Algorithms Laboratory (CS29203)
Lab Test - I
Department of CSE, IIT Kharagpur

12th September 2024

Question-1

(25 points)
Recall that a function $g(\cdot)$ is monotonically increasing if $g(x) <= g(y)$ for all values of x and y such that $x <= y$. For example, $g(x) = 2x + 3$ is a monotonically increasing function. Now given a monotonically increasing function $f(x)$ defined on $x > 0$, we aim at finding the **minimum** integer value of x for which $f(x) >= 0$. Of course considering all possible values of x starting from 0 and checking until the value of x for which $f(x)$ gets positive, is a solution yielding a complexity of $O(x)$.

Your task is to solve this problem in **logarithmic time**, *i.e.* in $\log(x)$ time. *Hint: Think of applying binary search to solve the problem. However this is a case where the search space is not bounded since the upper bound is not known in advance. You can think of finding an initial upper bound by using exponential, and then use binary search to find the exact value. Note that exponential search complexity will be logarithmic.*

Example:

```
(Input) f(x) = 3x - 100  (you can hard code the function)

(Output) The value of x = 34 yields f(x) to be positive
```

Question-2

(35 points)
You are participating in a long jump competition as inter-IIT championship league, and you are a part of the IIT Kgp team in the championship. To make it to the final, your team will have to win a number of games. Hence all team members must perform very well. There are games of different jump lengths, and these are defined in an array `jumps[ ]`, where `jump[i]` denotes the jump length of the i-th game. Your team members have trained well and have recorded the maximum distance they have jumped before this event. So it is expected that they can manage to jump the same this time as well. Let this is denoted as an array `members[ ]`, where `members[j]` denotes the distance which the j-th member can jump. Of course in order to win a game, the assigned distance should be less than or equal to the jump capacity of the corresponding member.

Now there is a catch. You can improve your performance if you have an energy drink just before your game. However your team has a limited number of energy drinks available, and these should be used only when it is necessary. The number of available energy drinks is `n` and each energy drink can help you to jump extra `d` distance for a game. So if your previous record of jumping a distance of d' and you take an energy drink, then you can jump a distance of $d' + d$ in a particular game. There is also another constraint that one member can take the drink only once.

For example, consider $jumps[\,] = \{3, 2, 1\}$, $members[\,] = \{0, 3, 3\}$, $n = 1$, $d = 1$. Then the maximum number of games that can be win is 3 because the energy drink can be used in the following way:

- First player takes the drink
- The first player plays the third game $(0 + 1 \geq 1)$
- Second player plays the second game $(3 \geq 2)$
- Third player plays the first game $(3 \geq 3)$

Given the integer arrays `jumps[ ]`, `members[ ]`, and two integers `n`, `d`, your task is to find out **maximum** number of games your team can win. *Hint: think of greedy approach.*

Example 1:

```
(Input) Enter the array jumps[ ]: 5, 4
        Enter the array members[ ]: 0, 0, 0
        Enter the value of n: 1
        Enter the value of d: 5

(Output) Maximum number of games that can be win is 1

Explanation: First player takes the drink.
The first player plays the first game.
```

Example 2:

```
(Input) Enter the array jumps[ ]: 10, 15, 30
        Enter the array members[ ]: 0, 10, 10. 10, 10
        Enter the value of n: 3
        Enter the value of d: 10

(Output) Maximum number of games that can be win is 2

Explanation: First and second player takes the drink.
The first player plays the first game.
The second player plays the second game.
One energy drink is not used.
```

Question-3

Consider an integer array `S[ ]` of size n. Let the minimum and maximum element of `S[ ]` are denoted as x and y respectively. Our goal is to prune the array elements so that the following condition satisfies after pruning:

$$2x > y,$$

with the constraint that pruning (or deletion of elements) can be performed only from the ends of the array. That is, elements can be deleted from the start &/or end of `S[ ]`. The task is to find the **minimum** number of deletions required so that the above condition satisfies in the pruned array.

For example, consider `S = [4,6,1,7,5,9,2]`. Then the minimum number of deletions is 4, where the pruned array is `[7,5,9]`, so that $x = 5$ and $y = 9$, and the condition $2 \times 5 > 9$ is satisfied.

(a) (20 points) Solve and implement the problem using **recursion**. Of course the complexity will be exponential.

(b) (20 points) Improve the solution by implementing **dynamic programming** approach.

Example:

```
(Input) Enter the number of elements: 5
        Enter the array S[ ]: 4  2  6  4  9

(Output) Minimum number of deletions is 3
```