Searching

Topics

1. Searching algorithms [Linear Search, Binary Search & Ternary Search]
2. Problems & Solutions
3. Important Notes
4. Important References

Searching Algorithms

1. **Linear Search**: In computer science, linear search or *sequential search* is a method for finding a *particular value* in a list that *checks each element* in sequence until the desired element is *found* or the list is *exhausted*. The list can have *any order*.

```
for(int i = 0; i < n; i++) {
    if(ar[i] == x)
        return true;
}
return false;
Time Complexity: O(N)
```

2. **Binary Search**: In its simplest form, binary search is used to *quickly* find a value in a *sorted sequence* (consider a sequence of an ordinary array for now). We'll call the sought value the *target value* for clarity. Binary search maintains a *contiguous* subsequence of the starting sequence where the target value is *surely located*. This is called the ***search space***. The search space is initially the *entire sequence*. At each step, the algorithm compares the *median value* in the search space to the target value. Based on the comparison and because the sequence is sorted, it can then *eliminate* half of the search space. By doing this repeatedly, it will eventually be left with a search space consisting of a single element, the target value.

    a. Searching for an element in a given sorted array: [Note: Here, **`ar[]`** is the sorted array, and **`target`** is the element that we're searching for]

```
bool binary_search(int ar[], int n, int target) {
      int lo = 0, hi = n-1, mid;
      while (lo <= hi) {
            mid = lo + (hi-lo)/2;
            if (ar[mid] == target)
                  return true;
            else if (ar[mid] < target)
                  lo = mid+1;
            else hi = mid-1;
      }
      return false;
}
```

Recurrence Relation: $T(n) = T(n/2) + O(1)$

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

b. Given a sorted array **ar[]** and a **target** value, return the *index* of the *first element* in the array, which is greater than or equal to the target value, *i.e.*, find the *ceil* of the given **target** in **ar[]**.
Let us define a function **p(x)** which returns true if **ar[x]** is greater than or equal to the **target** value.

```
int binary_search(int ar[], int n, int target) {
    int lo = 0, hi = n-1, mid, ans = -1;
    while(lo < hi) {
        mid = lo + (hi-lo)/2;
        if(p(mid) == true) {  //or ar[mid] >= x
            ans = mid;
            hi = mid-1;
        }
        else
            lo = mid+1;
    }
    return ans;
}
```

c. Given a sorted array **ar[]** and a **target** value, return the *index* of the last element in the array, which is less than or equal to the target value, *i.e.*, find the *floor* of the given **target** in **ar[]**.
Let us define a function **p(x)** which returns true if **ar[x]** is less than or equal to the **target** value.

```
int binary_search(int ar[], int n, int target) {
    int lo = 0, hi = n-1, mid, ans = -1;
    while (lo < hi) {
        mid = lo + (hi-lo)/2;
        if(p(mid) == true) {  //or ar[mid] <= x
            ans = mid;
            lo = mid+1;
        }
        else
            hi = mid-1;
    }
    return ans;
}
```

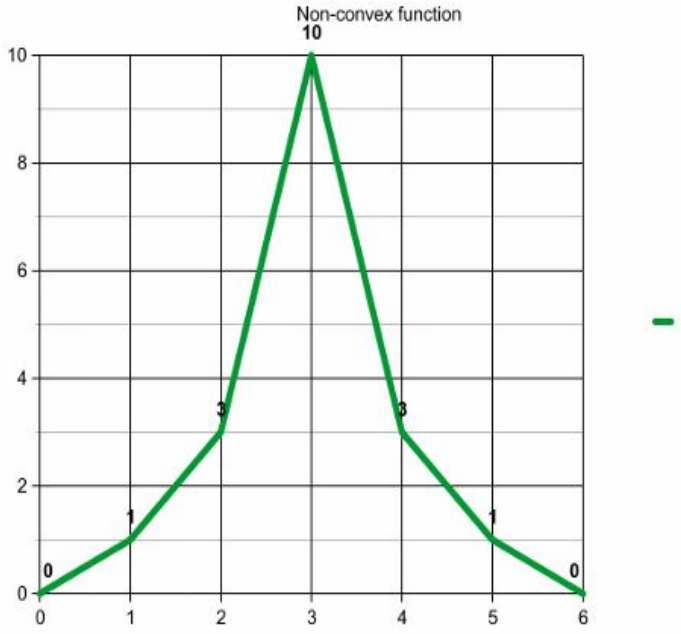
- You may also wonder as to why **mid** is calculated using,
mid = lo + (hi-lo)/2 instead of the usual **mid = (lo+hi)/2**. This is to avoid overflow in certain cases, where lo and hi are towards the max limit of the data type being used.

