**QUESTION:**

The Monk is trying to explain to its users that even a single unit of time can be extremely important and to demonstrate this particular fact he gives them a challenging task.

There are **N** processes to be completed by you, the chosen one, since you're Monk's favourite student. All the processes have a unique number assigned to them from **1 to N**.

Now, you are given two things:

The **calling** order in which all the processes are called.

The **ideal** order in which all the processes should have been executed.

Now, let us demonstrate this by an example. Let's say that there are **3 processes**, the calling order of the processes is: **3 - 2 - 1.** The ideal order is: **1 - 3 - 2,** i.e., process number 3 will only be executed after process number 1 has been completed; process number 2 will only be executed after process number 3 has been executed.

***Iteration #1****:* Since the ideal order has process #1 to be executed firstly, the calling ordered is changed, i.e., the first element has to be pushed to the last place. Changing the position of the element takes 1 unit of time. The new calling order is: 2 - 1 - 3. Time taken in step #1: 1.

***Iteration #2****:* Since the ideal order has process #1 to be executed firstly, the calling ordered has to be changed again, i.e., the first element has to be pushed to the last place. The new calling order is: 1 - 3 - 2. Time taken in step #2: 1.

***Iteration #3****:* Since the first element of the calling order is same as the ideal order, that process will be executed. And it will be thus popped out. Time taken in step #3: 1.

***Iteration #4****:* Since the new first element of the calling order is same as the ideal order, that process will be executed. Time taken in step #4: 1.

***Iteration #5****:* Since the last element of the calling order is same as the ideal order, that process will be executed. Time taken in step #5: 1.

Total time taken: 5 units.

**PS:** Executing a process takes 1 unit of time. Changing the position takes 1 unit of time.

**Input format:**

The first line a number **N**, denoting the number of processes. The second line contains the calling order of the processes. The third line contains the ideal order of the processes.

**Output format:**

Print the total time taken for the entire queue of processes to be executed.

**Constraints:**

1<=**N**<=100

**SAMPLE INPUT**

3
3 2 1
1 3 2

**SAMPLE OUTPUT**

5

| EX NO:1.1(a) | PRINTING THE UNITS OF TIME |
|---|---|
| DATE: | |

**AIM:**

To print the total time taken for the entire queue of processes to be executed.

**PSEUDOCODE:**

```
//Program: To print the total time
//Input: sequence of numbers
//Output: total time taken
   while(val)
         p++
         if(que[front]==arr1[k])
                  temp=de()
                  k++
         else
                  temp=de()
                  en(temp)
        if(k==n)
                  val=0
```

**SOURCE CODE:**

```
#include<stdio.h>
#include<stdlib.h>
void en(int ele);
int de();
void display();
#define size 100
int que[size];
int rear=-1,front=-1;
void en(int ele)
{
   rear=(rear+1)%size;

   if (rear==front)
   {
      printf("stack overfull");
      if(rear==0)
      {
         rear=size-1;
      }
      else
      {
         rear=rear-1;
```

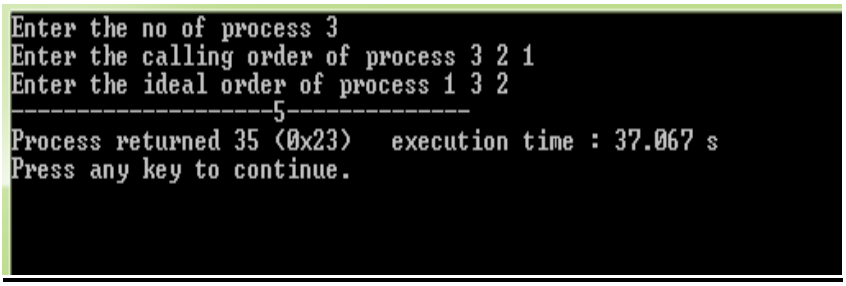
```
        }
    }
    else
    {
        if (front==-1)
        front=0;
     que[rear]=ele;
    }
}
int de()
{
    int item;
    if((rear==front)||(front==-1))
    {
        printf("empty");
    }
    else
    {
        item=que[front];
        front=(front+1)%size;
        return item;
    }
}
void main()
{
    int n,i,temp,val=1,k=0,p=1;
    printf("Enter the no of process ");
    scanf("%d",&n);
    int arr1[n],arr2[n];
    printf("Enter the calling order of process ");
    for(i=0;i<n;i++)
        scanf("%d",&arr1[i]);
    printf("Enter the ideal order of process ");
    for(i=0;i<n;i++)
    {
        scanf("%d",&arr2[i]);
        en(arr2[i]);
    }
    while(val)
    {
        p++;
        if(que[front]==arr1[k])
        {
            temp=de();
            k++;
        }
        else
```

```
        {
            temp=de();
            en(temp);
        }
        if(k==n)
            val=0;
    }

    printf("--------------------%d--------------",p);

}
```

**DATA STRUCTURE USED:** Queue

**TIME COMPLEXITY:** O(n)

**OUTPUT:**




**RESULT:**

Thus a program to print the total time taken for entire queue was successfully executed and verified.

**QUESTION:**

Sherlock and Watson are playing swapping game. Watson gives to Sherlock a string **S** on which he has performed **K** swaps. You need to help Sherlock in finding the original string.

One swap on a string is performed in this way:

 Assuming 1 indexing, the**i'th** letter from the end is inserted between **i'th** and **(i+1)'th** letter from the starting.

For example, we have "contest". After one swap, it would change to "ctosnet". Check this image:



**Input:**

First line contains **K**, the number of swaps performed by Watson. Next line contains **S**, the string Watson gives to Sherlock.

**Output:**

You have to print in one line the original string with which Watson had started.

**Constraints:**

**1 ≤ K ≤ 109**

**3 ≤ Length(S) ≤ 1000**

**All characters in S will be small English alphabets.**

**SAMPLE INPUT**

3

hrrkhceaate

**SAMPLE OUTPUT**

hackerearth

| EX NO:1.1(b) | SWAPPING OF STRING |
|---|---|
| DATE: | |

**AIM:**

To find the original string which performed K swaps.

**PSEUDOCODE:**

```
//Program: To find the original string
//Input: string which performed K swaps
//Output: Original string
   For i←o to sw
      va1←n-1
      va2←0
      For j←1 to n
         arr2[va1] ← arr1[j]
         va1--
      For  k←0 to n
         arr2[va2]←arr1[k]
         va2++
      strcpy(arr1,arr2)
```

**SOURCE CODE:**

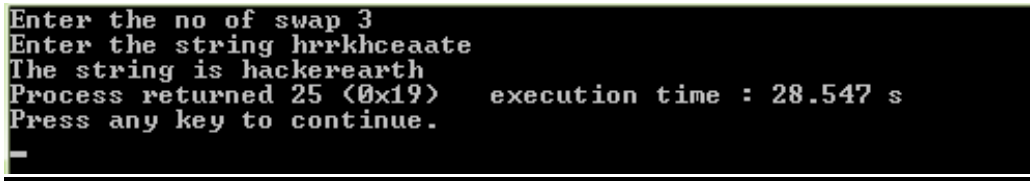
```
#include<stdio.h>
#include<string.h>
void main()
{
   int sw,i,j,k,va1,va2,n;
   char arr1[20],arr2[20];
   printf("Enter the no of swap ");
   scanf("%d",&sw);
   printf("Enter the string ");
   scanf("%s",arr1);
   n=strlen(arr1);
   for(i=0;i<sw;i++)
   {
      va1=n-1;
      va2=0;
      for(j=1;j<n;j=j+2)
      {
         arr2[va1]=arr1[j];
         va1--;
      }
      for(k=0;k<n;k=k+2)
      {
         arr2[va2]=arr1[k];
         va2++;
```

```
            }
            strcpy(arr1,arr2);
        }
        printf("The string is %s",arr1);
    }
```

**DATA STRUCTURE USED:** Array

**TIME COMPLEXITY:** $O(n^2)$

**OUTPUT:**




**RESULT:**

Thus a program to print find the original string which performed K swaps was successfully executed and verified.

**QUESTION:**

In a parallel universe, there are not just two charges like positive and negative, but there re 26 charges represented by lower english alphabets. Charges have a property of killing each other or in other words neutralizing each other if they are of same charge and next to each other. You are given a string s where each si represents a charge, where $0 \leq i \leq |s|-1$. You need to output size of final string followed by string after which no neutralizing is possible.

**SAMPLE INPUT**

12
aaacccbbcccd

**SAMPLE OUTPUT**

2
ad

**Explanation**

**aa**a**cc**c**bbcc**cd -> a**cc**d -> ad

| EX NO:1.2(a) | NEUTRALIZING THE STRING |
|---|---|
| DATE: | |

**AIM:**

To output size of final string followed by string after which no neutralizing is possible.

**PSEUDOCODE:**

```
//Program: To find the original string
//Input: string which performed K swaps
//Output: Original string
   change(arr[])
        j←0,k←1,y←1
        while(y)
                if(arr[j++]==arr[k++])
                        For i←k to strlen(arr)+2
                                arr[i-2]=arr[i]
                        y←0
```

**SOURCE CODE:**

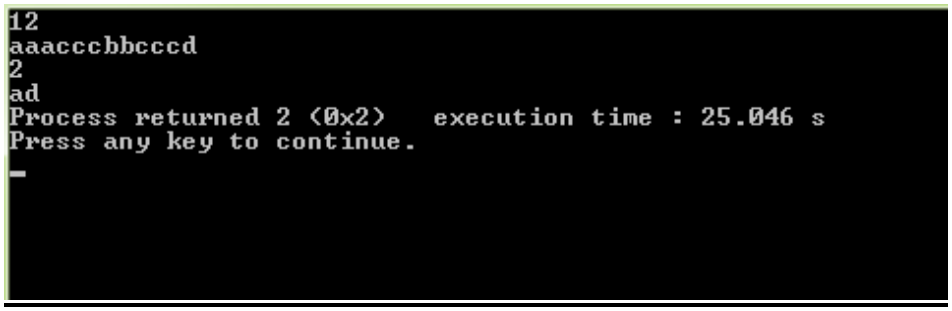
```
#include<stdio.h>
#include<string.h>
void change(char arr[])
{
        int i,j=0,k=1,y=1;
        while(y)
        {
                if(arr[j++]==arr[k++])
                {
                        for(i=k;i<strlen(arr)+2;i++)
                        {
                                arr[i-2]=arr[i];
                        }
                        y=0;
                }
        }


}
void main(){
        int n,i;
        scanf("%d",&n);
        char arr[n];
        scanf("%s",arr);
        for(i=0;i<n;i++)
                change(arr);
        printf("%d\n",strlen(arr));
```

```
        printf("%s",arr);
    }
```

**DATA STRUCTURE USED:** Array

**TIME COMPLEXITY:** $O(n^2)$

**OUTPUT:**




**RESULT:**

Thus a program to output size of final string followed by string after which no neutralizing is possible was successfully executed and verified.

**QUESTION:**

Numeros the Artist had two lists that were permutations of one another. He was very proud. Unfortunately, while transporting them from one exhibition to another, some numbers were lost out of the first list. Can you find the missing numbers?

As an example, the array with some numbers missing, *arr*=[7,2,5,3,5,3]. The original array of numbers *brr*=[7,2,5,4,6,3,5,3]. The numbers missing are [4,6].

**Notes:**

- If a number occurs multiple times in the lists, you must ensure that the frequency of that number in both lists is the same. If that is not the case, then it is also a missing number.
- You have to print all the missing numbers in ascending order.
- Print each missing number once, even if it is missing multiple times.
- The difference between the maximum and minimum number in the second list is less than or equal to 100.
- arr: the array with missing numbers
- brr: the original array of numbers

**Input:**

There will be four lines of input:

- *n*, the size of the first list, *arr*.
- The next line contains *n* space-separated integers *arr*[*i*].
- *m*, the size of the second list, *brr*.
- The next line contains *m* space-separated integers *brr*[*i*].

