&n);
    for(int i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    int x;
    scanf("%d",&x);
    int flag=0;
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            if(a[i]+a[j]==x && i!=j)
            {
                printf("%d\n%d",a[i],a[j]);
                i=n+1;
                j=n+1;
                flag=1;
            }
        }
    }
    if(flag==0)
    {
        printf("No");
    }
}
```

5-Implementation of Quick Sort

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

Program:

```
#include<stdio.h>
#include<stdlib.h>
int a[100];
void swap(int *a,int *b)
{
    int t=*a;
    *a=*b;
    *b=t;
```

```
}
int part(int l,int h)
{
    int p=a[h],i=l-1;
    for(int j=l;j<h;j++)
    {
        if(a[j]<p)
        {
            swap(&a[++i],&a[j]);
        }
    }
    swap(&a[i+1],&a[h]);
    return i+1;

}
void sort(int l,int h)
{
    if(l<h)
    {
        int pa=part(l,h);
        sort(l,pa-1);
        sort(pa+1,h);
    }
}
int main()
{
    int n;
    scanf("%d",&n);
    for(int i=0;i<n;i++)
```

```
  {
      scanf("%d",&a[i]);
  }
  int l=0,h=n-1;
  sort(l,h);
  for(int i=0;i<n;i++)
  {
      printf("%d ",a[i]);
  }
}
```

Dynamic Programming

1-DP-Playing with Numbers

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

Program:

```
#include<stdio.h>
long int comb(int n)
{
   long int dp[n+1];
   dp[0]=0;
   dp[1]=1;
   dp[2]=1;
   dp[3]=2;
   for(int i=4;i<=n;i++)
   {
      dp[i]=dp[i-1]+dp[i-3];
   }
   return dp[n];
}
int main()
{
   int n;
   scanf("%d",&n);
   printf("%ld",comb(n));
}
```

2-DP-Playing with chessboard

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

Program:

```
#include<stdio.h>

int main()

{

    int n;

    scanf("%d",&n);

    int a[n][n];

    for(int i=0;i<n;i++)

    {
```

```
    for(int j=0;j<n;j++)
    {
        scanf("%d",&a[i][j]);
    }
}
int dp[n][n];
dp[0][0]=a[0][0];
for(int i=0;i<n;i++)
{
    for(int j=0;j<n;j++)
    {
        dp[i][j]=0;
    }
}
for(int i=0;i<n;i++)
{
    for(int j=0;j<n;j++)
    {
        if(i!=0||j!=0)
        {
            if(j>0)
            {
                dp[i][j]=(dp[i][j]>dp[i][j-1])?dp[i][j]:dp[i][j-1];
            }
            if(i>0)
            {
                dp[i][j]=(dp[i][j]>dp[i-1][j])?dp[i][j]:dp[i-1][j];
            }
            dp[i][j]+=a[i][j];
            }
        }
    }
    printf("%d",(dp[n-1][n-1])+1);
}
```

3-DP-Longest Common Subsequence

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

Program:

```
#include<stdio.h>

#include<string.h>
```

```
int cls(char s1[],char s2[])
{
    int n=strlen(s1),m=strlen(s2);
    int dp[n+1][m+1];
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            dp[i][j]=0;
        }
    }
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
           if(s1[i-1]==s2[j-1])
           {
               dp[i][j]=dp[i-1][j-1]+1;
           }
           else
           {
               dp[i][j]=(dp[i-1][j]>dp[i][j-1])?dp[i-1][j]:dp[i][j-1];
           }
        }
    }
    return dp[n][m];
}
int main()
{
    char s1[5],s2[5];
    scanf("%s %s",s1,s2);
    printf("%d",cls(s1,s2));
}
```

[4-DP-Longest non-decreasing Subsequence]

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

Program:

```
#include<stdio.h>
int lnds(int a[],int n)
{
    int dp[n];
    dp[0]=1;
    int max=1;
    for(int i=1;i<n;i++)
    {
        dp[i]=1;
        for(int j=0;j<i;j++)
        {
            if(a[i]>=a[j]&&dp[i]<dp[j]+1)
            {
                dp[i]=dp[j]+1;
            }
        }
        max=(max<dp[i]?dp[i]:max);
    }
    return max;
}
int main()
{
    int n;
    scanf("%d",&n);
    int a[n];
    for(int i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    printf("%d",lnds(a,n));
}
```

Competitive Programming

1-Finding Duplicates-O(n^2) Time Complexity,O(1) Space Complexity

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 1 2 3 4	

Program:

```
#include<stdio.h>
void rep(int a[],int n)
{
    int cp[n-1];
    for(int i=0;i<n;i++)
    {
        cp[i]=0;
    }
    for(int i=0;i<n;i++)
    {
        cp[a[i]]++;
        if(cp[a[i]]>1)
        {
            printf("%d",a[i]);
            break;
        }
    }
}
int main()
{
    int n;
    scanf("%d",&n);
    int a[n];
    for(int i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }rep(a,n);
}
```

2-Finding Duplicates-O(n) Time Complexity,O(1) Space Complexity

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

Program:

```
#include<stdio.h>
void rep(int a[],int n)
{
    int cp[n+1];
    for(int i=0;i<n;i++)
    {
        cp[i]=0;
    }
    for(int i=0;i<n;i++)
    {
        cp[a[i]]++;
        if(cp[a[i]]>1)
        {
            printf("%d",a[i]);
            break;
        }
    }
}
int main()
{
    int n;
    scanf("%d",&n);
    int a[n];
    for(int i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }rep(a,n);
}
```

3-Print Intersection of 2 sorted arrays-O(m*n)Time Complexity,O(1) Space Complexity Quiz

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
```

Program:

```
#include<stdio.h>
int main()
{
    int t;
    scanf("%d",&t);
    while(t!=0)
    {
        int n;
        scanf("%d",&n);
        int a[n];
        for(int i=0;i<n;i++)
        {
            scanf("%d",&a[i]);
        }
        int m;
        scanf("%d",&m);
        int b[m];
        for(int i=0;i<m;i++)
        {
            scanf("%d",&b[i]);
        }
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<m;j++)
            {
                if(a[i]==b[j])
                {
                    printf("%d ",a[i]);
                    break;
                }
            }
        }
        t--;
    }
}
```

4-Print Intersection of 2 sorted arrays-O(m+n)Time Complexity,O(1) Space Complexity

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

Program:

```
#include<stdio.h>
int main()
{
    int t;
    scanf("%d",&t);
    while(t!=0)
    {
        int n;
        scanf("%d",&n);
        int a[n];
        for(int i=0;i<n;i++)
        {
            scanf("%d",&a[i]);
        }
        int m;
        scanf("%d",&m);
        int b[m];
        for(int i=0;i<m;i++)
        {
            scanf("%d",&b[i]);
        }
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<m;j++)
            {
                if(a[i]==b[j])
                {
                    printf("%d ",a[i]);
                    break;
                }
            }
        }
        t--;
    }
}
```

5-Pair with Difference-O(n^2)Time Complexity,O(1) Space Complexity Quiz

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

Program:

```
#include<stdio.h>
int main()
{
    int n;
    scanf("%d",&n);
    int a[n];
    for(int i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    int k;
    int flag=0;
    scanf("%d",&k);
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
        if((a[i]-a[j]==k)&&i!=j)
        {
            flag=1;
        }
       }
    }
    if(flag==1)
    printf("1");
    else
    printf("0");
}
```

6-Pair with Difference -O(n) Time Complexity,O(1) Space Complexity
Quiz

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

Program:

```
#include<stdio.h>
int main()
{
   int n;
   scanf("%d",&n);
   int a[n];
   for(int i=0;i<n;i++)
   {
      scanf("%d",&a[i]);
   }
   int k;
   int flag=0;
   scanf("%d",&k);
   for(int i=0;i<n;i++)
   {
      for(int j=0;j<n;j++)
      {
         if((a[i]-a[j]==k)&&i!=j)
         {
            flag=1;
         }
      }
   }
   if(flag==1)
   printf("1");
   else
```

```
    printf("0");
}
```