IE CP SMP'20

Binary Search:

Q. You are given an integer array of size N which is initially strictly increasing and then strictly decreasing. It might be only strictly increasing or strictly decreasing. Your task is to find the max element in the given valid array in o(log[N]).

```cpp
#include<bits/stdc++.h>
using namespace std;
 int maxArr(int arr[]);
 int main(){
     int N;
     cout<<"Enter size of array : ";
     cin>>N;
     int arr[N];
     cout<<"Enter the values of array elements :"<<endl;
     for (int i = 0; i < N; i++)
        cin>>arr[i];
     cout<<maxArr(arr);
 }
 int maxArr(int arr[]){
     int l=0,h,mid;
     h = 4;
     mid = l + (h-l)/2;
     int flag = 1;
     while(flag){
         if(arr[mid]>arr[mid-1]&&arr[mid]>arr[mid+1]||mid==l||mid==h){
             flag = 0;
             return arr[mid];
        }
         if(arr[mid]>arr[mid-1]){
             l = mid;
             mid = l + (h-l)/2;
         }
         else if(arr[mid]>arr[mid+1]){
             h = mid;
             mid = l + (h-l)/2;
         }
     }
 }
```

Q. Farmer John has built N stalls. The stalls are located along a straight line at positions x1…xN. His C cows don't like this barn layout and become aggressive towards each other once put into a stall. To prevent the cows from hurting each other, John wants to assign the cows to the stalls, such that the minimum distance between any two of them is as large as possible.

What is the largest minimum distance?

```cpp
#include<bits/stdc++.h>
using namespace std;

int N,C;
bool check ( int mid , int x[]) {
    int cow_placed = 1,previousPos = x[0]; // assuming we places 1st cow at x1
    for( int i = 1; i < N; i++) {
        if ( x[i] - previousPos >=mid ) {
            previousPos = x[i] ;
            cow_placed++;
        }
        if(cow_placed == C ) return true;
    }
    return false;
}

int main(){
    int low,high,ans=0,mid;
    cout<<"Enter the no. of stalls and cows : ";
    cin>>N>>C;
    int x[N];
    cout<<"Enter the positions of stalls : "<<endl;
    for(int i=0; i<N; i++)
        cin>>x[i];
    low = 1;
    high = x[N-1] - x[1];
    while( low <= high){
        mid = low + (high - low)/2;
        if( check(mid,x) ){
            ans = max( ans, mid);
            low = mid + 1;
        }else{
            high = mid - 1;
        }
    }
    cout<<"The largest minimum distance is "<<ans;
}
```

Q. Little Petya likes points a lot. Recently his mom has presented him n points lying on the line OX.

Now Petya is wondering how many ways he can choose three distinct points so that the distance between the two farthest of them doesn't exceed d.

Note that the order of the points inside the group of three chosen points doesn't matter.

INPUT: The first line contains two integers: n and d.

The next line contains n integers x1…xN.

The coordinates of the points in the input strictly increase.

OUTPUT: Print the number of groups of three points, where the distance between two farthest points doesn't exceed d.

```cpp
#include<bits/stdc++.h>
using namespace std;

int main(){
    int ans=0;
    int n,d;
    cout<<"Enter the values of n and d : ";
    cin>>n>>d;
    vector<int> v(n);
    cout<<"Enter the n coordinates :"<<endl;
    for (int i = 0; i < n; i++)
        cin>>v[i];
    for(int i = 0; i<n-2; i++){
        long long a= (upper_bound(v.begin(),v.end(),v[i]+d)-v.begin());
        long long b = a - i - 1;
        b=(b*(b-1))/2;
        ans+=b;
    }
    cout<<"The number of groups of three points are "<<ans;
}
```

Algo:

PRESUM: Given an array of size n, consisting of integers.

You have to tell the sum of all integers in a given range (l to r (inclusive))?

```cpp
#include<iostream>
using namespace std;
void presum(int arr[],int n,int l,int r)
{
    int pre_sum[n+1]={0};
    for(int i=1;i<=n;i++)
        pre_sum[i]=pre_sum[i-1]+arr[i-1];
    cout<<pre_sum[r]-pre_sum[l-1];
}
int main(){
    int n;
    cout<<"Enter size of array : ";
    cin>>n;
    int arr[n], l, r;
    cout<<"Enter the elements of aray : "<<endl;
    for (int i = 0; i < n; i++)
        cin>>arr[i];
    cout<<"Enter values of l and r : ";
    cin>>l>>r;
    presum(arr, n, l, r);
}
```