**Output:**

Output the missing numbers in ascending order separated by space.

**Constraints**

- $1 \leq n,m \leq 2X\ 105$
- $n \leq m$
- $1 \leq brr[i] \leq 104$
- $Xmax - Xmin \leq 100$

**Sample Input:**

```
10
203 204 205 206 207 208 203 204 205 206
13
203 204 204 205 206 207 205 208 203 206 205 206 204
```

**Sample Output:**

```
204 205 206
```

**EXPLANATION:**

204 is present in both arrays. Its frequency in *arr* is 2, while its frequency in *brr* is 3. Similarly, 205 and 206 occur twice in *arr*, but three times in *brr*. The rest of the numbers have the same frequencies in both lists.

| EX NO:1.2(b) | MISSING ELEMENTS |
|---|---|
| DATE: | |

**AIM:**

To print the missing numbers in ascending order separated by space.

**PSEUDOCODE:**

```
//Program: To find the original string
//Input: string which performed K swaps
//Output: Original string
 iSort(arr[],n)
      For  i ← 1 to  n
              key ← arr[i];
              j ← i - 1
              While (j >= 0 && arr[j] > key)
                    arr[j + 1] ← arr[j]
                    j ← j - 1
              arr[j + 1] ← key
```

**SOURCE CODE:**

```
#include<stdio.h>
void iSort(int arr[], int n)
{
   int i, key, j;
   for (i = 1; i < n; i++) {
      key = arr[i];
      j = i - 1;
      while (j >= 0 && arr[j] > key) {
         arr[j + 1] = arr[j];
         j = j - 1;
      }
      arr[j + 1] = key;
   }
}

void main(){
        int n1,n2,i,k=0,j;
        scanf("%d",&n1);
        scanf("%d",&n2);
        int arr1[n1],arr2[n2];
        for(i=0;i<n1;i++)
                scanf("%d",&arr1[i]);
        for(i=0;i<n2;i++)
                scanf("%d",&arr2[i]);
        iSort(arr1,n1);
        iSort(arr2,n2);
```

```
        for(i=0;i<n2;i++)
        {
                if(arr1[k]!=arr2[i])
                        printf("%d--\n",arr2[i]);
                else
                        k++;
        }

    }
```

**DATA STRUCTURE USED:** Array

**TIME COMPLEXITY:** $O(n^2)$

**OUTPUT:**

```
10
13
203 204 205 206 207 208 203 204 205 206
203 204 204 205 206 207 205 208 203 206 205 206 204
204--
205--
206--

Process returned 13 (0xD)   execution time : 65.116 s
Press any key to continue.
```

**RESULT:**

Thus a program to output the missing numbers in ascending order separated by space was successfully executed and verified.

**QUESTION:**

Implement recursive bubble sort. Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

**Example:**

**First Pass:**

( **5 1** 4 2 8 ) → ( **1 5** 4 2 8 ), Here, algorithm compares the first two elements, and swaps since 5 > 1.

( 1 **5 4** 2 8 )→ ( 1 **4 5** 2 8 ), Swap since 5 > 4

( 1 4 **5 2** 8 ) → ( 1 4 **2 5** 8 ), Swap since 5 > 2

( 1 4 2 **5 8** ) → ( 1 4 2 **5 8** ), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

**Second Pass:**

( **1 4** 2 5 8 ) → ( **1 4** 2 5 8 )

( 1 **4 2** 5 8 ) → ( 1 **2 4** 5 8 ), Swap since 4 > 2

( 1 2 **4 5** 8 ) → ( 1 2 **4 5** 8 )

( 1 2 4 **5 8** ) → ( 1 2 4 **5 8** )

Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one **whole** pass without **any** swap to know it is sorted.

**Third Pass:**

( **1 2** 4 5 8 ) → ( **1 2** 4 5 8 )

( 1 **2 4** 5 8 ) → ( 1 **2 4** 5 8 )

( 1 2 **4 5** 8 ) → ( 1 2 **4 5** 8 )

( 1 2 4 **5 8** ) → ( 1 2 4 **5 8** )

| EX NO:1.3(a) | RECURSIVE BUBBLE SORT |
|---|---|
| DATE: | |

**AIM:**

To write the algorithm for recursive bubble sort.

**PSEUDOCODE:**

```
//Program: To sort the array
//Input:  Array
//Output: sorted array
bubbleSort( arr[],n)
   if (n > 1)
            For i←0 to i<n-1
               if (arr[i] > arr[i+1])
                  temp=arr[i]
                  arr[i]=arr[i+1]
                  arr[i+1]=temp
   bubbleSort(arr, n-1);
```

**SOURCE CODE:**

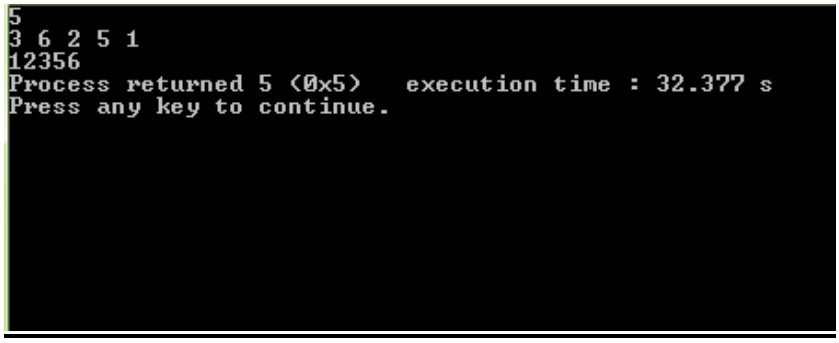
```
#include<stdio.h>
void bubbleSort(int arr[], int n)
{
   int i,temp;
   if (n > 1)
   {
            for(i=0; i<n-1; i++)
               if (arr[i] > arr[i+1])
               {
                  temp=arr[i];
                                 arr[i]=arr[i+1];
                                 arr[i+1]=temp;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

               bubbleSort(arr, n-1);
         }
}
void main()
{
         int n,i;
         scanf("%d",&n);
         int arr[n];
         for(i=0;i<n;i++)
                  scanf("%d",&arr[i]);
         bubbleSort(arr,n);
         for(i=0;i<n;i++)
```

```
            printf("%d",arr[i]);

    }
```

**DATA STRUCTURE USED:** Array

**TIME COMPLEXITY:** $O(n^2)$

**OUTPUT:**




**RESULT:**

Thus a program to sort the given array was successfully executed and verified.

**QUESTION:**
Given a number, we need to find sum of its digits using recursion.
Examples:
Input : 12345
Output : 15
Input : 45632
Output :20

| EX NO:1.3(b) | SUM OF DIGITS |
|---|---|
| DATE: | |

**AIM:**

To find the sum of digits using recursion for the given number.

**PSEUDOCODE:**

```
//Program: print the sum of digits
//Input:  Integer
//Output: Sum of digits
sumarr(n)
   if(n>0)
      sum←0
      sum←(n%10)+sumarr(n/10)
      return sum
   else
      return 0
```

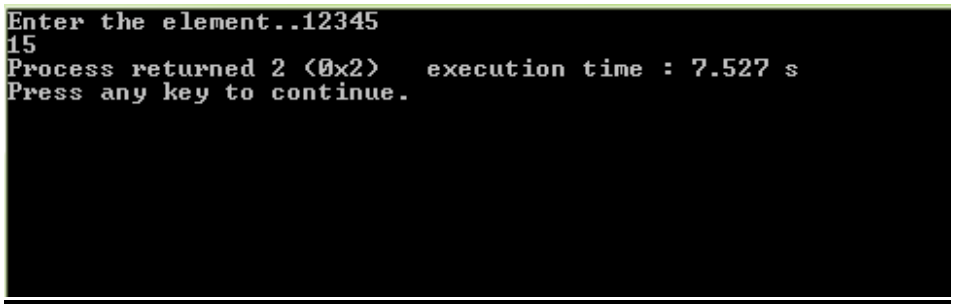
**SOURCE CODE:**

```
#include<stdio.h>
int sumarr(int n)
{
   if(n>0)
   {
      int sum=0;
      sum=(n%10)+sumarr(n/10);
      return sum;
   }
   else
      return 0;

}
void main()
{
   int i,n;
   printf("Enter the element..");
   scanf("%d",&n);
   printf("%d",sumarr(n));
}
```

**DATA STRUCTURE USED:** None

**TIME COMPLEXITY:** O(n)

**OUTPUT:**




**RESULT:**

Thus a program to print the sum of digits was successfully executed and verified.

**QUESTION:**

Given an array of integers. Find a peak element in it. An array element is peak if it is NOT smaller than its neighbors. For corner elements, we need to consider only one neighbor. For example, for input array {5, 10, 20, 15}, 20 is the only peak element. For input array {10, 20, 15, 2, 23, 90, 67}, there are two peak elements: 20 and 90. Note that we need to return any one peak element.

Following corner cases give better idea about the problem.

1) If input array is sorted in strictly increasing order, the last element is always a peak element. For example, 50 is peak element in {10, 20, 30, 40, 50}.

2) If input array is sorted in strictly decreasing order, the first element is always a peak element. 100 is the peak element in {100, 80, 60, 50, 20}.

3) If all elements of input array are same, every element is a peak element

| EX NO:2.1(a) | PEAK ELEMENT |
|---|---|
| DATE: | |

**AIM:**

To find the peak element in the given array of integers.

**PSEUDOCODE:**

```
//Program: to find the peak element
//Input:  Integer array
//Output: Peak element
peak(l,h,arr[])
        mid←(l+h)/2
        l1←mid-1,h1←mid+1
        if(l<=h)
                if(arr[mid]>=arr[l1]&&arr[mid]>=arr[h1])
                        return arr[mid]
                else if(arr[mid]<arr[l1])
                        return peak(l,l1,arr)
                else
                        return peak(h1,h,arr)
        return -1
```

**SOURCE CODE:**

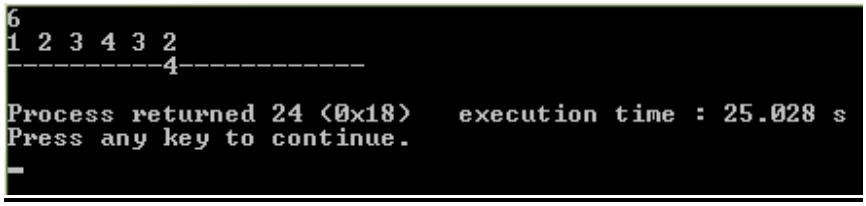
```
#include<stdio.h>
int peak(int l,int h,int arr[])
{
        int mid=(l+h)/2;
        int l1=mid-1,h1=mid+1;
        if(l<=h)
        {
                if(arr[mid]>=arr[l1]&&arr[mid]>=arr[h1])
                        return arr[mid];
                else if(arr[mid]<arr[l1])
                        return peak(l,l1,arr);
                else
                        return peak(h1,h,arr);
        }
        return -1;

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

```
                scanf("%d",&arr[i]);
            printf("----------%d----------\n",peak(0,n-1,arr));
        }
```

**DATA STRUCTURE USED:** Array

**TIME COMPLEXITY:** O(log n)

**OUTPUT:**




**RESULT:**

Thus a program to find the peak element in the given array of integers was successfully executed and verified.

**QUESTION:**

Let the sorted array be A[] = {2, 3, 5, 6, 8, 9, 12, 13, 14} with indices from 0 to 8. You are required to find the position of x = 13 in this array. Divide the sorted array into the following 3 parts by evaluating the values of mid1 and mid2:

- {2,3,5}
- {6,8,9}
- {12,13,14}

Run the ternary search algorithm to find x.

| EX NO:2.1(b) | TERNARY SEARCH |
|---|---|
| DATE: | |

**AIM:**

To implement ternary search to find the element in the sorted array.

**PSEUDOCODE:**

