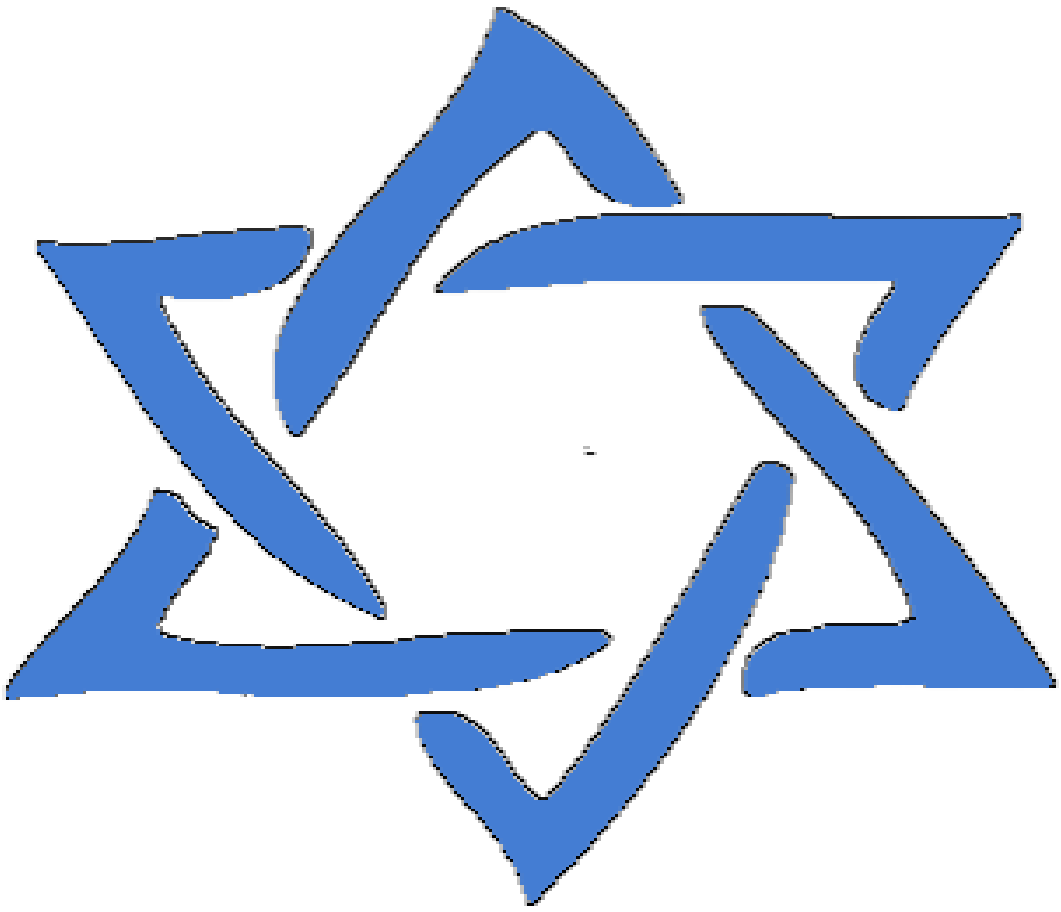
Binary search
Lesson 6



TARS AND THE MARKET

On a trip to jupiter and mars, TARS went to a space market to buy some goodies for his friends and relatives.

The market has some strange rules. It contains n different items numbered from 1 to n. The i-th item has base cost bi space pounds.

If TARS buys k items with indices x1, x2, ..., xk, then the cost of item xj is bxj + xj·k for 1 ≤ j ≤ k. In other words, the cost of an item is equal to its base cost plus the index multiplied by the factor k.

TARS wants to buy as many goodies as possible without paying more than SP space pounds. Note that he cannot buy an items more than once. your task is to help him with this task.

INPUT:

The first line contains two integers n and SP (1 ≤ n ≤ 10^5 and 1 ≤ SP ≤ 10^9) — the number of items in the market and TARS budget.

The second line contains n space-separated integers b1, b2, ..., an

(1 ≤ bi ≤ 10^5) — the base costs of the items

OUTPUT:

On a single line, print two integers k, T — the maximum number of items TARS can buy and the minimum total cost to buy these k items.

Sample Input:

3 11

2 3 5

Sample Output:

2 11

Explanation: In the example, he can't take all three items because they will cost him [5, 9, 14] and its total cost 28. now If he decides to take only two items, then the costs will be [4, 7, 11]. So now he can take the first and second items.