KADANE: Calculate the maximum subarray sum in a given array.

```cpp
#include<iostream>
using namespace std;
int main(){
    int n;
    cout<<"Enter the size of array : ";
    cin>>n;
    int a[n];
    cout<<"Enter the elements of array : "<<endl;
    for (int i=0; i<n;i++)
        cin>>a[i];
    int maxi=a[0];
    int end=a[0];
    for (int i=1;i<n;i++) {
        end=max(a[i],end+a[i]);
        maxi=max(maxi,end);
    }
    cout<<maxi;
}
```

SIEVE OF ERATOSTHENES: An algorithm to find the primes less than a given number.

```
#include<iostream>
using namespace std;

#define N  1000005
int prime[N];

void seive()
{
    for(int i=2;i<=N;i++)
    {
        if(!prime[i])
        {
            for(int j=i;j<=N;j+=i)
                if(!prime[j])
                    prime[j]=i;
        }
    }
}

int main()
{
    seive();
    int n, primes=0;
    cout<<"Enter the value of n : ";
    cin>>n;
    for(int i=0; i<n; i++)
        if(prime[i]==i)
            primes++;
    cout<<"The no. of primes less than n are : "<<primes;
}
```

Greedy Algorithm:

Q. Given a string and a positive number k.

Find the length of longest substring of given string containing k distinct characters.

```
#include<iostream>
#include<string>
using namespace std;

int main()
{
    int f[300], d=0;
    int i=0, j=0, k=0, ans=0;
    string s;
    cout<<"Enter String and k"<<endl;
    cin>>s>>k;
    for(int a=0; a<300; a++)
        f[a] = 0;
    for(j=i; j<s.length(); j++){
        f[(int)s[j]] += 1;
        if(f[(int)s[j]] == 1) d += 1;
        if(d == k) ans = ans>(j-i + 1) ? ans : (j-i) + 1;
        else if(d>k){
            for(; i<j; i++){
                f[(int)s[i]] -= 1;
                if(f[(int)s[i]] == 0){
                    d--;
                    i++;
                    break;
                }
            }
        }
    }
    cout<<"The max length is "<<ans;
}
```

Q. Given two arrays A and B sorted in ascending order and an integer x.

We need to find I and j, such that (a[i] + b[j]) is equal to x.

n = length of array A

m = length of array B

```cpp
#include<iostream>
using namespace std;

int main(){
    int A[1000], B[1000];
    int n,m,x;
    cout<<"Enter size of array A and B : "<<endl;
    cin>>n>>m;
    cout<<"Enter values of array A :"<<endl;
    for (int i = 0; i < n; i++)
        cin>>A[i];
    cout<<"Enter values of array B :"<<endl;
    for (int i = 0; i < m; i++)
        cin>>B[i];
    cout<<"Enter value of x : "<<endl;
    cin>>x;
    int j=m-1;
    for (int i=0; i<n; ++i)
    {
        while ((j>0) && (A[i]+B[j]>x))
        {
            j--;
        }
        if (A[i]+B[j]==x)
        {
            cout<<i<<" "<<j<<endl;
            return 0;
        }
    }
    cout<<"Not found!"<<endl;
}
```

Q. Given weights and values of n items.

We need to put these items in a knapsack of capacity W to get the maximum total value in the knapsack.

We can break items for maximizing the total value of knapsack.

```
#include<iostream>
using namespace std;

int main(){
    int n, W, max=0;
    cout<<"Enter number of items : ";
    cin>>n;
    float arr[n][2], ratio[n]; // arr of n weights[1] and n values[0]
    cout<<"Enter weights and values of n items :"<<endl;
    for (int i = 0; i < n; i++){
    cin>>arr[i][0]>>arr[i][1];
    ratio[i] = arr[i][0]/arr[i][1];
    }
    cout<<"Enter capacity of knapsack : ";
    cin>>W;
    for(int i=0; i<n; i++)
        for(int j=i; j<n; j++)
            if(ratio[i]<ratio[j]){
                ratio[i] += ratio[j] - (ratio[j] = ratio[i]);
                arr[i][0] += arr[j][0] - (arr[j][0] = arr[i][0]);
                arr[i][1] += arr[j][1] - (arr[j][1] = arr[i][1]);
            }
    int i=0;
    while(W){
        if(W-arr[i][1]>=0){
            W -= arr[i][1];
            max += arr[i][0];
            i++;
        }
        else{
            max += arr[i][0] * W / arr[i][1];
            W = 0;
        }
    }
    cout<<"The maximum value in Knapsack is : "<<max;
}
```

Q. You have n jobs. Each job can be completed at any time and takes time ti time to complete and has a point value of pi.

But with each second, the point value of the ith job decease by di.

If you have to complete all the jobs, What is the maximum points that you can get?