```
//Program: to find the element
//Input:  Integer array
//Output:index the key  element
ternarysearch(l,r,key, arr[])
        if(r>=l)
                mid1←l+((r-l)/3)
                mid2←r-((r-l)/3)
                if(arr[mid1]==key)
                        return mid1
                if(arr[mid2]==key)
                        return mid2
                if(key<arr[mid1])
                        return ternarysearch(l,mid1-1,key,arr)
                else if(key>arr[mid2])
                        return ternarysearch(mid2+1,r,key,arr)
                else
                        return ternarysearch(mid1+1,mid2-1,key,arr)
        return -1
```

**SOURCE CODE:**

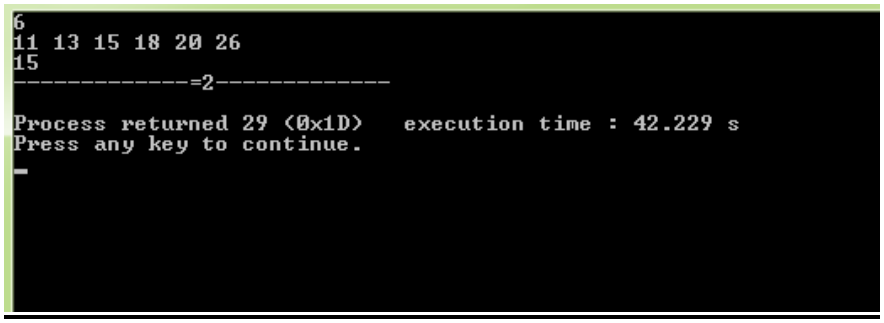
```
#include<stdio.h>
int ternarysearch(int l,int r, int key, int arr[])
{
        if(r>=l)
        {
                int mid1=l+((r-l)/3);
                int mid2=r-((r-l)/3);
                if(arr[mid1]==key)
                        return mid1;
                if(arr[mid2]==key)
                        return mid2;
                if(key<arr[mid1])
                        return ternarysearch(l,mid1-1,key,arr);
                else if(key>arr[mid2])
                        return ternarysearch(mid2+1,r,key,arr);
                else
                        return ternarysearch(mid1+1,mid2-1,key,arr);
        }
        return -1;
```

```
}
void main()
{
        int n,i,key;
        scanf("%d",&n);
        int arr[n];
        for(i=0;i<n;i++)
                scanf("%d",&arr[i]);
        scanf("%d",&key);
        printf("-------------=%d-------------\n",ternarysearch(0,n-1,key,arr));
}
```

**DATA STRUCTURE USED:** Array

**TIME COMPLEXITY:** O(log n)

**OUTPUT:**




**RESULT:**

Thus a program to implement ternary search was successfully executed and verified.

**QUESTION:**

Given two sorted arrays. There is only 1 difference between the arrays. First array has one element extra added in between. Find the index of the extra element.

Examples :

Input : {2, 4, 6, 8, 9, 10, 12};
{2, 4, 6, 8, 10, 12};
Output : 4
The first array has an extra element 9.
The extra element is present at index 4.
Input : {3, 5, 7, 9, 11, 13}
{3, 5, 7, 11, 13}
Output : 3

| EX NO:2.2(a) | EXTRA ELEMENT |
|---|---|
| DATE: | |

**AIM:**

To find the extra element in the sorted array.

**PSEUDOCODE:**

```
//Program: to find the extra element
//Input: Integer array
//Output:index of the extra element
unknown( l,h,arr1[],arr2[],k)
        mid←(l+h)/2
        if(mid<=k)
                if(arr1[mid]==arr2[mid])
                        return unknown(mid+1,h,arr1,arr2,k)
                else if(arr1[mid]==arr2[mid-1])
                        return mid-1
                else
                        return unknown(l,mid,arr1,arr2,k)
        return mid
```

**SOURCE CODE:**

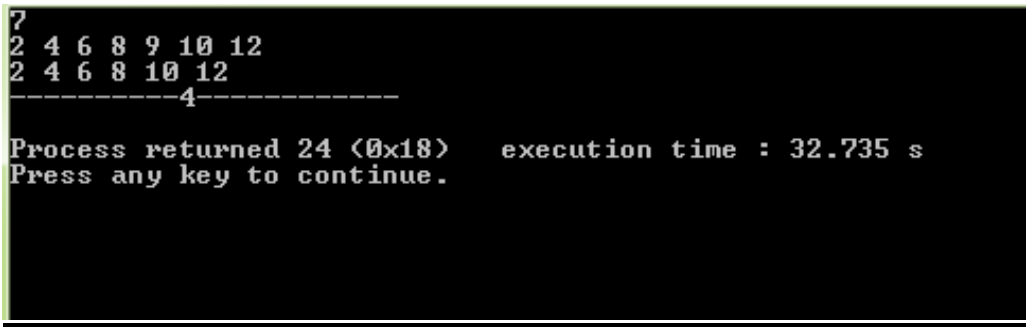
```
#include<stdio.h>
int unknown(int l,int h,int arr1[],int arr2[],int k)
{
        int mid=(l+h)/2;
        if(mid<=k)
        {
                if(arr1[mid]==arr2[mid])
                        return unknown(mid+1,h,arr1,arr2,k);
                else if(arr1[mid]==arr2[mid-1])
                        return mid-1;
                else
                        return unknown(l,mid,arr1,arr2,k);
        }
        return mid;
}
void main()
{
        int n1,i;
        scanf("%d",&n1);
        int arr1[n1],arr2[n1-1];
        for(i=0;i<n1;i++)
                scanf("%d",&arr1[i]);
        for(i=0;i<n1-1;i++)
                scanf("%d",&arr2[i]);
```

```
        printf("----------%d----------\n",unknown(0,n1-1,arr1,arr2,n1-2));
    }
```

**DATA STRUCTURE USED:** Array

**TIME COMPLEXITY:** O(log n)

**OUTPUT:**




**RESULT:**

Thus a program to find the extra element was successfully executed and verified.

**QUESTION:**

There are 2 sorted arrays A and B of size n each. Write an algorithm and implement it to find the median of the array obtained after merging the above 2 arrays (i.e. array of length 2n). The complexity should be O(log(n))

| EX NO:2.2(b) | MEDIAN FINDING |
|---|---|
| DATE: | |

**AIM:**

To find the median of the two sorted arrays.

**PSEUDOCODE:**

```
//Program: to find the  median
//Input:  Integer arrays
//Output: Median
   median( arr1[],arr2[],n)
      if(n==1)
            return (arr1[0]+arr2[0])/2;
      else if(n==2)
            return (max(arr1[0],arr2[0])+min(arr1[1],arr2[1]))/2
      else
            m1←getmedian(arr1,n)
            m2←getmedian(arr2,n)
            if(m1==m2)
                  return m1
            else if(m1>m2)
                  if(n%2==0)
                        return median(arr1,arr2+(n/2-1),n-(n/2)+1)
                  return median(arr1,arr2+(n/2),n-(n/2))
            else
                  if(n%2==0)
                        return median(arr1+(n/2)-1,arr2,n-(n/2)+1)
                  return median(arr1+(n/2),arr2,n-(n/2))
```

**SOURCE CODE:**

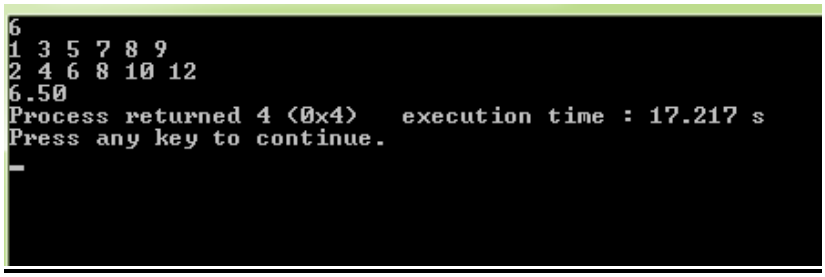
```
#include<stdio.h>
float max(float a,float b)
{
      return (a>b)?a:b;
}
float min(float a,float b)
{
      return (a<b)?a:b;
}
float getmedian(int a[],int n)
{
      if(n%2==0)
            {
                  return (float)(a[n/2]+a[n/2-1])/2;
            }
      else
```

```
                return (float)a[n/2];

        }
        float median(int arr1[],int arr2[],int n)
        {
                if(n==1)
                        return (float)(arr1[0]+arr2[0])/2;
                else if(n==2)
                        {
                                return (float)(max(arr1[0],arr2[0])+min(arr1[1],arr2[1]))/2;
                        }
                else
                {
                        float m1=getmedian(arr1,n);
                        float m2=getmedian(arr2,n);
                        if(m1==m2)
                                return (float)m1;
                        else if(m1>m2)
                        {
                                if(n%2==0)
                                        return (float)median(arr1,arr2+(n/2-1),n-(n/2)+1);
                                return (float)median(arr1,arr2+(n/2),n-(n/2));
                        }
                        else
                        {
                                if(n%2==0)
                                        return (float)median(arr1+(n/2)-1,arr2,n-(n/2)+1);
                                return (float)median(arr1+(n/2),arr2,n-(n/2));
                        }
                }

        }
        void main()
        {
                int n,i;
                scanf("%d",&n);
                int arr1[n],arr2[n];
                for(i=0;i<n;i++)
                        scanf("%d",&arr1[i]);
                for(i=0;i<n;i++)
                        scanf("%d",&arr2[i]);
                printf("%.2f",median(arr1,arr2,n));
        }
```

**DATA STRUCTURE USED:** Array

**TIME COMPLEXITY:** O(log n)

**OUTPUT:**




**RESULT:**

Thus a program to find the median in sorted arrays was successfully executed and verified.

**QUESTION:**

Ishaan wants to intern at GeeksForGeeks but for that he has to go through a test. There are N candidates applying for the internship including Ishaan and only one is to be selected. Since he wants to qualify he asks you to help him. The test is as follows.

The candidates are asked to stand in a line at positions 1 to N and given a number K. Now, every Kth candidate remains and the rest are eliminated. This is repeated until the number of candidates is less than K. Out of the remaining candidates; the one standing at smallest position is selected. You need to tell Ishaan at position he must stand to get selected.

**Input:** First line of input contains a single integer T denoting the number of test cases. The only line of each test case contains 2 space-separated integers N denoting number of candidates and K.

**Output:** For each test case, print the required position in a new line.

**Example :**

**Input :**

3
30 3
18 3
5 2

**Output :**

27
94

**Explanation :**

Case 1 :
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
3 6 9 12 15 18 21 24 27 30
9 18 27
27
Case 2 :
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
3 6 9 12 15 18
9 18 (less than K)
9
Case 3 :
1 2 3 4 5
2 4
4

| EX NO:2.3(a) | BETTER POSITION |
|---|---|
| DATE: | |

**AIM:**

To find the best position for the given criteria.

**PSEUDOCODE:**