ALGORITHM:

Naive approach: iterate over the array ,firstly choose 1 item and find cost of all items and then check can we buy 1 items.. If yes then go and check for higher items i.e, for 2 items, 3 items ...so on... ,...else if we cannot buy the considered no. of items then come out of the loop, and write the maximum no. of items we can buy and the minimum cost to buy these items.

Time complexity:O(N^2)

Using binary search:

- Take a variable low=0,high=n.
- Every time calculates the mid=(low+high)/2;and consider that we want to buy mid numbers of item, then find cost of all items and check that we can buy mid items or not.
- If we can buy then check for larger one(to buy items more than mid), in right part.i.e,low=mid+1
- Else go in left part, high=mid-1;
- Finally return the maximum no of items bought and cost to buy those items.

TIME COMPLEXITY:O(NLOGN)

CODE:

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long int

int main()
{
ll n,m;
cin>>n>>m;
ll a[n+1];
```

```
for(ll i=1;i<=n;i++)
  cin>>a[i];
vector<ll>v;
ll low=0,high=n,ans1=0,ans2=0;
while(low<=high)
{
    ll mid=(low+high)/2;
    v.clear();
    for(ll i=1;i<=n;i++)
    {
            v.push_back(a[i]+mid*i);
    }
    sort(v.begin(),v.end());
    ll cost=0;
    for(ll i=0;i<mid;i++)
    {
            cost+=v[i];
    }
    if(cost<=m)
    {
            if(mid>ans1)
            {
                    ans1=mid;
                    ans2=cost;
            }
            else if(mid==ans1)
            {
                    if(cost<ans2)
                      cost=ans2;
            }
            low=mid+1;
    }
    else
      high=mid-1;
}
cout<<ans1<<" "<<ans2<<endl;
}
```

Que link:[Tars and the market]

NUMBERS II

Given two numbers a and b, you have to find the number which is divisible by a or b.

Input :

First line consists of an integer T, denoting the number of test cases.

Second line contains three integers a, b and N .

Output :

For each test case, print the number in a new line.

Sample Input:

```
1
2 3 10
```

Sample Output:

```
15
```

Explanation:

The numbers which are divisible by 2 or 3 are:

2,3,4,6,8,9,10,12,14,15, and the 10 th number is 15.

ALGORITHM:

Naive Approach: A simple approach is to traverse over all the terms starting from 1 until we find the desired Nth term which is divisible by either a, b or c. This solution has time complexity of O(N).

Efficient Approach: The idea is to use Binary search. Here we can calculate how many numbers from 1 to num are divisible by either a, b using the formula: (num / a) + (num / b) – (num / lcm(a, b)) .
TIME COMPLEXITY:O(logN)

CODE:

```
#include<bits/stdc++.h>
#define int int64_t
#define pb push_back
#define sz(x) (int)(x.size())
#define ALL(x) (x).begin(),(x).end()
#define F0R(i,R) for(int i = (0); i < (R); ++i)
#define FOR(i,L,R) for(int i = (L); i <= (R); ++i)
using namespace std;


const int inf = 1e18;
    const int N = 5e5 + 5;

    void solve() {
        int a, b, n;
        cin >> a >> b >> n;
        int l = a * b / __gcd(a, b);
        int lo = 0, hi = inf;
        int ans = -1;
        while (lo <= hi) {
            int mid = (lo +hi) / 2;
            int cnt = mid / a + mid / b - mid / l;
            if (cnt >= n)
            { ans = mid,
            hi = mid - 1;
            }
            else lo = mid + 1;
        }

        cout << ans << '\n';
    }

    int32_t main() {
        ios::sync_with_stdio(0);
        cin.tie(0);
        int T;
        cin >> T;
        while (T--)
        {solve();
        }
    }
```

Que link:[Numbers 2]

CHARSI IN LOVE

Its been a few days since Charsi is acting weird. And finally you(his best friend) came to know that its because his proposal has been rejected.

He is trying hard to solve this problem but because of the rejection thing he can't really focus. Can you help him? The question is: Given a number n , find if n can be represented as the sum of 2 desperate numbers (not necessarily different) , where desperate numbers are those which can be written in the form of (a*(a+1))/2 where a > 0 .

Input :

The first input line contains an integer n ($1 \le n \le 10^9$).

Output :

Print "YES" (without the quotes), if n can be represented as a sum of two desperate numbers, otherwise print "NO" (without the quotes).

Sample Input: 256

Sample Output

YES

ALGORITHM:

Naive approach:

Iterate till n/2 and check if i & n-i are sum of natural numbers or not.

It yes ,return true

Else ,false. **TIME COMPLEXITY:O(N*sqrt(N))**

The idea is to use the formulae of the sum of first N natural numbers to compute the value of the N.

→ *1 + 2 + 3 + …. N = S*

→ *(N * (N + 1)) / 2 = S*

→ *N * (N + 1) = 2 * S*

→ *N2 + N – 2 * S = 0 if N *(N+1)=2*S then it is sum of N natural numbers.*

Code:

```
#include<bits/stdc++.h>
using namespace std;
bool is(long long i)
{
   long long root=sqrt(2*i);
   if(root*(root+1)==(i*2))
      return true;
   return false;

```

```
 }
int main()
 {
    long long n; cin>>n;
    int cnt=0;
    for(long long i=1;i<=n/2;i++)
    {
        if(is(i)&&is(n-i))
        {
            cnt++;
            cout<<"YES";
            break;
        }
    }
    if(cnt==0)
      cout<<"NO"<<endl;
 }
```

Using binary search:

- Iterate a loop from loop i=1 to 10^5
- Take a variable low=1,high=1e5, every time calculates the mid
- if(sum of i natural no +sum of mid natural no ==N) then return yes.

- Else if it is less than n ,check in right part ,i.e,low=mid+1;
- Else high=mid-1.

Time complexity:O(NLOGN)

CODE:

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long int

int main()
{
ll n;
cin>>n;
ll i;
    for(i=1;i<=1e5;i++)
{
        ll low=1,high=1e5,mid,m;
        while(low<=high)
        {
                mid=(low+high)/2;
                m=((i*(i+1)/2)+(mid*(mid+1)/2));
                if(m<n)
                 low=mid+1;
                 else if(m>n)
                  high=mid-1;
```

```
                    else{
                        cout<<"YES";
                        return 0;
                    }
        }
    }
    cout<<"NO";
    return 0;
    }
```

Que link:[charsi in love]

GAURAV AND SUBARRAY

You are given an array A[] consisting of N non-negative integers. Now, you need to answer Q queries of the following type given an integer K in each query.

You need to find the minimum length L of any subarray of A, such that if all elements of this subarray are represented in binary notation and concatenated to form a binary string, then no of 1's in the resulting string is at least K.

Input Format:

The first line of the input consists of two space-separated integers N and Q.

The second line contains N space separated integers, where the i th integer denotes A[i]

Next Q lines contains a non-negative integer K.

Output Format: For each query out of the Q ones, print the answer on a new line. If for a particular query no valid subarray exists, then print -1 instead as the answer to that query.

Sample Input

```
4 3
1 2 4 8
1
2
3
```

Sample Output

```
1
2
3
```

Explanation:

For first query consider subarray A[1,1], then binary string representing A[1,1] is 01 which has one 1's.

For second query consider subarray A[1,2], then binary string is 0110 which has two 1's.

Similarly, for third query consider subarray A[1,3].

ALGORITHM:

Using binary search: Make a array bits[] that stores the no of set (1's)bits in arr[],for each element. Make a array cum[] that stores total no. of set bits upto that index.

For each query do the following steps:

- Take a variable low=1,high=n.
- Start a loop till low<=high. Every time calculates the mid=(low+high)/2;
- And check that the the subarray of mid numbers have set bits greater than in the given query ,store the mid in answer variable and further check in lower half.
- Else check in right half. Finally return the answer.

TIME COMPLEXITY:O(Q*NLogN)

CODE:

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long int

const long long m=1e9+7;
ll solve(ll x)
{
        ll count=0;
        while(x>0)
        {
                int rem=x%2;
                if(rem)
                  count++;
                x=x/2;
        }
        return count;
}

bool check(ll cum[],ll n,ll mid,ll t)
{
        for(ll i=0;i<=n-mid;i++)
        {
                if(cum[i+mid]-cum[i]>=t)
                   return true;
        }
        return false;
}
int main()
{
        ios_base::sync_with_stdio(false);
        cin.tie(NULL);
        cout.tie(NULL);
        ll n,q,k;
        cin>>n>>q;
        ll a[n+1];
        for(ll i=1;i<=n;i++)
           cin>>a[i];
        ll bits[n+1];
        ll cum[n+1];
        for(ll i=1;i<=n;i++)
        {
                bits[i]=solve(a[i]);
        }
        cum[0]=0;
        for(ll i=1;i<=n;i++)
        {
                cum[i]=cum[i-1]+bits[i];
        }

        while(q--)
        {
                cin>>k;
                bool flag=false;
         ll low=1,high=n,ans;
         while(low<=high)
         {
                 ll mid=(low+high)/2;
                 if(check(cum,n,mid,k))
                 {
                         high=mid-1;
                         ans=mid;
                         flag=true;
                }
                else{
                        low=mid+1;
                }
        }
        if(flag==1)
         cout<<ans<<"\n";
        else
         cout<<"-1"<<"\n";
        }
        return 0;
}
```

Que link:gaurav and subarray

SHERLOCK AND NUMBERS

Watson gives to Sherlock a bag of numbers [1, 2, 3 ... N] and then he removes K numbers A1, A2 ... AK from the bag. He now asks Sherlock to find the P'th smallest number in the bag.

Input :
First line contains T, the number of test cases. Each test case consists of N, K and P followed by K integers in next line denoting the array A.
Output :
For each test case, print P'th smallest number in the bag. If no such number exists output -1.
Sample Input:

