7/22/2020 - Binary Search

Tuesday, July 7, 2020 7:52 PM

"strict" definition of a binary search is:

Given a sorted array, and a target value, determine if the target value is in the array.

If a function f is either increasing or decreasing, and it's easy to calculate forwards, but hard to calculate backwards, we can use binary search to calculate backwards.

Backwards calculation is: I give you f(x), tell me which x created it.

f(x) = 2x+ 4

This one is easy forwards and backwards…

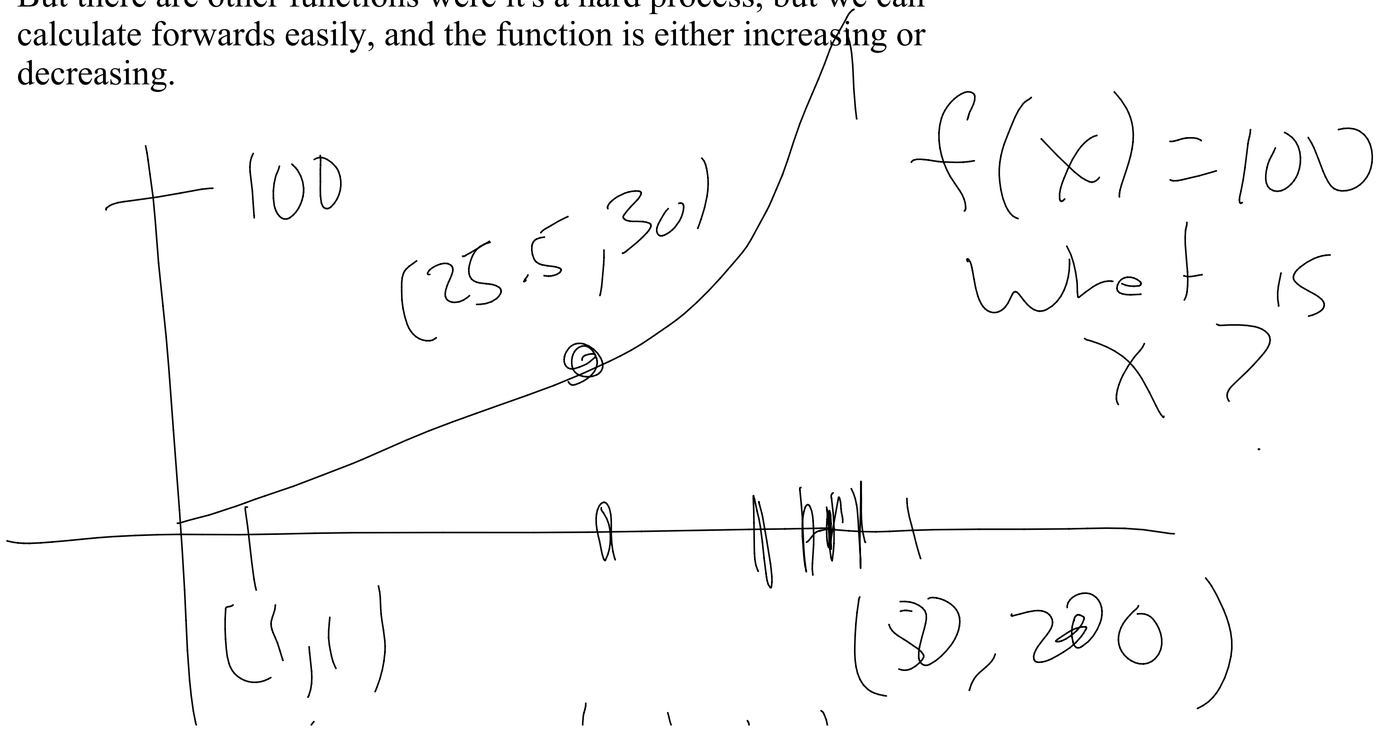
f(3) = 2 x 3 + 4 = 10

If I tell you the answer is 14

2x + 4 = 14
2x = 10
x = 5

So this process is easy for this specific function…

But there are other functions were it's a hard process, but we can calculate forwards easily, and the function is either increasing or decreasing.



