Bài 1: Search Insert Position (easy)

https://leetcode.com/problems/search-insert-position/

Cho 1 mảng được sắp xếp tăng dần chưa các số nguyên không trùng nhau. Cho 1 một số nguyên K, yêu cầu nếu mảng chứa số nguyên K thì trả về vị trí của số nguyên K trong mảng, nếu không có số K yêu cầu trả về vị trí để chèn số nguyên K vào trong dãy.

Ví dụ: A = [1, 3, 5, 6], K = 2

Vì số 2 không tồn tại trong mảng A ban đầu nên cần tìm vị trí để thêm số 2 vào trong mảng A, là vị trí thứ 2 trong mảng.

Problem analysis using brute-force:

- How to generate candidate and how to check candidate: Duyệt từ đầu mảng, tìm phần tử đầu tiên bằng hoặc lớn hơn K thì trả về vị trí của phần tử đó.
- Memory space, execution time and data structure:
  - Memory space: O(N)
  - Execution time: O(N)
  - Data structure: Array

Brute-force code:

```
int Solve(vector<int>& nums, int target) {
    int size = nums.size();
    for (int i=0; i<size; i++) {
       if (nums[i] >= target) return i;
    }
    return size;
}
```

Improvement analysis by using binary search or divide and conquer algorithm:

- Theoretical basis to apply binary search or divide and conquer: Nhận xét mảng ban đầu là mảng tăng dần, nếu phần tử hiện tại lớn hơn số K cần tìm thì có thể loại bỏ phần mảng bên phải của phần tử đó, ngược lại nếu nhỏ hơn thì loại bỏ không tìm kiếm trên phần mảng bên trái. Từ nhận xét trên, ta có thể áp dụng binary search để tối ưu giải thuật của chương trình.
- Memory space, execution time and data structure.
  - Memory space: O(N)
  - Execution time: O(log(N))
  - Data structure: Array
- Code

```
int Solve(vector<int>& nums, int target) {
```

```
        int l = 0, r = nums.size();
        while (l < r) {
            int m = (r+l)/2;
            if (nums[m] == target) return m;
            if (nums[m] > target) r = m;
            else l = m+1;
          }
          return l;
    }
```

Bài 2: ABCDEF (medium)

https://www.spoj.com/problems/ABCDEF/

1. Tóm tắt đề bài

   Cho 1 mảng gồm N (1<=N<=100) các số nguyên, mỗi số nằm trong khoảng từ -30000 đến 30000.
   In ra màn hình số cách chọn bộ 6 số a,b,c,d,e,f từ mảng trên thỏa mãn d!= 0 và:

$$\frac{a*b+c}{d} - e = f$$

2. Bruteforce-01

   2.1. Nhận xét

   - Các số được chọn có thể trùng nhau, một số có thể được chọn nhiều lần.
   - Số các cách chọn ra 6 số bất kỳ từ N số là: N^6 => Không gian tìm kiếm ~ 100^6 = 10^12
   - Với mỗi lần thử, ta kiểm tra xem đẳng thức trên có thỏa mãn không.

   2.2. Liệt kê

```
for(int i1=0;i1<N;i1++){
        for(int i2=0;i2<N;i2++){
                for(int i3=0;i3<N;i3++){
                        for(int i4=0;i4<N;i4++){
                                if(A[i4] == 0){
                                        continue;
                                }
                                for(int i5=0;i5<N;i5++){
                                        long long s = (A[i1]*A[i2]+A[i3])/A[i4]-A[i5];
                                        for(int i6=0;i6<N;i6++){
                                                if(s == A[i6]){
                                                        ans++;
                                                }
                                        }
                                }
```

```
                    }
                }
            }
        }
```

