CS20B Assignment 7

Student: Jonathan Reznik

ID: 1654919

Questions

I’m sorry for the somewhat long-winded answers, but I found this topic challenging enough to warrant some further explanations.

1. b...*The shorter code for recursive and the more lengthier usually it the iterative code.*

Caveat: Different ways of programming the iterative solution in code could result in either a better performing or a worse performance based on code implementation. Optimizing compiler would probably only implement iterative that would improve efficiency.

1. a...*The fact that it is unsorted doesn’t matter if the index is known*. Hashing points to the correct location of the item in a O(1) way.

**Question: Why does hashing always O(1), if there are collisions handled using different methods for the collisions with linear probing, quadrative probing etc, and won’t that increase O(1) for things?**

Problem 3 on next page

1. Unsorted: 26, 15, 27, 12, 33, 95, 9, 5, 99, 14

Sorted: 5, 9, 12, 14, 15, 26, 27, 33, 95, 99

Sequential searches from start to end index. **Corrected.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Sequential  search  (unsorted)  O(N) | Sequential  search  (sorted)  O(N) | Binary  search  (sorted)  O(logN) | Binary Search Tree  Big O: O(logN) |
| 15 | 2 | 5 | 1 | 2 |
| 17 | 10 | 6 | 4 | 2 |
| 14 | 10 | 4 | 4 | 4 |
| 5 | 8 | 1 | 3 | 5 |
| 99 | 9 | 10 | 4 | 5 |
| 100 | 10 | 10 | 4 | 5 |
| 0 | 10 | 1 | 4 | 5 |

Problem 4 on final page

1. I have expanded each part a-d in # 5 – 8 below, providing some tracing of comparisons on the simple *{1, 2, 3, 4} Array.*

**Correction: The result of the traces shown below prove that the methods of bubbleSort and**

* 1. With quickSort the first and last indexes move towards eachother, not very efficient for almost sorted list. Steps follow.
     1. **is 2 > 1**, *yes* terminate the first while loop *(next recursive call)*
     2. **is 4 <= 1**, *no* **move last index towards first** *(continue)*
     3. **is 3 <= 1**, *no* **move last index in to first** *(continue)*
     4. **is 2 <= 1**, *no* **move index in one last time** terminate loop #2
     5. ***no* swaps finished**
  2. shortBubble is relatively efficient with data being almost sorted, as is here. Steps follow.
     1. **is 4 < 3**, *no* ***next comparison***
     2. **is 3 < 2**, *no next comparison*
     3. **is 2 < 1**, *no*  function returns **done**
  3. selectionSort is highly inefficient. Steps follow.
     1. **is 2 < 1**, *no*
     2. **is 3 < 1**, *no*
     3. **is 4 < 1**, *no* returns **end first iteration**
     4. **is 3 < 2**, *no*
     5. **is 4 < 2**, *no* returns **end second iteration**
     6. **is 4 < 3**, *no* returns **done**
  4. insertionSort is generally inefficient, but performs well here. Steps follow.
     1. **is 2 < 1**, *no* terminate while loop **end iteration 1**
     2. **is 3 < 2**, *no*
     3. **is 2 < 1**, *no* terminate while loop **end interation 2**

**NOT SURE IF THIS IS GOING TO HAPPEN STILL OR NOT:**

* + 1. **is 4 < 3**, *no*
    2. **is 3 < 2**, *no*
    3. **is 2 < 1**, *no* terminate while loop

**done**