Unit 4, Week 3

Answer these questions on a separate piece of paper.

Binary Search Algorithm

1. Draw a binary search tree showing how to search for a given number in the following array, using the pseudocode specified.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Item | 2 | 4 | 5 | 8 | 10 | 13 | 15 | 22 | 29 | 31 |

def binary\_search(array, target):

    L = len(array)

    if L == 0:

        return -1

    pivot = L // 2

    if array[pivot] == target:

        return pivot

    elif array[pivot] < target:

        result = binary\_search(array[pivot+1:L], target)

        if result == -1:

            return -1

        else:

            return pivot + 1 + result

    else:

        return binary\_search(array[0:pivot], target)

2. What does a return value of -1 signify?

That the item is not in array

3. State which items could take the longest to find using this tree.

The numbers 8, 15 and 31

4. What must be true about the array for all items to take the same amount of time to find?

For the array to be of length 1, because if it has a length longer than this then the algorithm will inherently check one index before another, hence one item will be found before the other.

5. For each of the arrays below, explain why binary search does not work:

(a)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Item | 1 | 2 | 2 | 3 | 5 |

I believe this somewhat works but will only return the value of one of the 2s, not both, since they are duplicated. This means that the search will be able to tell you if the value is there or not in the array, but not the position of all elements.

(b)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Item | 3 | 2 | 4 | 9 | 1 |

The array is unsorted, therefore binary search does not work.

**Sorting Algorithms**

6. Considering MergeSort and QuickSort:

a. Which algorithm requires the most storage space?

Quick Sort seems to be an algorithm that doesn’t require any additional storage space, as it can be just done inside the original array. Meanwhile, Merge Sort would require an additional temporary array to sort the divided arrays, meaning it would require the most space.

b. Which algorithm is the most efficient for **any** given data set? Why?

Frankly, I’m not *too* sure, but I believe Quick Sort will be faster for any given data set, because even though they both have an average case time complexity of O(nlogn), Merge Sort needs to use additional space which might make it slower.

c. Which algorithm chooses a random pivot to divide the data? Why does it do this?

This is something the quick sort does, because in an ideal case it picks a pivot closest to the median of the dataset, but this is slow to calculate so it can just sort them using a random pivot.

d. Which algorithm best describes the following pseudocode:

function sort(array)

    if length(array) ≤ 1

        return array

    pivot = choose pivot element from array

    left = empty array

    right = empty array

    for each element in array

        if element < pivot

            append element to left

        else if element > pivot

            append element to right

        else

            // do nothing, element equal to pivot

    left = sort(left)

    right = sort(right)

    return concatenate(left, pivot, right)

Quick Sort

e. Write a one line description for each sorting algorithm.

Quick Sort: pick a pivot value and then go through the array from both sides while swapping values if they are in the wrong spot.

Merge Sort: keep splitting the array into halves and then reconstruct it by comparing in between each pair/array.