Binary Search

Problem

idx	0	1	2	3	4	5	6	7	8	9	10	11	12	13
value	1	5	7	10	17	29	35	47	49	51	53	107	125	137

A sorted array of numbers without repetition

X

You are given a number X, you have to say after which position X will be put in the sorted array ?

Problem

idx	0	1	2	3	4	5	6	7	8	9	10	11	12	13
value	1	5	7	10	17	29	35	47	49	51	53	107	125	137

X=48

X=12

A sorted array of numbers

X

You are given a number X, you have to say after which position X will be put in the sorted array ?

How we can solve

- A Linear Search: Iterate one by one and find the position

idx	0	1	2	3	4	5	6	7	8	9	10	11	12	13
value	1	5	7	10	17	29	35	47	49	51	53	107	125	137

X	How many index need to be traversed
6	2
36	7
140	14
108	12

In worst cases, we have to traverse |N| element each time.
|N| = array size

What if we were asked about X multiple times ? ≈ |N| * |N| * ....

Can we solve this faster

- Yes ! Binary Search !

Binary = 2 ⟷ Binary search is a searching technique, where we divide our problem's search space into **2 separate parts** and then search our solution.

Applying Binary Search

idx	0	1	2	3	4	5	6	7	8	9	10	11	12	13
value	1	5	7	10	17	29	35	47	49	51	53	107	125	137

mid = floor((st+en)/2)
M = array[mid]

Algorithm
1. Find Middle element (M) of search space
2. If x < M, we will search in left subarray/sub space [st,en=mid-1]
3. Else, we will search in right subarray/sub space [st=mid, en]
4. If remained with two numbers just normally search

Applying Binary Search(step: 1)

X=52

idx	0	1	2	3	4	5	6	7	8	9	10	11	12	13
value	1	5	7	10	17	29	35	47	49	51	53	107	125	137

st=0, en=13,
mid=(st+en)/2 = 6 =>
M=Array[6] = 35, X=52 $\nless$35,
st=mid=6, en=13

6	7	8	9	10	11	12	13
35	47	49	51	53	107	125	137

Applying Binary Search (step: 2)

X=52

6	7	8	9	10	11	12	13
35	47	49	51	53	107	125	137

st=6, en=13,
mid=(st+en)/2 = 9 =>
M=Array[9] = 51, X=52 $\nless$ 51,
st=mid=9, en=13

9	10	11	12	13
51	53	107	125	137

Applying Binary Search (step: 3)

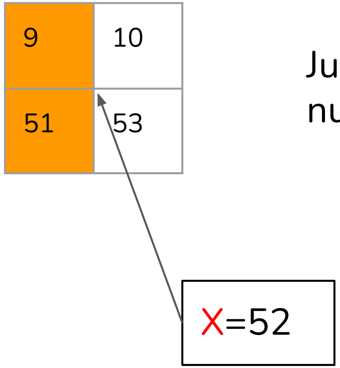
X=52

9	10	11	12	13
51	53	107	125	137

9	10
51	53

st=9, en=13,
mid=(st+en)/2 = 11 =>
M=Array[11] = 107, X=52 < 107
st=9, en=mid-1=10

Applying Binary Search (step: 4)



Just normally search between these two numbers and locate position.

Binary Search: Complexity

X	Linear Search	Binary Search
6	2	3
36	7	3
140	14	5
108	12	4

If we are asked Q times, then in worst cases,
$Q * N >= Q * \log_2(|N|)$
$=> Q * 13 >= Q * \log_2(13)$
$=> Q * 13 >= Q * 3^+$

m1<-[low, high]

m2<-[low, m1-1]

m3<-[m1,high]

Big O Complexity: $O(\log_2(N))$

In each step, we are nullifying half(n/2), and a number can be halved $\log_2(N)$ times.

N, N/2, N/4, ..., $N/\log_2(N)$

Binary Search: Properties

- Falls under the umbrella of **Divide and conquer** strategy
- Problem's characteristics need to support **monotonically increasing/ decreasing property** -> otherwise we can not discard half search space, E.g, unsorted array

Binary Search: Issues

- Partition System: [low, mid] [mid+1, high] vs [low, mid-1][mid, high]
  - Depends on problem specification how we need to divide
  - The intuition of searching space remains same
- Corner Case of Adjacent Indexes, high = low+1

```
While (low <= high) {
        If (low == high) {
                .....
                break
        }
        Mid = (low+high)/2
}
[low, mid-1] [mid+1, high]
```

```
While (low <= high) {
        If (low == high) {
                .....
                break
        }
        Mid = (low+high+1)/2
}
[low, mid-1] [mid+1, high]
```

low = 3, high = 4
mid = (3+4)/2 = 3
[3, 2] **[3, 4] -> Loop**

low = 3, high = 4
mid = (3+4+1)/2 = 4
[3, 3] **[4, 4] -> No Loop**

Binary Search: Floating Point Cases

- Binary search can be used to solve problems having floating point values also. E.g, find (√x)

```
While (low <= high) {
        If (low == high) {
                .....
                break
        }
        Mid = (low+high)/2
}
[low, mid] [mid, high]
```

- $(0+8)/2 = 4; 4 * 4 > 8; [0, 4]$
- $(0+4)/2 = 2; 2 * 2 < 8$ $[2, 4]$
- $(2+4)/2 = 3; 3 * 3 > 8; [2, 3]$

