Searching

Topics:

1. Searching algorithms
2. Problems & Solutions
3. Notes

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 need not be ordered.

```
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 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:**

```
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;
}
```

Time Complexity: O (log N) [T(n) = T(n/2) + O(1)]

b. **Given an array and a target value, return the index of the first element in the array, which is greater than or equal to the target value.**

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;
   while (lo < hi) {
      mid = lo + (hi-lo)/2;
      if (p(mid) == true)
         hi = mid;
      else
          lo = mid+1;
   }
   if (p(lo) == false)
      return n;            //p(x) is false for all x in the array
   return lo;              // lo is the least x for which p(x) is true
}
```

c. **Given an array and a target value, return the index of the last element in the array, which is less than or equal to the target value.**

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;
   while (lo < hi) {
      mid = lo + (hi-lo+1)/2;
      if (p(mid) == true)
         lo = mid;
      else
         hi = mid-1;
   }
   if (p(lo) == false)
      return -1;            //p(x) is false for all x in the array
   return lo;              // lo is the greatest x for which p(x) is true
}
```

- Please note that we have changed "mid = lo + (hi-lo)/2;" to "mid = lo + (hi-lo+1)/2;". Consider what happens when you run this code on some search space for which the function p gives:

0	1
true	false

The code will get stuck in a loop. It will always select the first element as mid, but then will not move the lower bound because it wants to keep the "no" in its search space. There are other ways of getting around the problem, but this one is possibly the cleanest. Just remember to always test your code on a two-element set where the predicate is false for the first element and true for the second.

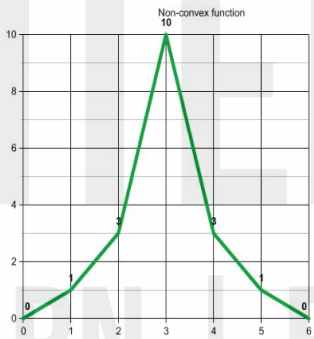
- You may also wonder as to why mid is calculated using mid = lo + (hi-lo)/2 instead of the usual mid = (lo+hi)/2. This is to avoid another potential rounding bug: in the first case, we want the division to always round down, towards the lower bound. But division truncates, so when lo+hi would be negative, it would start rounding towards the higher bound. Coding the calculation this way ensures that the number divided is always positive and hence always rounds as we want it to.

d. **Working with real numbers**: **Find square root of x**

```
double square_root(double x) {
   double lo = 0, hi = x, mid;
   double precision = 1e-9;
   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**.**

Example: Find the peak point in the below graph



```
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])
         hi = p2;
      else
         lo = p1;
   }
}
```

Time Complexity: O (log N) [T(n) = T(2n/3) + O(1)]

Problems

1. WAF to find floor value of input 'key' in a given array, say A = {-1, 2, 3, 5, 6, 8, 9, 10} 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 1. WAF to find the element occurring only once.
   Solution: Note the indexes of the array, find the pattern and use binary search. Something like:
   http://www.geeksforgeeks.org/find-the-element-that-appears-once-in-a-sorted-array/

3. WAF to count the number of occurrences in a sorted array.
   Solution: Use 2b and 2c explained above. Something like:
   http://www.geeksforgeeks.org/count-number-of-occurrences-in-a-sorted-array/

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 A = {1, 2, 3, 5, 6, 8, 9, 10} 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.
   Solution: Use 2d explained above.

6. WAF to search for a value in an m x 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
   b. Use 2c on first column and then use binary search on the row.

7. WAF to find the minimum element in a sorted and rotated array. - http://www.geeksforgeeks.org/find-minimum-element-in-a-sorted-and-rotated-array/

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 returns true if you can partition the cabinets in such a way that each partition contains at least x files. Binary Search on the answer.

Notes for using binary_search from C++ STL

1. http://www.cplusplus.com/reference/algorithm/binary_search/
2. http://www.cplusplus.com/reference/algorithm/lower_bound/
3. http://www.cplusplus.com/reference/algorithm/upper_bound/
4. http://www.cplusplus.com/reference/algorithm/equal_range/

Other Links:

1. https://www.topcoder.com/community/data-science/data-science-tutorials/binary-search/
2. https://www.codechef.com/wiki/tutorial-binary-search
3. Variation of problem6 - http://www.geeksforgeeks.org/search-in-row-wise-and-column-wise-sorted-matrix/
4. Register on spoj and try:
   a. http://www.spoj.com/problems/TEST/ [Sample]
   b. http://www.spoj.com/problems/AGGRCOW/ [Similar to Problem-8]
   c. http://www.spoj.com/problems/SUMFOUR