For ith job where 1 <= i <= n,

ti = time taken to complete the job

pi = points received after completing the job

di = points decreased from pi every second till the job hasn't finished

```cpp
#include<iostream>
using namespace std;

int main(){
    int n;
    cout<<"Enter the number of jobs : ";
    cin>>n;
    int t[n], p[n], d[n], ratio[n];
    cout<<"Enter time taken, points received and points decreased for n jobs :
 "<<endl;
    for (int i = 0; i < n; i++){
        cin>>t[i]>>p[i]>>d[i];
        ratio[i] = d[i] / t[i];
    }
    int points = 0, time = 0;
    for(int i=0; i<n; i++)
        for(int j=i; j<n; j++)
            if(ratio[i]<ratio[j]){
                ratio[i] += ratio[j] - (ratio[j] = ratio[i]);
                t[i] += t[j] - (t[j] = t[i]);
                p[i] += p[j] - (p[j] = p[i]);
                d[i] += d[j] - (d[j] = d[i]);
            }
    for (int i = 0; i < n; i++)
    {
        time += t[i];
        points += p[i] - d[i]*time;
    }
    cout<<"The maximum points are : "<<points;
}
```

Dynamic Programming:

Q. On a positive integer, you can perform any one of the following three steps –

1. **Subtract 1 from it (n = n-1)**
2. **If its divisible by 2, divide by 2 (if n%2==0, then n = n/2)**
3. **If its divisible by 3, divide by 3 (if n%3==0, then n = n/3)**

So, given a positive integer n, find the minimum number of steps that takes n to 1.

```cpp
#include<bits/stdc++.h>
using namespace std;
int getMinSteps ( int n )
{
    int dp[n+1] , i;

    dp[1] = 0;  // base case
    for( i = 2 ; i <= n ; i ++ )
    {
        dp[i] = 1 + dp[i-1];
        if(i%2==0) dp[i] = min( dp[i] , 1+ dp[i/2] );
        if(i%3==0) dp[i] = min( dp[i] , 1+ dp[i/3] );
    }
    return dp[n];
}
int main(){
    int n;
    cout<<"Enter the positive integer : ";
    cin>>n;
    cout<<getMinSteps(n);
}
```

Q. N buildings are built in a row, numbered 1 to N from left to right.

Spiderman is on building number 1, and want to reach building number N.

He can jump from building number I to building number j (if i<j and j-I is a power of 2).

Such a move costs him energy |Height[j] – Height[i]|, where Height[i] is the height of ith building.

Find the minimum energy using which he can reach building N?

```
#include<bits/stdc++.h>
using namespace std;
int min_energy(int n,int h[])
{
        int dp[n+1];
        dp[1]=0;
        for(int i=2;i<=n;i++)
        {
            dp[i]=INT_MAX;
            for(int j=1;j<i;j*=2)
                  dp[i]=min(dp[i-j]+abs(h[i]-h[i-j]),dp[i]);
        }
        return dp[n];
}
int main(){
    int n;
    cout<<"Enter the no. of buildings : ";
    cin>>n;
    int h[n+1];
    cout<<"Enter the heights of buildings :"<<endl;
    for (int i = 1; i <= n; i++)
        cin>>h[i];
    cout<<"The minimum required energy is "<<min_energy(n,h);
}
```

Q. Given weights and values of n items.

We need to put items in a knapsack of capacity W to get the maximum total value in the knapsack.

```cpp
#include<iostream>
using namespace std;
int KnapSack(int W, int wt[], int val[], int n){
    int i, j;
    int DP[n+1][W+1];
    for(i=0; i<=n; i++)
        for(j=0; j<=W; j++){
            if(i==0||j==0)
            DP[i][j]=0;
            else if(wt[i-1]<=j)
            DP[i][j]= max(val[i-1] + DP[i-1][j-wt[i-1]] , DP[i-1][j]);
            else
            DP[i][j]=DP[i-1][j];
        }
return DP[n][W];
}
int main(){
    int n=4, val[]={60, 100, 120, 200}, wt[]={10, 20, 30, 40}, W=60;
    cout<<KnapSack(W, wt, val, n);
}
```

Q. Given two sequences, find the length of longest subsequence present in both of them.

```cpp
#include<iostream>
using namespace std;
int LCS(int n, int m);
string s="tezcatlipoca" , t="quetzalcoatl";
const int n=12, m=12;
int DP[n+1][m+1];
int main(){
    for (int i = 0; i <= n; i++)
        for (int j = 0; j <= m; j++)
            DP[i][j] = -1;
    cout<<LCS(n,m);
}
int LCS(int n, int m){
    if(m==0||n==0) return 0;
    if(DP[n][m]!=-1) return DP[n][m];
    if(s[n-1] == t[m-1]) return DP[n][m] = 1+ LCS(n-1, m-1);
    return DP[n][m] = max(LCS(n,m-1), LCS(n-1,m));
}
```

Have a nice coding ahead :)

: Akshay Jain