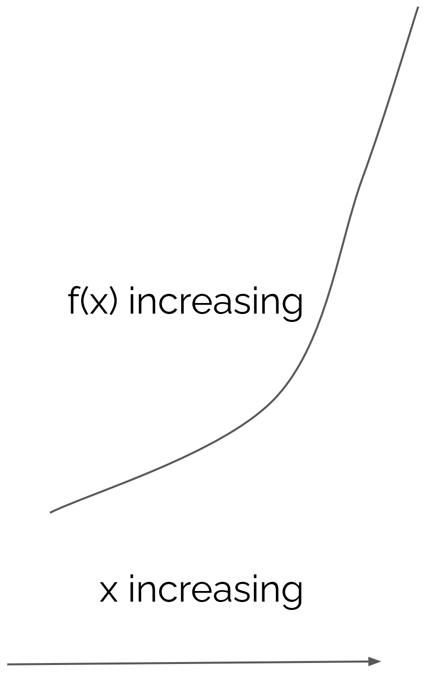
………

What is the stopping condition
- **Desired precision gap:** abs (found ans - desired ans) <= some defined small value. E.g, (mid * mid - result) <= $10^{(-9)}$
- Add precision flexibility with your comparison constraint

Binary Search: Good and Bad Cases

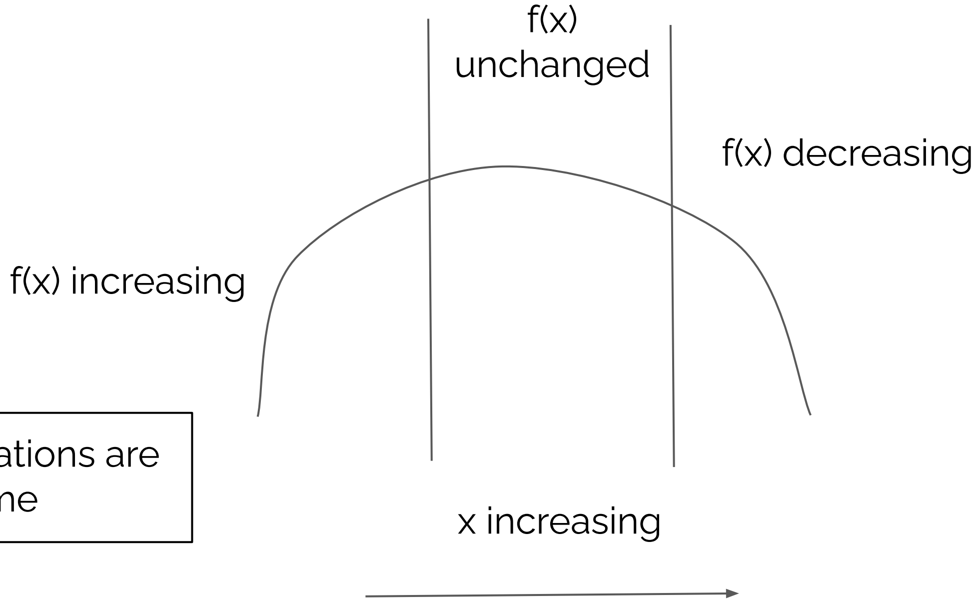
- If the value is not found: Complete search with no results
- The answer lying in the middle will be faster reached compared to the results lying in two sides,
    - The more the answers lying in the side, the more the cost will increase

Ternary Search



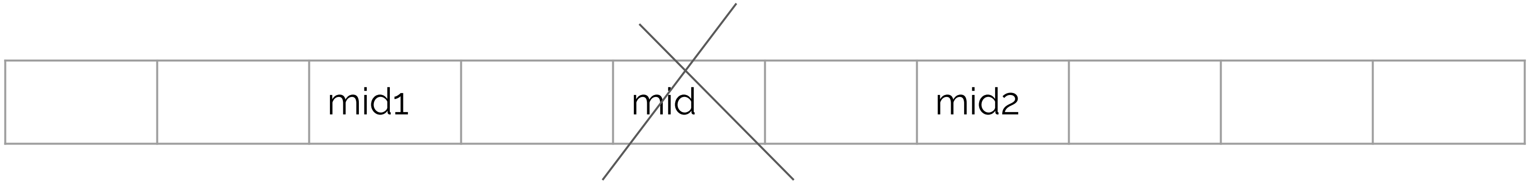
Binary Search Property

Vice Versa situations are also same



Ternary Search Property

Ternary Search



Mid1 = low + (high-low)/3
Mid2 = high - (high-low)/3

Search value >= mid1:
- [mid1, high]

Search value <= mid2:
- [low, mid2]

Ternary Search

idx	0	1	2	3	4	5	6	7	8	9	10	11	12	13
value	1	5	7	10	17	29	35	47	49	51	53	107	125	137

Let's assume, these values denote coordinates of n points. Need to find the point from which the summation of distance of each point gets minimized

Ternary Search

idx	0	1	2	3	4	5	6	7	8	9	10	11	12	13
value	1	5	7	10	17	29	35	47	49	51	53	107	125	137
distance	659	611	..	..	..	..	465	....	...	...	...	...	...	...

Mid1 = 0+(13-0)/3 = 4,
Mid2 = 13 - (13-0)/3 = 9

idx	0	1	2	3	4	5	6	7	8	9	10	11	12	13
value	1	5	7	10	17	29	35	47	49	51	53	107	125	137

Thank you