```
//Program: to find the  position number
//Input:  Integer arrays
//Output: best position in the given numbers
   While(k<=n)
                temp←k
                K←k*v
```

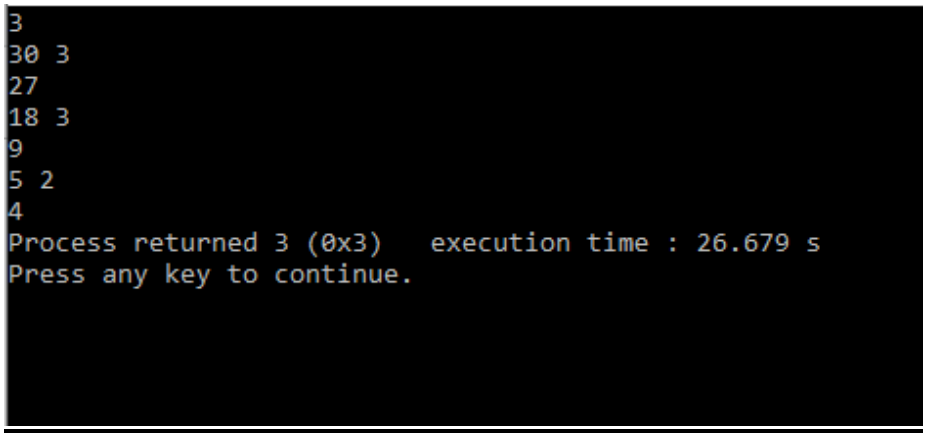
**SOURCE CODE:**

```
#include<stdio.h>
void main()
{
        int t,y,n,i;
        scanf("%d",&t);
        for(y=0;y<t;y++)
        {
                scanf("%d",&n);
                int k,temp;
                scanf("%d",&k);
                int v=k;
                while(k<=n)
                {
                        temp=k;
                        k=k*v;
                }
                printf("%d",temp);
        }
}
```

**DATA STRUCTURE USED:** Nil

**TIME COMPLEXITY:** O(n)

**OUTPUT:**




**RESULT:**

Thus a program to find the best position was successfully executed and verified.

**QUESTION:**

Lena is preparing for an important coding competition that is preceded by a number of sequential preliminary contests. Initially, her luck balance is 0. She believes in "saving luck", and wants to check her theory. Each contest is described by two integers, L[i] and T[i]:

- L[i] is the amount of luck associated with a contest. If Lena *wins* the contest, her luck balance will *decrease* by L[i]; if she *loses* it, her luck balance will *increase* by L[i].
- T[i] denotes the contest's *importance rating*. It's equal to 1 if the contest is *important*, and it's equal to 0 if it's *unimportant*.

If Lena loses no more than k *important* contests, what is the maximum amount of luck she can have after competing in all the preliminary contests? This value *may* be negative.

For example, k=2 and:

Contest L[i] T[i]
1 5 1
2 1 1
3 4 0

If Lena loses all of the contests, her will be 5+1+4 = 10. Since she is allowed to lose 2 important contests, and there are only 2 important contests. She can lose all three contests to maximize her luck at 10. If k=1, she has to win at least 1 of the 2 important contests. She would choose to win the lowest value important contest worth 1. Her final luck will be 5+4-1 = 8.

**Function Description**

Complete the *luckBalance* function in the editor below. It should return an integer that represents the maximum luck balance achievable.

luckBalance has the following parameter(s):

- *k*: the number of important contests Lena can lose
- *contests*: a 2D array of integers where each contests[i] contains two integers that represent the luck balance and importance of the ith contest.

**Input Format**

The first line contains two space-separated integers n and k, the number of preliminary contests and the maximum number of important contests Lena can lose.

Each of the next n lines contains two space-separated integers, L[i] and T[i], the contest's luck balance and its importance rating.

**Constraints**

- 1<=n<=100
- 0<=k<=N
- 1<=L[i]<=104
- T[i] in {0,1}

**Output Format**

Print a single integer denoting the maximum amount of luck Lena can have after all the contests.

**Sample Input**

6 3
5 1
2 1
1 1
8 1
10 0

| EX NO:2.3(b) | MAXIMUM LUCK |
|---|---|
| DATE: | |

**AIM:**

To find the maximum amount of luck Lena can have.

**PSEUDOCODE:**

```
//Program: to find maximum amount of luck
//Input:  Integer arrays
//Output: maximum amount of luck
    For i←0 to i<k1
        if(i<k)
                sum+=a[i]
        else
                sum-=a[i]
   For i←0 to i<k2
        sum+=b[i]
```

**SOURCE CODE:**

```
#include<stdio.h>
void selectionsort(int arr[], int n)
{
   int i, j, min_idx,temp;
   for (i = 0; i < n-1; i++)
   {
      min_idx = i;
      for (j = i+1; j < n; j++)
      if (arr[j] > arr[min_idx])
         min_idx = j;
      temp=arr[min_idx];
      arr[min_idx]=arr[i];
      arr[i]=temp;
   }
}
main()
{
        int n,i,k,t1,t2,k1=0,k2=0,sum=0;
        scanf("%d%d",&n,&k);
        int a[n],b[n];
        for(i=0;i<n;i++)
        {
                scanf("%d%d",&t1,&t2);
                if(t2==1)
                        a[k1++]=t1;
                else
                        b[k2++]=t1
```

5 0

**Sample Output**

29

**Explanation**

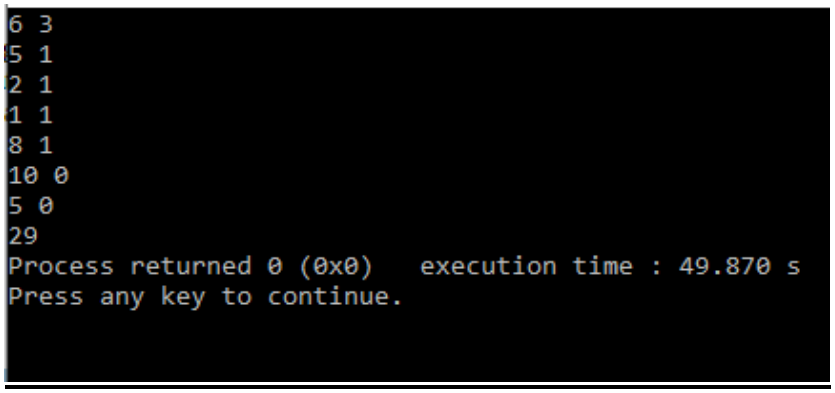
There are n=6 contests. Of these contests, 4 are important and she cannot lose more than k=3 of them. Lena maximizes her luck if she wins the 3rd important contest (where L[i] = 1) and loses all of the other five contests for a total luck balance of 5+2+8+10+5-1 = 29.

```
        }
        selectionsort(a,k1);
        for(i=0;i<k1;i++)
        {
                if(i<k)
                        sum+=a[i];
                else
                        sum-=a[i];
        }
        for(i=0;i<k2;i++)
                sum+=b[i];
        printf("%d",sum);
}
```

**DATA STRUCTURE USED:** Array

**TIME COMPLEXITY:** $O(n^2)$

**OUTPUT:**




**RESULT:**

Thus a program to find maximum amount of luck Lena can have was successfully executed and verified.

**QUESTION:**

Mark and Jane are very happy after having their first child. Their son loves toys, so Mark wants to buy some. There are a number of different toys lying in front of him, tagged with their prices. Mark has only a certain amount to spend, and he wants to maximize the number of toys he buys with this money.

Given a list of prices and an amount to spend, what is the maximum number of toys Mark can buy? For example, if prices = [1,2,3,4] and Mark has k=7 to spend, he can buy items [1,2,3] for 6, or [3,4] for 7 units of currency. He would choose the first group of 3 items.

**Function Description**

Complete the function *maximumToys* in the editor below. It should return an integer representing the maximum number of toys Mark can purchase.

maximumToys has the following parameter(s):

- *prices*: an array of integers representing toy prices
- *k*: an integer, Mark's budget

**Input Format**

The first line contains two integers, n and k, the number of priced toys and the amount Mark has to spend.

The next line contains n space-separated integers prices[i]

**Constraints**

1<=n<=105

1<=k<=109

1<=prices[i]<=109

A toy can't be bought multiple times.

**Output Format**

An integer that denotes the maximum number of toys Mark can buy for his son.

**Sample Input**

7 50

1 12 5 111 200 1000 10

**Sample Output**

4

**Explanation**

He can buy only 4 toys at most. These toys have the following prices: 1, 12, 5, 10.

| EX NO:2.4(a) | ATMOST TOYS |
|---|---|
| DATE: | |

**AIM:**

To find the maximum number of toys can buy.

**PSEUDOCODE:**

```
//Program: to find the  maximum count
//Input:  Integer arrays
//Output: maximum count
While(s<=k)
        s+=a[i1++]
        c++
```

**SOURCE CODE:**

```
#include<stdio.h>
void selectionsort(int arr[], int n)
{
   int i, j, min_idx,temp;
   for (i = 0; i < n-1; i++)
   {
      min_idx = i;
      for (j = i+1; j < n; j++)
      if (arr[j] < arr[min_idx])
         min_idx = j;
      temp=arr[min_idx];
      arr[min_idx]=arr[i];
      arr[i]=temp;
   }
}
main()
{
        int n,k,c=0,s=0,i,i1=0;
        scanf("%d%d",&n,&k);
        int a[n];
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);
        selectionsort(a,n);
        while(s<=k)
        {
                s+=a[i1++];
                c++;
        }
        printf("%d",c-1);
}
```

**DATA STRUCTURE USED:** Array

**TIME COMPLEXITY:** $O(n^2)$

**OUTPUT:**

```
7 50
1 12 5 111 200 1000 10
4
Process returned 0 (0x0)   execution time : 19.664 s
Press any key to continue.
```

**RESULT:**

Thus a program to find the maximum number of toys can buy was successfully executed and verified.

**QUESTION:**

You are given n activities with their start and finish times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time.

Example: Consider the following 6 activities. start[] = {1, 3, 0, 5, 8, 5}; finish[] = {2, 4, 6, 7, 9, 9};The maximum set of activities executed by a single person is {0, 1, 3, 4}.

| EX NO:2.4(b) | MAXIMUM NO OF ACTIVITIES |
|---|---|
| DATE: | |

**AIM:**

To find the maximum number of activities.

**PSEUDOCODE:**

```
//Program: to find the  maximum activities
//Input:  Integer arrays
//Output: index of activities
i←0
For  j← 1 to j < n
        if (s[j] >= f[i])
                display  j
                i = j
```

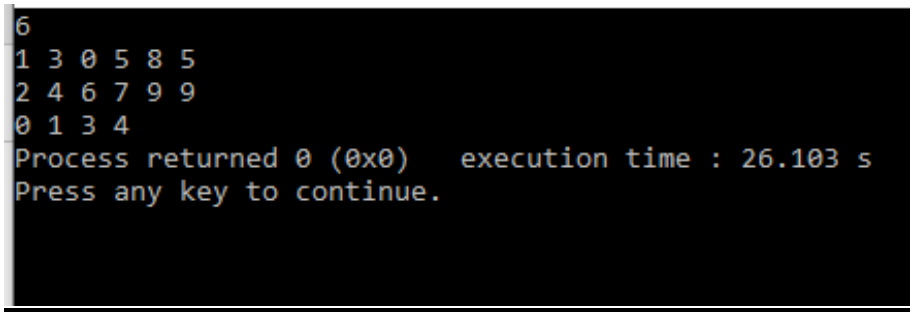
**SOURCE CODE:**

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

**DATA STRUCTURE USED:** Array

**TIME COMPLEXITY:** O(n)

**OUTPUT:**




**RESULT:**

Thus a program to find the maximum activities was successfully executed and verified.

**QUESTION:**

A Tiny Robot RX-001 is exploring ancient ruins. He found a piece of paper with a word written on it. Fortunately, people who used to live at this location several thousand years ago used only two letters of modern English alphabet: 'a' and 'b'. It's also known, that no ancient word contains two letters 'a' in a row. RX-001 has already recognized some of the word letters, the others are still unknown. RX-001 wants to look up all valid words that could be written on this paper in an ancient dictionary. He needs your help. Find him the word, which is the first in alphabetical order and could be written on the paper.

**Input format:** The first line contains non-empty string *s* consisting of 'a', 'b' and '?' characters. Character '?' corresponds to unrecognized letter. It's guaranteed, that there exists at least one ancient word, which could be written on the paper.

**Constraints:** Length of *s* is at most 50.

**Output format:** Output the first in alphabetical order word, which could be written on the paper, found by RX-001.

**SAMPLE INPUT1 :** ?ba??b

**SAMPLE OUTPUT:** ababab

**SAMPLE INPUT 2:** ?????????a????ab??

**SAMPLE OUTPUT 2:** abababababbababbabab

**SAMPLE INPUT**

?ba??b

**SAMPLE OUTPUT**

ababab

**Explanation**

**Explanation1:** Words ababab, ababbb, bbabab and bbabbb could be written on paper. The first in alphabetical order is ababab.

| EX NO:2.5(A) | ALPHABETICAL ORDER |
|---|---|
| DATE: | |

**AIM:**

To print the first alphabetical order word

**PSEUDOCODE:**