d. Working with real numbers: Find square root of x, *i.e.*, find $\sqrt[2]{x}$.

```
double square_root(double x) {
    double lo = 0, hi = x, mid;
    double precision = 1e-8;
    while(hi-lo > precision) {
        mid = lo + (hi-lo)/2;
        if(mid*mid > x)
            hi = mid;
        else
            lo = mid;
    }
    return lo;
}
```

3. **Ternary Search:** A ternary search algorithm is a technique in Computer Science for finding the *minimum* or *maximum* value of a **unimodal function** (function that is either strictly increasing and then strictly decreasing or vice versa). A ternary search determines either that the minimum or maximum cannot be in the first third of the domain or that it cannot be in the last third of the domain, then repeats on the remaining two-thirds**.** [**Video Tutorial on Ternary Search**]

   Example: Find the *peak point* in the graph given below.




```
int ternary_search(int ar[], int n) {
    int lo = 0, hi = n-1;
    while(lo<hi) {
        int p1 = (lo*2+hi)/3;
        int p2 = (lo+2*hi)/3;
        if(ar[p1] > ar[p2]) //You can also compare f(p1) and f(p2)
instead, if you are dealing with a unimodal function instead of an array
            hi = p2;
        else
            lo = p1;
    }
}
```

Recurrence Relation: $T(n) = T(2n/3) + O(1)$

Time Complexity: $O(\log_3(N))$

Problems & Solutions

1. WAF to find *floor* value of input '**key**' in a given array, say, **ar = [1, 2, 5, 6, 8, 9, 15, 18, 20]** and **key = 7**, we should return **6** as outcome.
   Solution: Use 2c [explained above].

2. Given an array of sorted integers where each element occurs *twice*, except a single element. WAF to find the element occurring only *once*.
   Solution: Note the indexes of the array, find the pattern and use binary search.
   Refer: **GeeksForGeeks solution**

3. WAF to count the number of occurrences in a sorted array.
   Solution: Use slight modification of 2b & 2c combined.

4. Given an array of sorted integers which represent box sizes and an integer representing an item size, find best fit box for the item, say **ar = [1, 2, 5, 6, 8, 9, 15, 18, 20]** and **size = 7**, we should return **8** as outcome.
   Solution: Use 2b [explained above].

5. WAF to find the square root of a given number, x, *i.e.*, find $\sqrt[2]{x}$.
   Solution: Use 2d [explained above].

6. WAF to search for a value in an $(m \times n)$ matrix. Integers in each *row are sorted* from *left to right* and the *first* integer of each row is greater than or equal to the *last* integer of the previous row.
   Solution:
   a. Binary Search in each *row* or *column* - **$O(N\log_2(M))$ or $O(M\log_2(N))$**
   b. Use 2c on first column and then binary search on that row - **$O(\log_2 N+\log_2 M)$**
   c. Use 2b on last column and then binary search on that row - **$O(\log_2 N+\log_2 M)$**

7. WAF to find the minimum element in a sorted and rotated array.
   Solution: Refer: **GeeksForGeeks Solution**

8. There are **N** *cabinets* arranged in a fixed line, each having **f_i** number of files. *Partition* the line of filing cabinets into **K** *sections* so as to *minimize the maximum number of files* a partition contains. Eg: **10 20 30 40 50 | 60 70 | 80 90 [N = 9, K = 3]**.
   Solution: Define a function **p(x)** which **return**s **true** if you can partition the cabinets in such a way that each partition contains *at max* **x** files. *Binary Search on the answer* using 2b as explained above.

Important Notes

Using `binary_search()` from C++ STL

1. **Documentation for `binary_search()` in C++**
2. **Documentation for `lower_bound()` in C++**
3. **Documentation for `upper_bound()` in C++**
4. **Documentation for `equal_range()` in C++**

Important References

1. **Topcoder Tutorial for Binary Search**
2. **CodeChef Tutorial for Binary Search**
3. Register on **LeetCode** and try some problems from the **Binary Search** section.