Binary Search

Pittsford Sutherland High School

by Yiyou Chen, Yizuo Chen

What is a “search” algorithm?

- Search algorithms are used to find particular values including lowerbound, upperbound, maximum and minimum values, or an exact value in an array, stack, graph, or any data structure that saves values.

- In this section, we are going to talk about **normal(slow) ways of searching**, or brute force, and **binary search** on a **linear structure**.

Normal search(slow) --- O(n)

- Given an ordered integer array x[] and a known constant k, we want to find the position i where x[i] = k.
- The normal search can be used when the size of the array is small. For example, size <= 1000000.
- Normal search is easy to write. In this example, we can directly search all the values in x and find the position where k appears.
- Sample code:

```
Loop i: 1 -- sizeof x
    if x[i] = k    return i
```

- Obviously the time complexity is O(n) for normal search.

How can we make it faster?

- The order in normal search is:

1, 2, 3, 4……..

And it seems like we are not able to make this faster.

- However, if the given array of x[] is **ordered**(assumed in increasing order), **Do we have to search in order, too?**

- **NO!!!**

A faster way – binary search

- The idea of binary search can **only be used when** an array or whatever sequence values **is ordered**.
- Let's take increasing order for example. X[1] < x[2] < x[3] <…… and the size of array x is N and the value we are searching is k.
- The binary search says that every time we **only search the medium value** of **an interval**.
- Obviously the first interval where k is located is [1,N], so we will compare k with the medium value of the interval, which is x[N/2]. If k is greater than x[N/2], it means that the interval where k is located is [N/2, N] and we will continue to compare k with x[(N/2+N)/2]; Otherwise, k is located in interval [1, N/2] and we will continue to compare k with x[(1+N/2)/2]…….
- When to end: the left bound and right bound of the interval are the same, [p,p], which means the p is the position where k occurs in the sequence.

How to write it?

```
While (left_bound <= Right_bound){
   midpoint = (left_bound + right_bound) / 2;
   if k > midpoint   left_bound = midpoint + 1
   if k = midpoint return midpoint
   if k < midpoint   right_bound = midpoint - 1
}
```

Time complexity is approximately: O(logn)

Applications of Binary Search

- The idea of Binary search can be used to find the lower and upper bound.

Finding Lower(upper) bound

- When finding the Lower(upper) bound of a number k in an array x[], k does not necessarily occur in the array. We want to find the position the largest number that is less than k occurs in the array.
- Similar to the binary search, we can use midpoints and intervals to solve this:

```
while (left <= right) {
    mid = (left + right)  / 2;
    if x[mid] <= k {
        ans = mid
        left = mid + 1
    }
    else right = mid – 1
}
```

More challenging applications

- Binary search can not only be used on finding a number in an array, but it can also be used when a problem contains an ordered propriety.
- It's sometimes hard to think, but binary search is a really useful tool to these kind of problems.
- For any kind of function f, if f is always increasing(or decreasing) respect to any variable t, then we can always use binary search to find the optimistic value of t in O(logn) time.

Here're two sample problems from Topcoder for practice:

AutoLoan – SRM 258

SortEstimate - SRM 230

Higher dimensional searches

- As you may have noticed, the time complexity will reduce as the dimension of the search increases. If the time complexity for binary search is O(log2 n), the time complexity for 3 dimensional search is O(log3 n), and so on.
- However, it will get harder to code when the dimension gets higher. For example, for 3 dimensional search we want 2 points in the middle of an interval with the endpoints: leftbound, leftbound + (leftbound + rightbound) / 3, rightbound - (leftbound + rightbound) / 3, rightbound. Every time we want to search all 3 intervals to determine which interval k will fit in.