3. Bruteforce-02 + Binary search

3.1. Nhận xét

- Thuật toán với không gian tìm kiếm 10^12 như 2.1 sẽ nhận được Time Limit Exception.
- Sắp xếp lại không gian tìm kiếm với nhận xét sau:
  Đẳng thức ban đầu tương đương:

$$a*b + c = (f+e)*d$$

- Bằng việc liệt kê ra 3 số bất kỳ và lưu lại các giá trị ở vế trái và vế phải, ta có thể tìm được số các cách chọn chỉ với không gian tìm kiếm N^3 ~ 100^3 = 10^6.
- Ví dụ ta có 2 mảng S1, S2 tương ứng với các giá trị của vế trái và vế phải như dưới đây:
  S1[] = {1,1,3,4,2,5,5,4,2};
  S2[] = {1,3,5,6,6,2,4,7,7};
- Ta có thể duyệt qua từng phần tử trong mảng S1, sau đó tìm kiếm số lần xuất hiện của phần tử đó trong S2. Số cách chọn chính là tổng kết quả của các lần tìm kiếm này
  - ⇨ Không gian tìm kiếm: M^2 với M là kích thước của S1.
  - ⇨ Độ phức tạp thuật toán: M^2
- Tuy nhiên, ta để ý thấy nếu sắp xếp mảng S1 và S2 tăng dần, khi đó, phần tử s1 trong mảng S1 nếu tồn tại trong S2, số lần xuất hiện của nó bằng "vị trí xuất hiện cuối cùng" – "vị trí xuất hiện đầu tiên" + 1.
  - ⇨ Không gian tìm kiếm: M^2
  - ⇨ Độ phức tạp thuật toán: M*Log(M)

3.2. Liệt kê

```
for(int i = 0; i<N;i++){
    for(int j=0;j<N;j++){
        for(int k=0;k<N;k++){
            v1.push_back(A[i]*A[j]+A[k]);
            if(A[k]!=0){
                v2.push_back((A[i]+A[j])*A[k]);
            }
        }
    }
}
```

3.3. Kiểm tra

```
sort(v1.begin(),v1.end());
sort(v2.begin(),v2.end());
int ans=0;
for(int i=0;i<v1.size();i++){
    auto  lo = lower_bound(v2.begin(),v2.end(),v1[i]); //Vị trí xuất hiện đầu tiên
```

```
        auto  hi = upper_bound(v2.begin(),v2.end(),v1[i]); //Vị trí xuất hiện cuối cùng
        ans+=hi-lo;
    }
```

Bài 3: Difficulty of an array (medium)

https://codingcompetitions.withgoogle.com/kickstart/round/000000000019ffc7/00000000001d3f5b

Cho một mảng không giảm bất kì có độ dài N (2 ≤ N ≤10^5). Định nghĩa "độ khó" D của 1 mảng A là giá trị lớn nhất của hiệu 2 phần tử liên tiếp trong mảng đó: D = max(A[i+1] - A[i]) với 1 ≤ i ≤ N. Cho số K (1 ≤ K ≤ 10^5), yêu cầu với tối đa K lần thêm 1 phần tử vào trong mảng, tìm ra D sao cho D là nhỏ nhất.

Ví dụ: A = [100, 200, 230], K = 1.

Mảng A hiện tại có D = 200 – 100 = 100. Thêm phần tử 150 vào mảng A:

A = [100, 150, 200, 230], khi đó D = 150 – 100 = 50.

Problem analysis using brute-force:

- How to generate candidate and how to check candidate: Đề bài yêu cầu đưa ra D nhỏ nhất sau khi chèn thêm K phần tử mới vào mảng, do đó ta sẽ dử dụng phương pháp brute-force tìm kiếm tất cả các giá trị D thỏa mãn điều kiện chèn thêm tối đa K phần tử vào mảng, sau đó đưa ra giá trị D nhỏ nhất.
- Memory space, execution time and data structure:
    - Memory space: O(N)
    - Execution time: O(N*10^9)
    - Data structure: Array

Brute-force code: Nhận xét 1 ≤ D ≤ A[N-1], ta sẽ giảm dần D từ giá trị lớn nhất có thể đến 1, kiểm tra giá trị D hiện tại thỏa mãn yêu cầu đề bài hay không, nếu thỏa mãn thì tiếp tục giảm D, ngược lại thì ta sẽ thoát khỏi vòng lặp và đưa ra D nhỏ nhất

```
bool check(vector<int>& a, int k, int m) {
    int temp = 0;
    for (int i=1; i<a.size(); ++i) {
        temp += (a[i] - a[i-1]-1)/m;
    }
    if (temp > k) return false;
    return true;
```

```
}

int Solve(vector<int>& a, int k) {
        int ret = INT_MAX;
        for (int i = a[a.size()-1]; i>0; --i) {
                if (check(a, k , i)) {
                        ret = i;
                }
                else break;
        }
        return ret;
}
```

