CS 162
Intro to Programming II
Searching
Searching

• Data is stored in various structures
  – Typically it is organized on the type of data
  – Optimized for retrieval of information

• Searching implies that items can be compared

• Ideal search algorithm is $O(1)$
  – Is it possible?
Searching

• Usually want more than just a boolean result
• We have an element to remove, add or update
• If the item is found, should return the location of the search item
• If not found, should return location where it belongs (if list is ordered)

• How should duplicates be handled?
Linear Search

• Suppose you want to find a number in an unordered sequence

• You have no choice – look through all elements until you have found a match

• This is called linear or sequential search

• Example: find the number 4 in the array

   5  3  2  6  4  1  3  7
Complexity

Best Case-

target in first location  \( O(1) \)

Worst Case-

\( O(n) \)

Average Case-

\( O(n) \)

Linear search is an \( O(n) \) algorithm
int search(int a[], int size, int v)
{
    for( int i = 0; i < size; i++ )
    {
        if( a[i] == v )
            return i;
    }
    return -1;
}
Linear Search

• What if you do a linear search on a sorted array?

• Not much changes
  – Best case: $O(1)$
  – Worst case: $O(n)$
  – Average case: $O(n)$

• It’s still $O(n)$. But there is a better search technique called binary search!
Binary Search

• What if you’re looking for 15 in this array:

1 5 8 9 12 17 20 32

• We’ll look at the first or last half.

1 5 8 9 12 17 20 32

• The number in the middle of the entire sorted array is 9. (Note: we’ll consider the middle to be the last number in the first half).

• 9 < 15, so 15 must be in the second half. We’ll grey out the first half
Binary Search

The number in the middle of the right half is 17. 15 < 17, so 15

The number in the middle of the left half is 12. 12 < 15, so 15 must be in the right half (of the white numbers)
Binary Search

- 17 is obviously not 15. So 15 is not in the array.
- However, if we had to insert 15 into the array, we would put it before 17 at index 5.
- What we just did is called binary search.
  - It’s binary because we cut the size of the search in half at each step.
  - This cutting in half only works because the array is sorted!
int search(int a[], int size, int v) {
    int low = 0;
    int high = size - 1;
    while( low <= high ) {
        int mid = (low + high)/2;
        int diff = a[mid] - v;
        if( diff == 0 ) // a[mid] == v
            return mid;
        else if (diff < 0) // a[mid] < v
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}
Complexity

The size of the search space is reduced by 2 every iteration.
That makes the complexity $O(\log n)$

Best Case- $O(1)$
Worst Case- $O(\log n)$
Average Case- $O(\log n)$
Choosing a Search Method

- Linear search is $O(n)$
- Binary search is $O(\log n)$

- Linear search is easier to code
- Binary search will be MUCH faster on large data
  - But the data must be sorted first
Tradeoff

Is it faster to:

a) Sort an array first then use binary search OR
b) Do a linear search on the unsorted array?

It depends…

If you search only once, then it’s better to do a linear search, which is $O(n)$

If you search many times, then it’s better to pay the price of sorting, which is $O(n \log(n))$ then do as many $O(\log(n))$ binary searches as you have to do