```
//Program: to print the first alphabetical order of the word
//Input:  string
//Output: string
For  i← 1 to  str[i]
      if (str[i] == '?')
            if (str[i-1] != 'a' && str[i+1] != 'a')
                  str[i] ← 'a'
           else
                  str[i]← 'b';
```

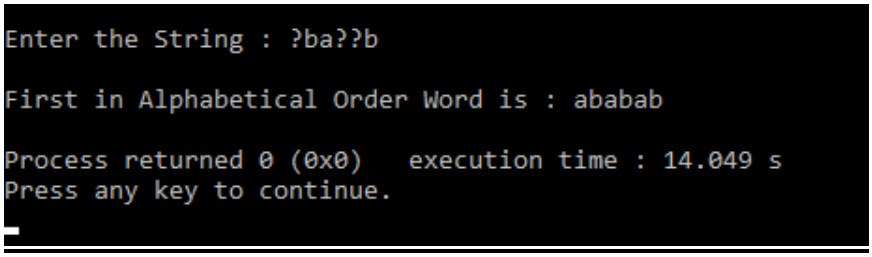
**SOURCE CODE:**

```
#include<stdio.h>
char str[100];
int main()
{
        int i;
   printf("\nEnter the String : ");
   scanf("%s", &str[1]);
   for (i = 1; str[i]; ++i)
        {
      if (str[i] == '?')
                {
            if (str[i-1] != 'a' && str[i+1] != 'a')
                  str[i] = 'a';
                        else
                        str[i] = 'b';
        }
   }
   printf("\nFirst in Alphabetical Order Word is : %s\n", &str[1]);
   return 0;
}
```

**DATA STRUCTURE USED:** Array

**TIME COMPLEXITY:** $O(n^2)$

**OUTPUT:**




**RESULT:**

Thus a program to print the first alphabetical order was successfully executed and verified.

**QUESTION:**

Tomya is a girl. She loves Chef Ciel very much. Tomya like a positive integer **p**, and now she wants to get a receipt of Ciel's restaurant whose total price is exactly **p**. The current menus of Ciel's restaurant are shown the following table.

**Name of Menu price**

| Name of Menu | price |
|---|---|
| eel flavored water | 1 |
| deep-fried eel bones | 2 |
| clear soup made with eel livers | 4 |
| grilled eel livers served with grated radish | 8 |
| savory egg custard with eel | 16 |
| eel fried rice (S) | 32 |
| eel fried rice (L) | 64 |
| grilled eel wrapped in cooked egg | 128 |
| eel curry rice | 256 |
| grilled eel over rice | 512 |
| deluxe grilled eel over rice | 1024 |
| eel full-course | 2048 |

Note that the **i**-th menu has the price $2_{i-1}$ ($1 \leq \mathbf{i} \leq 12$). Since Tomya is a pretty girl, she cannot eat a lot. So please find the minimum number of menus whose total price is exactly **p**. Note that if she orders the same menu twice, then it is considered as two menus are ordered. (See **Explanations** for details)

**Input**

The first line contains an integer **T**, the number of test cases. Then **T** test cases follow. Each test case contains an integer **p**.

**Output**

For each test case, print the minimum number of menus whose total price is exactly **p**.

**Constraints**

$1 \leq \mathbf{T} \leq 5$

$1 \leq \mathbf{p} \leq 100000$ ($10_5$)

There exists combinations of menus whose total price is exactly **p**.

**SAMPLE INPUT**

2
10
256

**SAMPLE OUTPUT**

21

**Explanation**

In the first sample, examples of the menus whose total price is 10 are the following:
1+1+1+1+1+1+1+1+1+1 = 10 (10 menus)
1+1+1+1+1+1+1+1+2 = 10 (9 menus)
2+2+2+2+2 = 10 (5 menus)
2+4+4 = 10 (3 menus)
2+8 = 10 (2 menus)
Here the minimum number of menus is 2.

| EX NO:2.5(B) | MINIMUM NUMBER OF MENUS |
|---|---|
| DATE: | |

**AIM:**

To print the minimum number of menus

**PSEUDOCODE:**

```
//Program: to print the minimum no of menus
//Input:  total price
//Output: no of items
For  i←n-1 to  i >= 0
        ans += P / largest;
        P %= largest;
        largest >>= 1;
```

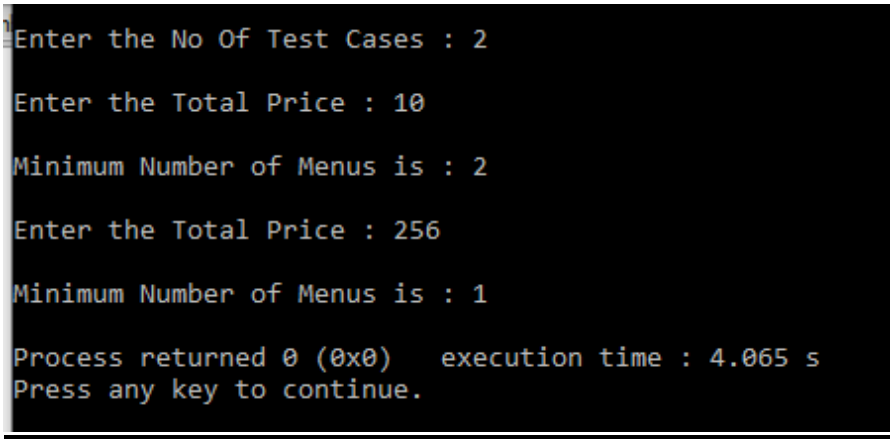
**SOURCE CODE:**

```
#include <stdio.h>
int main()
{
        int n=12,T;
   printf("\nEnter the No Of Test Cases : ");
        scanf("%d",&T);
        while(T--)
        {
                int P;
      printf("\nEnter the Total Price : ");
                scanf("%d",&P);
                int largest = 2048,ans = 0;
                for (int i =n-1; i >= 0 ; i--)
                {
                        ans += P / largest;
                        P %= largest;
                        largest >>= 1;
                }
                printf("\nMinimum Number of Menus is : %d\n",ans);
        }
        return 0;
}
```

**DATA STRUCTURE USED:** NIL

**TIME COMPLEXITY:** O(n)

**OUTPUT:**




**RESULT:**

Thus a program to print the minimum number of items was successfully executed and verified.

**QUESTION:**

Find out the maximum sub-array of non negative numbers from an array.
The sub-array should be continuous. That is, a sub-array created by choosing the second and fourth element and skipping the third element is invalid.
Maximum sub-array is defined in terms of the sum of the elements in the sub-array. Sub-array A is greater than sub-array B if sum(A) > sum(B).
Example:
A : [1, 2, 5, -7, 2, 3]
The two sub-arrays are [1, 2, 5] [2, 3].
The answer is [1, 2, 5] as its sum is larger than [2, 3]
**NOTE 1**: If there is a tie, then compare with segment's length and return segment which has maximum length.
**NOTE 2**: If there is still a tie, then return the segment with minimum starting index

**Input:**

The first line contains an integer T, depicting total number of test cases.
Then following T lines contains an integer N depicting the size of array and next line followed by the value of array.

**Output:**

Print the Sub-array with maximum sum.

**Constraints:**

$1 \leq T \leq 40$
$1 \leq N \leq 100$
$-100 \leq A[i] \leq 100$

**Example:**

**Input**

23
1 2 3
2
-1 2

**Output**

1 2 3
2

| EX NO:3.1(A) | SUB ARRAY HAVING MAXIMUM SUM |
|---|---|
| DATE: | |

**AIM:**

To find the subarray having maximum sum.

**PSEUDOCODE:**

```
//Program: to find the subarray having maximum sum
//Input: Integer arrays
//Output: sub array
For i=0 to i<n
      max_ending_here ← max_ending_here +A[i]
      If(max_so_far < max_ending_here)
         max_so_far ← max_ending_here
         sum_arr[j++]←A[i]
      if(max_ending_here < 0)
         max_ending_here = 0
         j=0
```

**SOURCE CODE:**

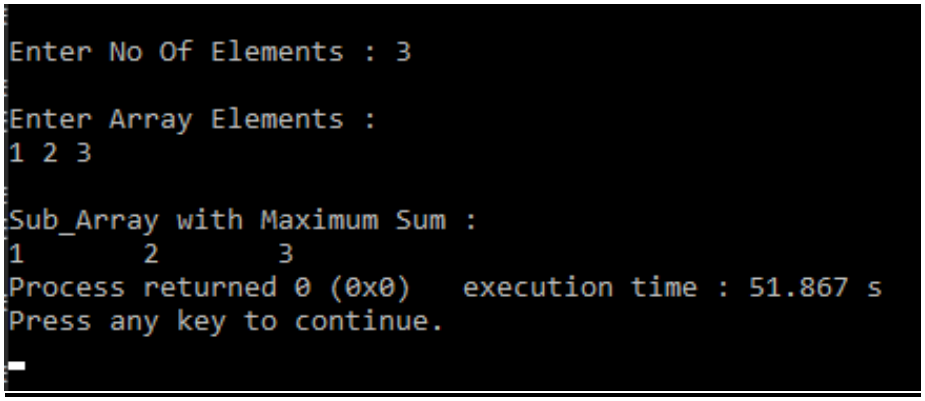
```
#include<stdio.h>
int max_subarray(int A[],int n)
{
   int max_ending_here = 0;
   int max_so_far = 0;
   int i,j=0,sum_arr[n];
   for(i=0;i<n;i++)
   {
      max_ending_here = max_ending_here +A[i];
      if(max_so_far < max_ending_here)
      {
         max_so_far = max_ending_here;
         sum_arr[j++]=A[i];
      }
      if(max_ending_here < 0)
      {
         max_ending_here = 0;
         j=0;
      }
   }
   printf("\nSub_Array with Maximum Sum : \n");
   for(i=0;i<j;i++)
      printf("%d\t",sum_arr[i]);
}
int main()
{
```

```
    int n;
    printf("\nEnter No Of Elements : ");
    scanf("%d",&n);
    int A[n],i;
    printf("\nEnter Array Elements : \n");
    for(i=0;i<n;i++)
       scanf("%d",&A[i]);
    max_subarray(A,n);
}
```

**DATA STRUCTURE USED:** Array

**TIME COMPLEXITY:** O(n)

**OUTPUT:**




**RESULT:**

Thus a program to find the maximum sum of subarray was successfully executed and verified.

**QUESTION:**

Given a rod of length **n** inches and an array of prices that contains prices of all pieces of size smaller than **n**. Determine the maximum value obtainable by cutting up the rod and selling the pieces.

**Input:**

First line consists of **T** test cases. First line of every test case consists of **n**, denoting the size of array. Second line of every test case consists of price of ith length piece.

**Output:**

For each testcase, in a new line, print a single line output consists of maximum price obtained.

**Constraints:**

1 <= T <= 100

1 <= n <= 100

1 <= Ai <= 100

**Example:**

**Input:**

18

1 5 8 9 10 17 17 20

**Output:**

22

| EX NO:3.1(B) | ROD CUTTING PROBLEM |
|---|---|
| DATE: | |

**AIM:**

To find the maximum value obtained by cutting the rod and selling

**PSEUDOCODE:**

