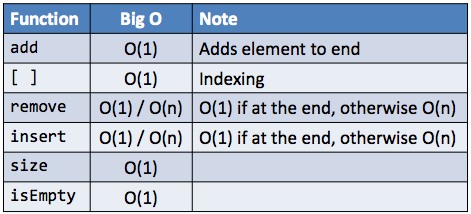
ITP 365

Lecture 10

2/9/17

Search and Algorithmic Analysis

* Searching for a movie in a vector
  + Simplest solution: start at index – and keep incrementing until we find movie we are looking for (Top Gun)
  + **Linear Search**
    - Loop from index 0 to size
      * Check at each index if movie = Top Gun?
      * If so, we found movie and return index
      * If we reach end of vector and we don’t find movie, return -1
    - Linear Search: Worst Case
      * If we want to see if a movie is NOT in a list, we have to go through all elements in the list: n elements
  + Another way: BINARY SEARCH
    - Only works on vectors that are already sorted
    - What if we have an alphabetized (sorted) list
      * If we start at middle of list (Inception) then we check and since Inception is NOT Top gun, we know top gun has to be in 2nd half of list IF it is there.
      * Keep repeating
    - Algorithm:
      * Start with a range of indices
      * Check index at middle of range
      * If contains value we’re looking for, return it!
      * Otherwise, is the value we’re looking for greater than the middle value?
        + If so, cut the range into the first half, and do another binary search
        + Otherwise, cut the range into the second half, and do another binary search
    - Binary search, worst case
      * Log2(n) -> which means 2^x = n
      * For log2(10) we visit roughly 3.3 elements
  + Linear Vs. Binary Search
    - Linear much steeper
    - Worst case: How many steps do you have to check list if you are 100% sure it isn’t in list. For binary search it is log2(n). For linear search it is (n)
  + Big O Notation
    - Convention used to describe the worst-case performance of a particular algorithm
    - Linear search: O(n)
    - Binary search: O(log n)
    - Constant time: if performance is independent of the size of collection
      * O(1)
      * Ex. Accessing an index in a Vector
        + bigVec[0] = 5
    - Random accessing – constant O(1)
    - Insertion or removal of elements at the end – amortized constant O(1)
    - Insertion or removal of elements – linear in distance to the end of the vector O(n)



* Queue and Stack all functions are O(1)
* Set – behind the scenes is in ascending order so is using binary search
  + Add is O(log n)
  + Contains is O (log n)
* Map – behind the scenes sorted with keys in ascending order
  + Put – O(log n)
  + containsKey - O(log n)
  + get - O(log n)