$[1, 11)$ $(0, 400)$

low = 1, high = 50

mid = 25.5

low = 25.5 high = 50

after 50 iterations

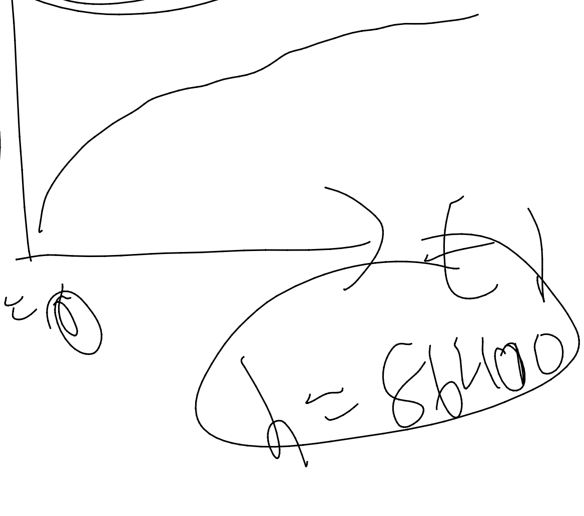
high − low < $\underline{10^{-6}}$

$$\frac{f_2 - f_1}{f_1 f_2} = at + b(1 - e^{-ct})$$

high

$$\frac{D}{a} = t$$

low ≈ 0



Real Valued Binary Searches

1. Now, I write these with a for loop either 50 or 100 iterations. (Reason float pt error with the comparison may yield an infinite while loop if you keep on going until some fine precision point.)
   (The while loop way would be something like while (high-low>1e-6)
2. mid = (low+high)/2 using real number division
3. compare f(mid) to target, afterwards setting either low=mid or high=mid

4. Make sure that low is below the real answer and high is above the real answer
5. Make sure that the initial value of low and high don't cause any weird problems when plugged into the function. (Problems could be things like integer overflow, sqrt of negative, arccos of a value greater than 1, etc.)

Careful Approach Summary

Several planes landing on the same runway. (At most 8 planes.)
Each plane has some range of time it can land based on its flight path, fuel etc.

P1	5 - 15
P2	20 - 50
P3	10 - 25
P4	30 - 40
P5	25 - 45

Goal is to maximize the smallest gap between landing times.

Here is a possible schedule

P1 5
P3 15
P2 25
P4 35
P5 45

Gaps are 10, 10 10 and 10. The smallest of these is 10.

P1 5
P2 22
P3 25
P4 38
P5 45

Gaps are 17, 3, 13, and 7. The smallest of these is 3. This is worse than the first schedule.

Greedy approaches tend to run into problems here.

Maybe, let's approach an easier question…

Let's say we were told which order to land the planes. So let's say someone told us You must land them in this order:

P1, P3, P2, P5 and P4

P1	5 - 15
P3	10 - 25
P2	20 - 50
P5	25 - 45
P4	30 - 40

Easy to see we want to land P1 at 5.
Still hard to know exactly when to land P3…hard to know whether to err to the end of the time range or the beginning?

What if I simply asked: Is it possible to land the planes in this order with a gap of 10 minutes?

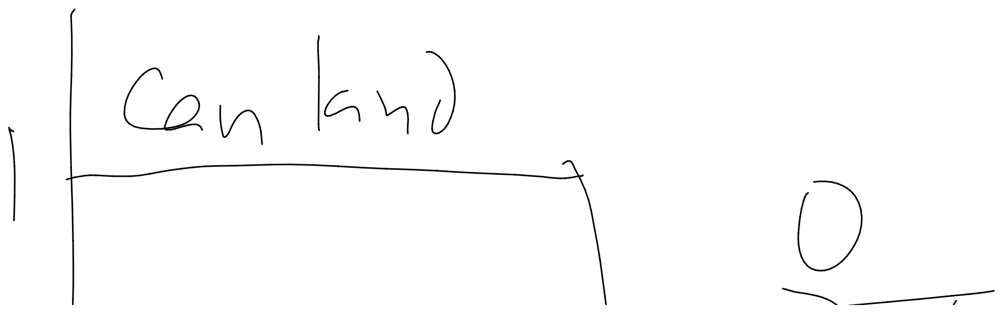
P1	5
P3	15
P2	25
P5	35
P4	x (can't do it)