```
//Program: to find the  maximum value obtained
//Input:  Integer arrays
//Output: Value
For i←1 to i<=n
        max_val←-9999
        For j←0 to j<i
           max_val←max(max_val, price[j] + val[i-j-1])
        val[i]←max_val
```

**SOURCE CODE:**

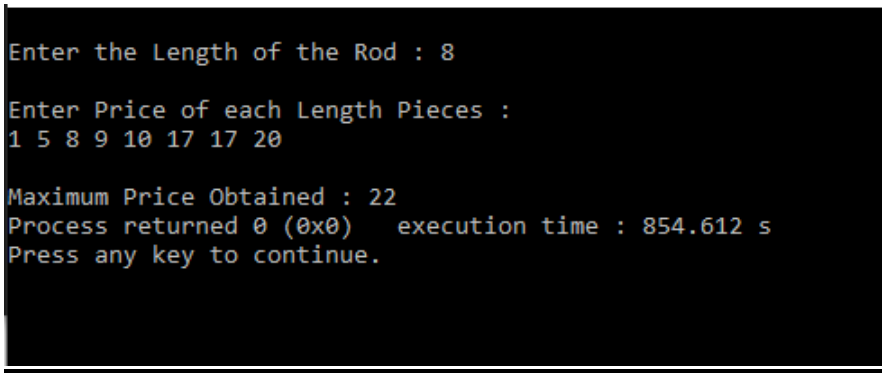
```
#include <stdio.h>
int max(int a, int b)
{
   return (a > b) ? a : b;
}

int cut_rod(int price[], int n)
{
        int val[n+1];
        val[0]=0;
        int i,j;
        for(i=1;i<=n;i++)
        {
                int max_val=-9999;
                for(j=0;j<i;j++)
                   max_val=max(max_val, price[j] + val[i-j-1]);
                val[i]=max_val;
        }
   printf("\nMaximum Price Obtained : %d",val[n]);
}

int main()
{
   int n;
   printf("\nEnter the Length of the Rod : ");
   scanf("%d",&n);
   int price[n],i;
   printf("\nEnter Price of each Length Pieces : \n");
   for(i=0;i<n;i++)
```

```
        scanf("%d",&price[i]);
    cut_rod(price,n);
}
```

**DATA STRUCTURE USED:** Array

**TIME COMPLEXITY:** $O(n^2)$

**OUTPUT:**




**RESULT:**

Thus a program to find the maximum value obtained was successfully executed and verified.

**QUESTION:**

You are given weights and values of **N** items, put these items in a knapsack of capacity **W** to get the maximum total value in the knapsack. Note that we have only **one quantity of each item**.

In other words, given two integer arrays **val[0..N-1]** and **wt[0..N-1]** which represent values and weights associated with **N** items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of **val[]** such that sum of the weights of this subset is smaller than or equal to **W.** You cannot break an item, **either pick the complete item, or don't pick it (0-1 property)**.

**Input:**

The first line of input contains an integer **T** denoting the number of test cases. Then **T** test cases follow. Each test case consists of four lines.

The first line consists of **N** the number of items.

The second line consists of **W**, the maximum capacity of the knapsack.

In the next line are **N** space separated positive integers denoting the values of the **N** items, and in the fourth line are **N** space separated positive integers denoting the weights of the corresponding items.

**Output:**

For each testcase, in a new line, print the **maximum possible** value you can get with the given conditions that you can obtain for each test case in a new line.

**Constraints:**

$1 \leq T \leq 100$

$1 \leq N \leq 1000$

$1 \leq W \leq 1000$

$1 \leq wt[i] \leq 1000$

$1 \leq v[i] \leq 1000$

**Example:**

**Input:**

234

1 2 3

4 5 1

23

1 2 3

4 5 6

**Output:**

31

| EX NO:3.1(C) | MAXIMUM POSSIBLE VALUE |
|---|---|
| DATE: | |

**AIM:**

To find the maximum possible value

**PSEUDOCODE:**

```
//Program: to find the  maximum possible value
//Input:  Items and capacity
//Output: maximu possible value
For  i← 0 to  i <= n
   For t ← 0 to  t <= W
          if (i==0 || t==0)
                K[i][t] ← 0
          else if (wt[i-1] <= t)
                K[i][t] ← max(val[i-1] + K[i-1][t-wt[i-1]], K[i-1][t])
          else
                K[i][t] ← K[i-1][t]
```

**SOURCE CODE:**

```
#include<stdio.h>
int max(int a, int b)
{
   return (a > b)? a : b;
}
int KnapSack(int W, int wt[], int val[], int n)
{
   int i, t;
   int K[n+1][W+1];
   for (i = 0; i <= n; i++)
   {
          for (t = 0; t <= W; t++)
          {
                if (i==0 || t==0)
                      K[i][t] = 0;
                else if (wt[i-1] <= t)
                            K[i][t] = max(val[i-1] + K[i-1][t-wt[i-1]], K[i-1][t]);
                else
                            K[i][t] = K[i-1][t];
          }
   }
   return K[n][W];
}
int main()
{
   int n,W;
   printf("\nEnter the Number of Items : ");
```

```
    scanf("%d",&n);
    printf("\nEnter the Maximum Capacity of the Knapsack : ");
    scanf("%d",&W);
    int val[n],wt[W],i;
    printf("\nEnter the Values of %d Items : \n",n);
    for(i=0;i<n;i++)
       scanf("%d",&val[i]);
    printf("\nEnter the Weights of %d Items : \n",n);
    for(i=0;i<n;i++)
       scanf("%d",&wt[i]);
          printf("\nMaximum Possible Value : %d \n", KnapSack(W, wt, val, n));
}
```

**DATA STRUCTURE USED:** Array

**TIME COMPLEXITY:** $O(n^2)$

**OUTPUT:**

```
Enter the Number of Items : 3

Enter the Maximum Capacity of the Knapsack : 4

Enter the Values of 3 Items :
1 2 3

Enter the Weights of 3 Items :
4 5 1

Maximum Possible Value : 3

Process returned 0 (0x0)   execution time : 21.009 s
Press any key to continue.
```

**RESULT:**

Thus a program to find the maximum possible value was successfully executed and verified.

**QUESTION:**

Given a sequence of matrices, find the most efficient way to multiply these matrices together. The problem is not actually to perform the multiplications, but merely to decide in which order to perform the multiplications. There are many options to multiply a chain of matrices because matrix multiplication is associative i.e. no matter how one parenthesize the product, the result will be the same.

Example:

if you had four matrices A, B, C, and D, you would have:

(ABC)D = (AB)(CD) = A(BCD) = ....

However, the order in which one parenthesize the product affects the number of simple arithmetic operations needed to compute the product, or the efficiency.

For example:

A: 10 × 30 matrix

B : 30 × 5 matrix

C : 5 × 60 matrix

Then,

(AB)C = (10×30×5) + (10×5×60)

= 1500 + 3000

= 4500 operations

A(BC) = (30×5×60) + (10×30×60)

= 9000 + 18000

= 27000 operations.

Given an array **arr[]** which represents the chain of matrices such that the ith matrix Ai is of dimension **arr[i-1] x arr[i]**. Your task is to write a function that should print the minimum number of multiplications needed to multiply the chain.

**Input: p[] = {40, 20, 30, 10, 30}**

**Output: 26000**

There are 4 matrices of dimensions 40x20,
20x30, 30x10 and 10x30. Let the input 4
matrices be A, B, C and D. The minimum
number of multiplications are obtained
by putting parenthesis in following way
(A(BC))D --> 20*30*10 + 40*20*10 + 40*10*30

**Input: p[] = {10, 20, 30, 40, 30}**

**Output: 30000**

There are 4 matrices of dimensions 10x20,
20x30, 30x40 and 40x30. Let the input 4
matrices be A, B, C and D. The minimum
number of multiplications are obtained by
putting parenthesis in following way
((AB)C)D --> 10*20*30 + 10*30*40 + 10*40*30

**Input:**

The first line of the input contains an integer **T**, denoting the number of test cases. Then **T** test case follows. The first line of each test case contains an integer **N**, denoting the number of elements in the array.

Then next line contains **N** space separated integers denoting the values of the element in the array.

| EX NO:3.2(A) | MINIMUM NUMBER OF OPERATION |
|---|---|
| DATE: | |

**AIM:**

To find the minimum no of operation

**PSEUDOCODE:**

```
//Program: to find the  minimum no of operation
//Input:  Array
//Output: maximum operations
For  L ← 2 to  L < n
   For  i← 1 to  i < n - L + 1
          j ← i + L - 1
               m[i][j] ←9999999
               For k = i to k <= j - 1
                  q ← m[i][k] + m[k + 1][j] + p[i - 1] * p[k] * p[j];
                       if (q < m[i][j])
                              m[i][j] ← q
```

**SOURCE CODE:**

```
#include<stdio.h>
int MatrixChainOrder(int n,int p[])
{
        int m[n][n];
        int i, j, k, L, q;
        for(i = 1; i < n; i++)
           m[i][i] = 0;
        for(L = 2; L < n; L++)
        {
           for (i = 1; i < n - L + 1; i++)
                {
                   j = i + L - 1;
                        m[i][j] =9999999;
                        for(k = i; k <= j - 1; k++)
                        {
                           q = m[i][k] + m[k + 1][j] + p[i - 1] * p[k] * p[j];
                                if (q < m[i][j])
                                        m[i][j] = q;
                        }
                }
        }

        return m[1][n - 1];
}

int main()
{
```

**Output:**
For each test case the print the minimum number of operations needed to multiply the chain.
**Constraints:**
1<=T<=100
2<=N<=100
1<=A[]<=500
**Example:**
**Input:**
25
1 2 3 4 5
3
3 3 3
**Output:**
38
27

```
    int n;
    printf("\nEnter No Of Elements : ");
    scanf("%d",&n);
    int A[n],i;
    printf("\nEnter Array Elements : \n");
    for(i=0;i<n;i++)
       scanf("%d",&A[i]);
    printf("\nMinimum Number Of Multiplications is : %d\n",MatrixChainOrder(n,A));
}
```

**DATA STRUCTURE USED:** Array

**TIME COMPLEXITY:** $O(n^3)$

**OUTPUT:**

```
Enter No Of Elements : 5

Enter Array Elements :
1 2 3 4 5

Minimum Number Of Multiplications is : 38

Process returned 0 (0x0)   execution time : 69.231 s
Press any key to continue.
```

**RESULT:**

Thus a program to find the minimum operations was successfully executed and verified.

**QUESTION:**

You are given weights and values of **N** items, put these items in a knapsack of capacity **W** to get the maximum total value in the knapsack. Note that we have only **one quantity of each item**.

In other words, given two integer arrays **val[0..N-1]** and **wt[0..N-1]** which represent values and weights associated with **N** items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of **val[]** such that sum of the weights of this subset is smaller than or equal to **W.** You cannot break an item, **either pick the complete item, or don't pick it (0-1 property)**.

**Input:**

Given a String, find the longest palindromic subsequence

**Input:**

The first line of input contains an integer T, denoting no of test cases. The only line of each test case consists of a string S(only lowercase)

**Output:**

Print the Maximum length possible for palindromic subsequence.

**Constraints:**

1<=T<=100

1<=|Length of String|<=1000

**Examples:**

**Input:**

2

bbabcbcab

abbaab

**Output:**

74

| EX NO:3.2(B) | LONGEST PALINDROMIC SUBSEQUENCE |
|---|---|
| DATE: | |

**AIM:**

To find the longest palindromic subsequence

**PSEUDOCODE:**