Improvement analysis by using binary search or divide and conquer algorithm:

- Theoretical basis to apply binary search or divide and conquer: Xét thấy đây là bài toán tìm kiếm giá trị D nhỏ nhất thỏa mã điều kiện cho trước (bài toán search) với $1 \leq D \leq A[N-1]$. Sử dụng phương pháp binary search để giảm thiểu thời gian tính toán của chương trình
- Memory space, execution time and data structure.
    - Memory space: O(N)
    - Execution time: O(N*log(10^9))
    - Data structure: Array
- Code

```
bool check(vector<int>& a, int k, int m) {
    int temp = 0;
    for (int i=1; i<a.size(); ++i) {
        temp += (a[i] - a[i-1]-1)/m;
    }
    if (temp > k) return false;
    return true;
}

int Solve(vector<int>& a, int k) {
```

```
    int l = 0, r = a[a.size()-1]-a[0];
    while (l < r-1) {
        int m = (r+l)/2;
        if (check(a,k,m)) r = m;
        else l = m;
    }
    return r;
}
```

Bài 4: Socially Distancing - Problem D, LG code jam Online Round 1 (medium)

https://codejam.lge.com/contest/problem/609/4

Cho n địa điểm(cửa hàng) trên 1 trục đường thẳng gọi x [1], x [2], ..., x [n] là vị trí của các cửa hàng . Cho số s (<=n) cửa hàng sao cho khoảng cách giữa 2 cửa hàng gần nhất có giá trị lớn nhất.

- 2 ≤ s ≤ n ≤ 200,000
- 1 ≤ x[i] ≤ 1,000,000,000

Input: Nhập n,s và x[i].

Output: In ra khoảng cách D(giữa 2 địa điểm gần nhất trong s địa điểm) có giá trị lớn nhất

Solution

Let's define the distance of two closest outlets as D.

It is easily think that if we increase D more and more to maximize its number, we may not select s locations of electronic outlets at some moment.

From this intuition, the final answer D can be obtained just before the moment we could not select s locations of electronic outlets.

So, to find answer, we need to keep inspect whether we can select s locations with current D.

Brute Force

We can search maximum D from the smallest(=1) till the given condition does not meet.

This can be done in O(max{x[i]}) time complexity.

However, as x[i] can be large as 1,000,000,000, this method can not make solution within time limit boundary.

Binary Search

We can select s locations of outlets with the smallest D(=1) initially, but we can not select s outlets anymore after meet some boundary value of D.

Which means, the condition whether we can select s outlets or not is dependent with D values and it is monotonic(True → False).

So, in this case we can apply binary search for the D.

From this method, time complexity is $O(N \log_2(\max\{x[i]\}))$ and can make solution for this problem.

```
#include <cstdio>
#include <algorithm>

#include <vector>

using namespace std;

int n, s;
vector<int> num_list;

bool check(int mid)
{
 int count = 1;
 int curr = num_list[0];

 for (int v : num_list)
 {
  if (v >= curr + mid)
  {
   curr = v;

   count++;
   if (count == s)
   {
    return true;
   }
  }
 }

 return false;
}

int binary_search(int lo, int hi)
{
```

```
 int result = lo;
 while (hi - lo >= 2)
 {
  int mid = (lo + hi) / 2;

  if (!check(mid))
  {
   hi = mid;
  }
  else
  {
   lo = mid;
  }
 }

 for (int x = hi; x >= lo; x--)
 {
  if (check(x))
  {
   result = x;
   break;
  }
 }

 return result;
}

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

 while (T--)
 {
  scanf("%d %d", &n, &s);

  num_list.assign(n, 0);

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

```
  sort(num_list.begin(), num_list.end());

  printf("%d\n", binary_search(1, num_list[n - 1] - num_list[0]));
 }

 return 0;
}
```