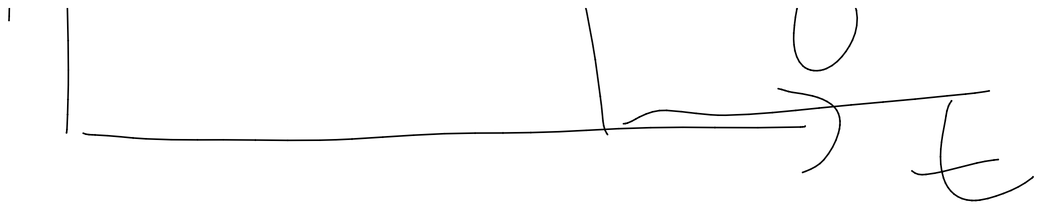
We CAN answer this question with a greedy algorithm.

1. Land the first plane at the earliest possible time
2. For each subsequent plane, add the gap requirement to the previous landing time and see if it is in range. If so, this is when you land the plane. If this time is too late (past the window), we know the task is impossible. If this time is too early, then wait until the plane's time window opens and then land it.

P1, P4, P2, P5, P3, try gap = 1
P1 = 5
P4 = see that 5+1 = 6 6, and the window for 6 isn't open yet, so we would have to land P4 at time 30
P2 = 31
P5 = 32
P3 = x (33 is too late)

Last observation: If we can land the planes with a gap of t minutes, we can always land the planes with a gap less than t minutes. If we can NOT land the planes with a gap of t minutes, then we CAN NOT land the planes with a gap greater than t minutes…THIS IS A DECREASING FUNCTION!!!





We can binary search this (boolean) function

If we knew the landing order, then we could binary search the function…

But there are ONLY 8 planes!!! So, we can just try all the landing orders and find the best answer.

P0	5 - 15	(second)
P1	20 - 50	(fifth)
P2	10 - 25	(third)
P3	30 - 40	(fourth)
P4	25 - 45	(first)

perm = 4, 0, 2, 3, 1

planes[perm[i]]

Problems to code live:

1. Need for Speed (kattis)
2. Incremental House of Pancakes (Code Jam Round 2 2020)

Need for Speed Sample 1:

3 5 (3 segments of driving, 5 units of time total)
4 -1 (distance 4, speed showing = -1)
4 0 (distance 4, speed showing = 0)
10 3 (distance 10, speed showing = 3)

Answer turns out to be 3, which we can verify:

d=4, s=2
d=4, s= 3
d=10, d= 6

Seg 1 = 4/2
Seg 2 = 4/3
Seg 3 = 10/6

2 + 4/3 + 5/3 = 5 (voila!)

We can guess what c is. If we guess too low, then the calculated time would be too big. If we guess too high, the calculated time will be too low.

For this sample, guess c = 2:

d=4, s=1
d=4, s=2
d=10, s=5
time = 4/1 + 4/2 + 10/5 = 4 + 2 + 2 = 8 (too big)

This means our guess of c = 2 was too small.

low = 1 (because one of the listed speeds was -1), in general low is -min.
high = 10000000 (is safe)

Incremental House of Pancakes

L, R up to 10^{18} pancakes…
we grab 1 then 2 then 3, etc.

Keep in mind that $1 + 2 + 3 \ldots + 1.8 \times 10^9 > 2 \times 10^{18}$

sum of the first n ints is n(n+1)/2…

One stack might be way bigger than another, so step 1…eat a bunch of pancakes so the two stacks are close.

After they are close, we alternate…

Issues with integer binary searches
----
1. sometime mid = (low+high)/2, other times we want mid=(low+high+1)/2. It's problem specific.
2. depending on how you characterize the problem, we could set low = mid or low = mid+1. We could set high = mid or high = mid-1. You have to be SUPER CAREFUL FOR BOTH DECISIONS.
3. Also, make sure that you set low when you are supposed to and high when you are supposed (people flip these sometimes)

Close case…

100	104	steps = 15

Now binary search again.

arithmetic sequence a1 = cur, numVals, skip = 2
sum = (a1 + an)/2*n
an = cur + (nV-1)*2
sum = (cur + cur + (n-1)*2)/2*n
        = (2 * (cur + n-1))/2*n
        = (cur +n-1)*n

arith seq a1 = cur+1 skip 2
sum = (cur+1+cur+1+(n-1)*2)/2*n
= (cur+1+n-1)*n