```
2
4 1 2
1
5 2 4
1 3
```

Sample Output:

```
3
-1
```

Explanation:
Test case 1: Remaining numbers are [2, 3, 4]. 3 is the 2nd smallest remaining number.

Test case 2: Remaining numbers are [2, 4, 5]. The 4th smallest remaining number doesn't exist.

ALGORITHM:

Naive Approach:

Create the resultant array by skipping elements present in K[] array (elements to be removed). And then return arr[P - 1].

Better Approach:

Any **i**th number in the series of n natural numbers is **i** . If we remove **k** elements that are lesser than i. Then the ith element is **(i + k)** .

We will use this logic to find the **P**th smallest element in the series.

- The lower index in our search space will be **P** (When no element less than P is removed)
- The higher index will be the upper limit of series i.e. **N** .
- In every binary search iteration, we'll find the number of elements that have to be removed and are less than or equal to mid. (**remove**)
- The position of mid in the resultant series will be **(mid-remove).**
- Now compare and keep iterating.

Time Complexity(for each test case) : O (Log (N - P) * Log(K))

Space Complexity(for each test case) : O (K)

CODE:

```
#include<bits/stdc++.h>
```

```cpp
using namespace std;
#define ll int
#define endl "\n"

int main()
{
   ios_base::sync_with_stdio(false);
   cin.tie(NULL);
   cout.tie(NULL);

      int T; cin>>T;
      while(T--)
      {
          ll N, K, P;
          cin>>N>>K>>P;
          ll arr[K];
          for(ll i = 0; i<K; i++)
              cin>>arr[i];

          ll low = P, high = N;
          int flag = 0;
          while(low <= high)
          {
              ll mid = low + (high - low)/2;
```

```
            ll remove = upper_bound(arr, arr + K, mid) - arr;

            if(mid - remove == P)
            {
                if(arr[remove - 1] == mid)
                {
                    high = mid - 1;
                    continue;
                }
                cout<<mid<<endl;
                flag++;
                break;
            }
            else if(mid - remove > P)
                high = mid - 1;
            else
                low = mid + 1;
        }
        if(flag == 0)
            cout<<-1<<endl;
    }
    return 0;
}
```

QUES LINK:Sherlock and Numbers

A SPECIAL SEQUENCE

An array contains integers with the following constraints:

- A contains elements in sorted order.
- Integer i occurs { i * floor(sqrt(i)) + ceil(i/2) } times in the array.
- All elements are greater than or equal to 1 .

You are given Q queries of type:

- L R : Find the number of distinct values present in subarray A[L:R] .

Note: 1-based indexing is followed.

Input :
The first line contains an integer Q denoting the number of queries.
Next Q lines contain two space-separated integers L R , denoting the query.

Output :
For each query in a new line, print the required number of distinct values.

Constraints :
$1 \leq Q \leq 10^5$
$1 \leq L \leq R \leq 10^{13}$

Sample Input:

```
2
1 3
1 6
```

Sample Output:

```
2
3
```

Explanation:

First few elements of the array A are

1, 1, 2, 2, 2, 3, 3, 3, 3, 3, ...

For Query 1:-

Number of distinct elements in subarray A[1...3] is 2.

For Query 2:-

Number of distinct elements in subarray A[1...6] is 3.

ALGORITHM:

1. Any element **i** occurs { **i * floor(sqrt(i)) + ceil(i/2)** } times in the series.
2. The broad idea is to store the number of occurrences of **i** in an array. And create a **SumArray** (i.e. it stores the sum of all elements upto that index). [In this array, for any index i it shows the position of last occurrence of the value i in the special sequence]

3. Now, for a given value of **L** and **R**, find in SumArray; the index of an element equal or just greater than L and R.
4. Find the difference and we get the number of distinct values.

[Step 3 : Is performed using binary Search]

Time Complexity(for utility Function) : O(N)
Space Complexity : O(N)

Time Complexity(for each query) : O(Log(N))
N = 10^6

CODE:

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define endl "\n"

ll BS(ll arr[], ll N, ll ele)
{
    ll low = 0, high = N-1, ans = 0;
    while(low <= high)
    {
        ll mid = low + (high - low)/2;
        if(arr[mid] >= ele)
        {
            ans = mid;
            high  = mid - 1;
        }
        else
            low = mid + 1;
    }
    return ans;
    }

    void utility(ll arr[])
    {
    ll sum = 0;
    arr[0] = 0;
    for(ll i = 1; i< 1000000; i++)
    {
        float a = i;
        ll value = (i * floor(sqrt(i))) + ceil(a/2);
        sum += value;
        arr[i] = sum;
    }
    }

    int main()
    {
```

```
    int Q; cin>>Q;
    ll arr[1000000];
    utility(arr);

    while(Q--)
    {
        ll L, R; cin>>L>>R;
        ll low = BS(arr, 1000000, L);
        ll high = BS(arr, 1000000, R);
       //ll low=lower_bound(arr,arr+1000000,L)-arr;
       //ll high=lower_bound(arr,arr+1000000,R)-arr;

        cout<<high - low + 1<<endl;
    }
    return 0;
}
```

QUES LINK: A special sequence

PRIME CUBES

Tani likes prime numbers very much but she is stuck in a question. She has to find four positive integers a, b, c, d such that
$a^3 + b^3 + c^3 + d^3 = N$

Where N and a are Odd numbers and b, c, d are Prime numbers such that $a < b \leq c \leq d$.

She asked Yash to solve it. But Yash is unable to solve it. Being his friend, you help him solve the problem.

Input :
The first and the only line contains a single integer N.

Output :
Print a single line containing four positive integers a, b, c, d satisfying above conditions.If there are multiple answers print any of them.If there is no such answer print -1.

Constraints :
$1 \leq N \leq 10^{18}$

Sample Input:
161

Sample Output:
1 2 3 5

ALGORITHM:

Sieve Technique (to find prime) : Creating a bool array to mark the multiples of prime numbers as false. Insert the primes in a vector and their corresponding cubes in another vector.

Problem:
As it is mentioned, N is an odd number and **a** is odd and **b, c, d** are prime numbers,
Sum of four odd numbers will always be even. So out of b, c, d, one should be even and the only even prime number (also having its cube as even) is 2.
Now it is also given a, b, c, d are in increasing order and a ≠ b and a<b; so a must be 1.
Hence, **a = 1, b = 2** for the given conditions.
Iterate over the cubes vector and for every element find the **N - i^3** in the vector (using binary search) .

Time Complexity(for Sieve Function) : O(N * log(logN))
Space Complexity : O(N) + O(n) + O (n)
Time Complexity : O(N) + (N * log(N))
N = 10^6 , n = number of primes less than N

CODE:

```
#include<bits/stdc++.h>
using namespace std;
#define ll unsigned long long
#define endl "\n"

vector<ll> primeNum;
vector<ll> cube;

ll BS (vector<ll>arr, ll low, ll high, ll ele)
    {
        while(low <= high)
        {
            ll mid = low + (high - low)/2;
            if(arr[mid] == ele)
                return mid;
            else if(arr[mid] > ele)
                high = mid - 1;
            else
                low = mid + 1;
        }
        return -1;
    }
```

```cpp

void Sieve(ll prime[], ll n)
{
    for (ll p = 2; p <= n; p++)
    {
        if (prime[p] == true)
        {
            primeNum.push_back(p);
            cube.push_back( p * p * p);
            if(p < 1001)
            {
                for (int i = p * p; i <= n; i += p)
                    prime[i] = false;
            }
        }
    }
}


int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);

    ll N; cin>>N;
    if(N < 25)
    {
        cout<<-1<<endl;
        return 0;
    }
    int a = 1, b = 2, c = 2, d = 2;
    N = N - 9;

    ll prime[1000000];
    for(ll i = 0; i< 1000000; i++)
        prime[i] = true;

    Sieve(prime, 1000000);

    int flag = 0;
    for(ll i = 0; i<= cube.size(); i++)
    {
        if(cube[i] > N)
            break;

        ll ind = BS(cube, i, cube.size() - 1, N - cube[i]);
        if(ind != -1)
        {
            c = primeNum[i];
            d = primeNum[ind];
            flag++;
            break;
        }
    }

    if(flag == 1)
        cout<<a<<" "<<b<<" "<<c<<" "<<d<<endl;
    else
        cout<<-1<<endl;

    return 0;
}
```

QUES LINK : Prime Cubes

PRACTICE QUE:

1. The soap Mystery
2. Sumit and chocolates
3. Picu Bank