```
//Program: to find the longest palindromic subsequence
//Input:  String
//Output: length of the longest palindromic subsequence
For cl←2 to  cl<=n
        For i←0 to i<n-cl+1
           j ← i+cl-1
                If (string[i] == string[j] && cl == 2)
                   L[i][j] ← 2
                else if (string[i] == string[j])
                   L[i][j]← L[i+1][j-1] + 2
                else
                   L[i][j] = max(L[i][j-1], L[i+1][j])
```

**SOURCE CODE:**

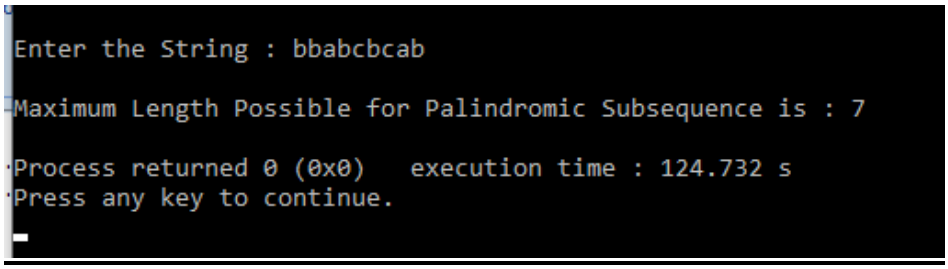
```
#include<stdio.h>
#include<string.h>

int max (int x, int y)
{
   return (x > y)? x : y;
}

int LPS(char string[])
{
   int n = strlen(string);
   int i, j, cl;
   int L[n][n];
   for (i = 0; i < n; i++)
            L[i][i] = 1;
        for (cl=2; cl<=n; cl++)
        {
                for (i=0; i<n-cl+1; i++)
                {
                   j = i+cl-1;
                        if (string[i] == string[j] && cl == 2)
                           L[i][j] = 2;
                        else if (string[i] == string[j])
                           L[i][j] = L[i+1][j-1] + 2;
                        else
                           L[i][j] = max(L[i][j-1], L[i+1][j]);
```

```
                    }
                }
                return L[0][n-1];
        }

        int main()
        {
                char string[1000];
           printf("\nEnter the String : ");
           gets(string);
                printf ("\nMaximum Length Possible for Palindromic Subsequence is :
        %d\n",LPS(string));
        }
```

**DATA STRUCTURE USED:** Array

**TIME COMPLEXITY:** $O(n^2)$

**OUTPUT:**




**RESULT:**

Thus a program to find the length of the longest substring was successfully executed and verified.

**QUESTION:**

Given two sequences, find the length of longest subsequence present in both of them. Both the strings are of uppercase.

**Input:**

First line of the input contains no of test cases **T**,the **T** test cases follow.

Each test case consist of 2 space separated integers **A** and **B** denoting the size of string **str1** and **str2** respectively.

The next two lines contains the 2 string **str1** and **str2** .

**Output:**

For each test case print the length of longest common subsequence of the two strings .

**Constraints:**

1<=T<=200

1<=size(str1),size(str2)<=100

**Example:**

**Input:**

2

6 6

| EX NO:3.2(C) | LONGEST COMMON SEQUENCE |
|---|---|
| DATE: | |

**AIM:**

To find the longest common subsequence

**PSEUDOCODE:**

```
//Program: to find the longest common subsequence
//Input:  String
//Output: length of the longest common subsequence
For i←0 to i<=m
        For j←0 to  j<=n
          if (i == 0 || j == 0)
                L[i][j] ← 0
          else if (str1[i-1] == str2[j-1])
                L[i][j] ← L[i-1][j-1] + 1
           else
                L[i][j] ←max(L[i-1][j], L[i][j-1])
```

**SOURCE CODE:**

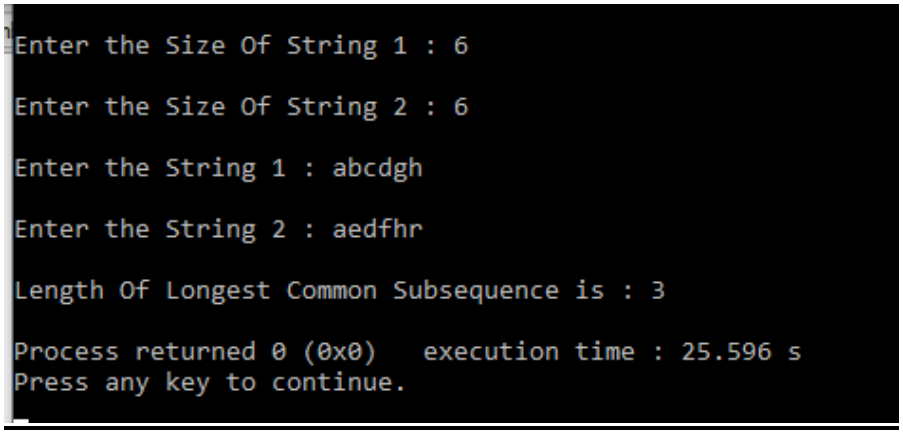
```
#include<stdio.h>
#include<string.h>

int max (int x, int y)
{
   return (x > y)? x : y;
}

int LCS(char str1[], char str2[], int m, int n )
{
   int L[m+1][n+1];
   int i, j;
   for(i=0; i<=m; i++)
   {
        for (j=0; j<=n; j++)
        {
           if (i == 0 || j == 0)
                L[i][j] = 0;
      else if (str1[i-1] == str2[j-1])
                L[i][j] = L[i-1][j-1] + 1;
      else
                L[i][j] = max(L[i-1][j], L[i][j-1]);
     }
   }
      return L[m][n];
}
```

```
int main()
{
   char str1[1000],str2[1000];
   int m,n;
   printf("\nEnter the Size Of String 1 : ");
   scanf("%d",&m);
   printf("\nEnter the Size Of String 2 : ");
   scanf("%d",&n);
   printf("\nEnter the String 1 : ");
   scanf("%s",str1);
   printf("\nEnter the String 2 : ");
   scanf("%s",str2);
   printf("\nLength Of Longest Common Subsequence is : %d\n",LCS(str1,str2,m,n));
}
```

**DATA STRUCTURE USED:** Array

**TIME COMPLEXITY:** $O(n^2)$

**OUTPUT:**




**RESULT:**

Thus a program to find the length of the longest common subsequence was successfully executed and verified.

**QUESTION:**

Given a sorted array keys[0.. n-1] of search keys and an array freq[0.. n-1] of frequency counts, where freq[i] is the number of searches to keys[i]. Construct a binary search tree of all keys such that the total cost of all the searches is as small as possible.

Let us first define the cost of a BST. The cost of a BST node is level of that node multiplied by its frequency. Level of root is 1.

**Input:**

First line consists of test cases T. First line of every test case consists of N, denoting the number of key. Second and Third line consists N spaced elements of keys and frequency respectively.

**Output:**

Print the most minimum optimal cost.

**Constraints:**

1<=T<=100

1<=N<=100

**Example:**

**Input:**

22

10 12

34 50

3

10 12 20

34 8 50

**Output:**

118

142

| EX NO:3.3(A) | MOST MINIMUM OPTIMAL COST |
|---|---|
| DATE: | |

**AIM:**

To find the most minimum optimal cost

**PSEUDOCODE:**

```
//Program: to find the most minimum optimal cost
//Input:  String
//Output: most minimum optimal cost
For  L←2 to L<=n
      For i←0 to i<=n-L+1
         j ← i+L-1
         cost[i][j] ← INT_MAX
         For  r←i to  r<=j
            c ← ((r > i)? cost[i][r-1]:0) + ((r < j)? cost[r+1][j]:0) + sum(freq, i, j)
            if (c < cost[i][j])
               cost[i][j] ← c
```

**SOURCE CODE:**

```
#include <stdio.h>
#include <limits.h>
int sum(int freq[], int i, int j)
{
   int s = 0,k;
   for (k = i; k <=j; k++)
     s += freq[k];
   return s;
}

int Optimal_Binary_Search_Tree(int keys[], int freq[], int n)
{
   int cost[n][n];
   int i,j,r,c,L;
   for (i = 0; i < n; i++)
      cost[i][i] = freq[i];
   for (L=2; L<=n; L++)
   {
      for (i=0; i<=n-L+1; i++)
      {
         j = i+L-1;
         cost[i][j] = INT_MAX;
         for (r=i; r<=j; r++)
         {
            c = ((r > i)? cost[i][r-1]:0) + ((r < j)? cost[r+1][j]:0) + sum(freq, i, j);
            if (c < cost[i][j])
               cost[i][j] = c;
```

```
            }
          }
       }
       return cost[0][n-1];
    }

    int main()
    {
       int n;
       printf("\nEnter the Number Of Keys : ");
       scanf("%d",&n);
       int keys[n],freq[n],i;
       printf("\nEnter the Keys : \n");
       for(i=0;i<n;i++)
          scanf("%d",&keys[i]);
       printf("\nEnter the Frequencies : \n");
       for(i=0;i<n;i++)
          scanf("%d",&freq[i]);
       printf("\nMost Minimum Optimal Cost Of BST is : %d
    \n",Optimal_Binary_Search_Tree(keys,freq,n));
    }
```

**DATA STRUCTURE USED:** Array

**TIME COMPLEXITY:** $O(n^3)$

**OUTPUT:**

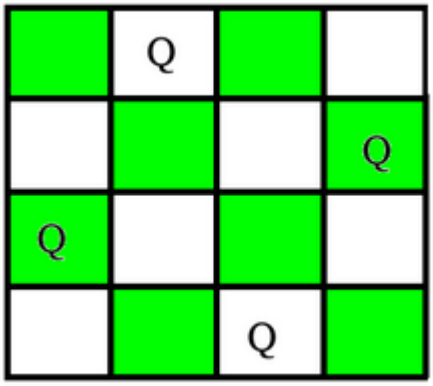
```
Enter the Number Of Keys : 2

Enter the Keys :
10 12

Enter the Frequencies :
34 50

Most Minimum Optimal Cost Of BST is : 118

Process returned 0 (0x0)   execution time : 10.167 s
Press any key to continue.
```

**RESULT:**

Thus a program to find the most minimum optimal cost was successfully executed and verified.

**QUESTION:**

The n-queens puzzle is the problem of placing n queens on an n×n chessboard such that no two queens attack each other. Given an integer n, print all distinct solutions to the n-queens puzzle. Each solution contains distinct board configurations of the n-queens' placement, where the solutions are a permutation of [1,2,3..n] in increasing order, here the number in the *ith* place denotes that the *ith*-column queen is placed in the row with that number. For eg below figure represents a chessboard [3 1 4 2].




**Input:**

The first line of input contains an integer **T** denoting the no of test cases. Then T test cases follow. Each test case contains an integer n denoting the size of the chessboard.

**Output:**

For each test case, output your solutions on one line where each solution is enclosed in square brackets '[', ']' separated by a space . The solutions are permutations of {1, 2, 3 …, *n*} in increasing order where the number in the ith place denotes the ith-column queen is placed in the row with that number, if no solution exists print -1.

**Constraints:**

1 <= T <= 10

1 <= n <= 10

**Example:**

**Input**

214

**Output:**

[1 ]

[2 4 1 3 ] [3 1 4 2 ]

| EX NO:3.3(B) | N QUEEN PROBLEM |
|---|---|
| DATE: | |

**AIM:**

To find the solution for n queen problem

**PSEUDOCODE:**

