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)

13 (Pivot: 6)

/ \

[2, 4, 5, 8, 10] [15, 22, 29, 31]

(Pivot: 3) (Pivot: 2)

/ \ / \

[2, 4] [5, 8, 10] [15] [22, 29, 31]

(Pivot: 1) (P: 2) (Target) (Pivot: 2)

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

target item is not found in the array

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

Max (31) and min (2)

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

All values have to be same, aka target is either all items (aka the first one it hits after running the algo) or none

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 |

Duplicates

(b)

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

Array isn’t sorted

**Sorting Algorithms**

6. Considering MergeSort and QuickSort:

a. Which algorithm requires the most storage space? MergeSort requires more storage space as it needs additional arrays for merging, whereas QuickSort works in-place

b. Which algorithm is the most efficient for **any** given data set? Why? QuickSort is generally more efficient due to its average-case time complexity of O(nlogn)) and in-place sorting, but its worst-case time complexity is O(n^2). MergeSort guarantees O(nlogn) time complexity but requires extra space

c. Which algorithm chooses a random pivot to divide the data? Why does it do this? QuickSort chooses a random pivot to avoid the worst-case scenario of O(n^2) which can occur when the smallest or largest element is always chosen as the pivot

d. Which algorithm best describes the following pseudocode: The pseudocode describes QuickSort, which selects a pivot and partitions the array into elements less than and greater than the pivot

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)

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

MergeSort: A divide-and-conquer algorithm that recursively splits the array into halves, sorts each half, and merges them back together.

QuickSort: A divide-and-conquer algorithm that selects a pivot, partitions the array around the pivot, and recursively sorts the partitions.