Binary Searching I

By: Ethan Pronev, Andrew Qi Tang

Motivation

Given an array of size N, find the largest element smaller than a given K.

Eg. [1, 3, 2, 6, 7, 4, 3] and K = 5. Ans = 4

Naive Solution

O(N) method.

Loop through the elements one by one and keep track of best so far.

If we have an element better than ours that satisfy the condition update best.

1	2	3	4	5	6	7
1	3	2	6	7	4	3

Best = 0

K = 5

1	2	3	4	5	6	7
1	3	2	6	7	4	3

Best = 1

K = 5

1	2	3	4	5	6	7
1	3	2	6	7	4	3

Best = 3

K = 5

1	2	3	4	5	6	7
1	3	2	6	7	4	3

Best = 3

K = 5

1	2	3	4	5	6	7
1	3	2	6	7	4	3

Best = 3

K = 5

1	2	3	4	5	6	7
1	3	2	6	7	4	3

Best = 3

K = 5

1	2	3	4	5	6	7
1	3	2	6	7	4	3

Best = 4

K = 5

1	2	3	4	5	6	7
1	3	2	6	7	4	3

Best = 4

K = 5

1	2	3	4	5	6	7
1	3	2	6	7	4	3

Best = 4

K = 5

Answer is thus 4

Extensions

What if we have multiple Ks? (Q queries)

O(QN) time

We can do better though

Binary Searching

Divide and Conquer Technique

Only works for sorted array.

Idea is to use this condition in a sorted array. $A_i <= A_{i+1}$ (Non-Decreasing)

Maintain an active subarray where the answer potentially be.

Subarray can be represented with its left endpoint and right endpoint (L and R).

Binary Searching

Check the middle element

We know everything to the left of it is <= A_{mid}

We know everything to the right of it is >= A_{mid}

Divide based on the middle element, where would the potential answer fall

If falls in the left section set R = mid-1, otherwise L = mid+1 for right section

Continue until no such section anymore

Binary Searching for the problem

Check the middle value and update the answer.

Is it <= K?

If <= K, then move to the right. (We want to maximize the answer)

If > K, then move to the left. (Everything to the right of mid is also > K)

Binary Searching Code

```
int l = 1, r = N, ans = 0; //initializing
while(l <= r){ //checking if the range exist
  int mid = (l + r) / 2; //get the middle value
  if(arr[mid] <= K){ //our condition
    ans = arr[mid]; //update answer
    l = mid + 1; //in the right section
  }
  else{
    r = mid - 1; //in the left section
  }
}
```

1	2	3	4	5	6	7
1	2	3	3	4	6	7

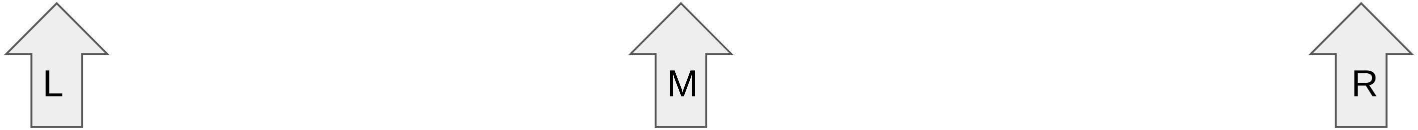
Best = 0

K = 5

L = 1

R = N

1	2	3	4	5	6	7
1	2	3	3	4	6	7



Best = 3

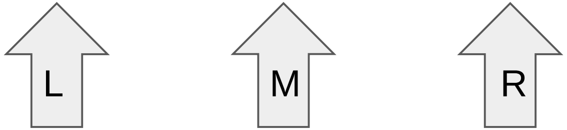
K = 5

L = 1

R = N

Mid = (L+R)/2 = 4

1	2	3	4	5	6	7
1	2	3	3	4	6	7



Best = 3

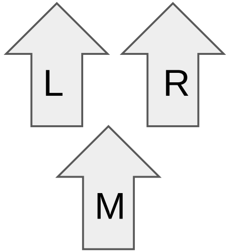
K = 5

L = 1

R = N

Mid = (L+R)/2 = 6

1	2	3	4	5	6	7
1	2	3	3	4	6	7



Best = 4

K = 5

L = 1

R = N

Mid = (L+R)/2 = 5

Binary Search Time Complexity

Every single time the range is being cut in half.

Stops when the subarray is not existent.

Dividing repeatedly by 2 means it’s logarithmic.

O(log N) time

Sorting is O(N log N) time

Doesn’t have to be exactly middle!!!

If (L+R)/2 is not an integer, then that’s ok.

Just use (L+R)/2 integer division as the mid point.

Still works in O(log N) time.

Final Notes

Very useful searching skill.

Can change from $O(N) \rightarrow O(\log N)$ (Significantly faster)

In C++ lower_bound(), upper_bound(), binary_search() are functions.

More application (binary searching for answer), but will not be discussed

Problems

https://dmoj.ca/problem/year2016p3

https://dmoj.ca/problem/seed2

https://dmoj.ca/problem/cpc19c1p2

https://dmoj.ca/problem/tle17c7p3