```
//Program: to find the solution for n queen problem
//Input:  Size
//Output: matrix
solveNQUtil( N,board[N][N], col)
        If (col == N)
              printSolution(N, board)
              return true
        res ←0
        For  i = 0 to i < N
               If ( isSafe(N, board, i, col) )
                        board[i][col] = 1
                        res = solveNQUtil(N, board, col + 1) || res
                        board[i][col] = 0
        return res
```

**SOURCE CODE:**

```
#include <stdio.h>
#include<stdbool.h>
#include<stdlib.h>
#include<string.h>
void printSolution(int N,int board[N][N])
{
   static int k = 1;
   printf("%d-\n",k++);
   for (int i = 0; i < N; i++)
   {
      for (int j = 0; j < N; j++)
         printf(" %d ", board[i][j]);
      printf("\n");
   }
   printf("\n");
}
bool isSafe(int N, int board[N][N], int row, int col)
{
   int i, j;
   for (i = 0; i < col; i++)
      if (board[row][i])
         return false;
   for (i=row, j=col; i>=0 && j>=0; i--, j--)
      if (board[i][j])
```

```
            return false;
        for (i=row, j=col; j>=0 && i<N; i++, j--)
            if (board[i][j])
                return false;
        return true;
    }
    bool solveNQUtil(int N, int board[N][N], int col)
    {
        if (col == N)
        {
            printSolution(N, board);
            return true;
        }
        bool res = false;
        for (int i = 0; i < N; i++)
        {
            if ( isSafe(N, board, i, col) )
            {
                board[i][col] = 1;
                res = solveNQUtil(N, board, col + 1) || res;
                board[i][col] = 0;
            }
        }
        return res;
    }
    void solveNQ(int N)
    {
        int board[N][N];
        memset(board, 0, sizeof(board));
        if (solveNQUtil(N, board, 0) == false)
        {
            printf("Solution does not exist");
            return ;
        }

        return ;
    }
    int main()
    {
        int N;
        printf("\nEnter the Size of the Chessboard : ");
        scanf("%d",&N);
        solveNQ(N);
        return 0;
    }
```

**DATA STRUCTURE USED:** N

**TIME COMPLEXITY:** O(n!)

**OUTPUT:**

```
Enter the Size of the Chessboard : 4
1-
 0  0  1  0
 1  0  0  0
 0  0  0  1
 0  1  0  0

2-
 0  1  0  0
 0  0  0  1
 1  0  0  0
 0  0  1  0


Process returned 0 (0x0)   execution time : 1.788 s
Press any key to continue.
```

**RESULT:**

Thus a program to find the solution for n queen problem was successfully executed and verified.

**QUESTION:**

Given an array of integers and a sum, the task is to print all subsets of given array with sum equal to given sum.

Examples:

Input : arr[] = {2, 3, 5, 6, 8, 10}
sum = 10
Output : 5 2 3
2 8
10
Input : arr[] = {1, 2, 3, 4, 5}
sum = 10
Output : 4 3 2 1
5 3 2
5 4 1

| EX NO:3.3(C) | SUBSET OF THE ARRAY |
|---|---|
| DATE: | |

**AIM:**

To print all subsets of given array with sum equal to given sum

**PSEUDOCODE:**

```
//Program: print the subset of the array
//Input:  array,sum
//Output: subsets
For  i ← 1 to  i <= n
      For  j ← 1 to  j <= sum
         if(j<set[i-1])
            subset[i][j] ← subset[i-1][j];
         if (j >= set[i-1])
            subset[i][j] ← subset[i-1][j] ||
            subset[i - 1][j-set[i-1]];
```

**SOURCE CODE:**

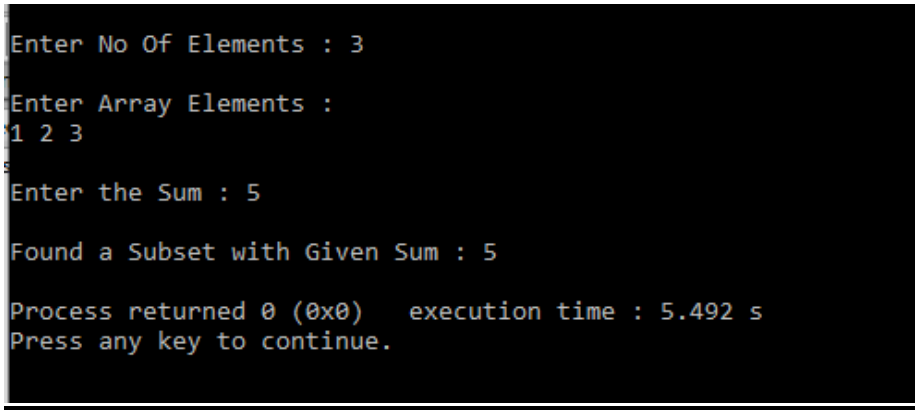
```
#include <stdio.h>
#include<stdbool.h>
#include<stdlib.h>
bool isSubsetSum(int set[], int n, int sum)
{
   bool subset[n+1][sum+1];
   for (int i = 0; i <= n; i++)
      subset[i][0] = true;
   for (int i = 1; i <= sum; i++)
      subset[0][i] = false;
   for (int i = 1; i <= n; i++)
   {
      for (int j = 1; j <= sum; j++)
      {
         if(j<set[i-1])
            subset[i][j] = subset[i-1][j];
         if (j >= set[i-1])
            subset[i][j] = subset[i-1][j] ||
            subset[i - 1][j-set[i-1]];
      }
   }
   return subset[n][sum];
}

int main()
{
   int n;
   printf("\nEnter No Of Elements : ");
```

```
    scanf("%d",&n);
    int set[n],i,sum;
    printf("\nEnter Array Elements : \n");
    for(i=0;i<n;i++)
       scanf("%d",&set[i]);
    printf("\nEnter the Sum : ");
    scanf("%d",&sum);
    int p=isSubsetSum(set, n, sum);
    if (isSubsetSum(set, n, sum) == true)
       printf("\nFound a Subset with Given Sum : %d\n",sum);
    else
       printf("\nNo Subset with Given Sum : %d\n",sum);
    return 0;
}
```

**DATA STRUCTURE USED:** Array

**TIME COMPLEXITY:** $O(n^2)$

**OUTPUT:**




**RESULT:**

Thus a program to find subsets of given array with sum equal to given sum was successfully executed and verified.

**QUESTION:**

We are given a sorted array A[] of n elements. We need to find if x is present in A or not. In binary search we always used middle element, here we will randomly pick one element in given range. Execute an algorithm for randomized binary search.

| EX NO:3.4(A) | RANDOMIZED BINARY SEARCH |
|---|---|
| DATE: | |

**AIM:**

To execute the solution for randomized binary search

**PSEUDOCODE:**

```
//Program: to find the element
//Input:  array,key
//Output: True || false
If(low <= high)
        Mid←Randoms(low,high)
        If(key == A[mid])
                return 1
        Else if(key < A[mid])
                return Randomized_binary_search(A,low,mid-1,key)
        Else
                return Randomized_binary_search(A,mid+1,high,key)
```

**SOURCE CODE:**

```
#include<stdio.h>
#include <stdlib.h>
#include<time.h>
int Randoms(int lower, int upper)
{
   int num = (rand() % (upper - lower + 1)) + lower;
   return num;
}

int Randomized_binary_search (int A[],int low,int high,int key)
{
   int mid;
        srand(time(0));
        if(low <= high)
        {
           mid=Randoms(low,high);
                if(key == A[mid])
                        return 1;
                else if(key < A[mid])
                        return Randomized_binary_search(A,low,mid-1,key);
                else
                        return Randomized_binary_search(A,mid+1,high,key);
            }
     return 0;
}

int main()
```

```
{
   int n;
        printf("\nEnter the number of elements : ");
        scanf("%d",&n);
        int A[n],i,key;
   printf("\nEnter the elements : \n");
        for(i=0;i<=n-1;i++)
                scanf("%d",&A[i]);
        printf("\nEnter the search element : ");
        scanf("%d",&key);
        if(Randomized_binary_search(A,0,n-1,key))
      printf("\nELEMENT %d IS FOUND\n",key);
   else
      printf("\nELEMENT %d IS NOT FOUND\n",key);
   return 0;
}
```

**DATA STRUCTURE USED:** Array

**TIME COMPLEXITY:** O(log n)

**OUTPUT:**

```
Enter the number of elements : 5

Enter the elements :
1 3 4 6 7

Enter the search element : 6

ELEMENT 6 IS FOUND

Process returned 0 (0x0)   execution time : 16.258 s
Press any key to continue.
```

**RESULT:**

Thus a program to implement the randomized binary search was successfully executed and verified.

**QUESTION:**

By making use of a pseudo-random number generator to simulate random choices of the pivot element, we can make QuickSort behave as if whatever input it receives is actually an average case. Execute an algorithm for randomized quick sort.

| EX NO:3.4(B) | RANDOMIZED QUICK SORT |
|---|---|
| DATE: | |

**AIM:**

To execute the solution for randomized quick sort

**PSEUDOCODE:**

```
//Program: to sort the array
//Input:  array
//Output: sorted array
while(i < j)
      while(A[i] < piv)
           i++
     while(A[j] > piv)
          j--
    temp←A[i]
     A[i]←A[j]
     A[j]←temp
```

**SOURCE CODE:**

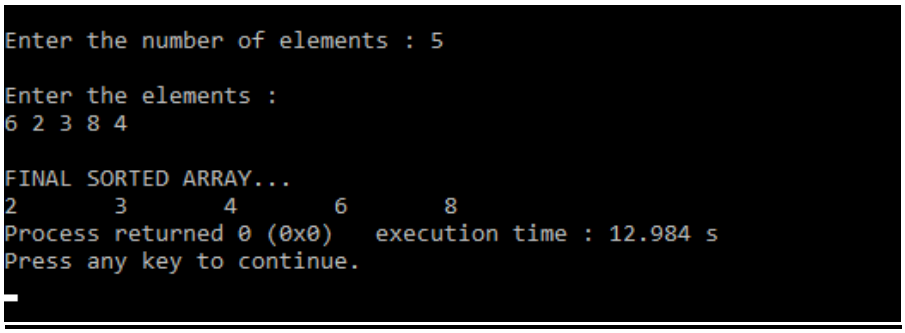
```
#include<stdio.h>
#include <stdlib.h>
#include<time.h>
int Randoms(int lower, int upper)
{
   int num = (rand() % (upper - lower + 1)) + lower;
   return num;
}

void Randomized_Quick_Sort(int A[],int low,int high)
{
   int i,j,temp,piv;
   if(low < high)
   {
      i=low;
      j=high;
      srand(time(0));
      piv=A[Randoms(low,high)];
      while(i < j)
      {
         while(A[i] < piv)
            i++;
         while(A[j] > piv)
            j--;
         temp=A[i];
         A[i]=A[j];
         A[j]=temp;
```

```
        }
        Randomized_Quick_Sort(A,low,j-1);
        Randomized_Quick_Sort(A,j+1,high);
    }
}

int main()
{
    int n;
        printf("\nEnter the number of elements : ");
        scanf("%d",&n);
    int a[n],i;
    printf("\nEnter the elements : \n");
        for(i=0;i<=n-1;i++)
                scanf("%d",&a[i]);
    Randomized_Quick_Sort(a,0,n-1);
    printf("\nFINAL SORTED ARRAY...\n");
    for(i=0;i<n;i++)
      printf("%d\t",a[i]);
}
```

**DATA STRUCTURE USED:** Array

**TIME COMPLEXITY:** O(n log n)

**OUTPUT:**




**RESULT:**

Thus a program to implement the randomized quick sort was successfully executed and verified.