**CSC 180, Lab #8**
**Fall 2021**

***Directions:*** Turn in a hard copy of this assignment, as well as the Python code for question (1) of this lab.

1. Implement the *sequential search* algorithm as a function in Python. The function has a list parameter named *values* and a second parameter named *query*. The function returns *True* if the *query* is in *values*; the function returns *False* otherwise. Your implementation must follow the algorithm in our notes. (Note: This question is given as an exercise, but in practice, if you want to find out whether an element is in a list in Python, you would use the *in* operator. For example, the following code would return True: `3 in [1,2,3,4,5]` [7 points]

2. Consider the list containing the elements: 2, 7, 15, 22, 37, 92, 100, and answer the following questions based on *sequential search*. [10 points]

    a. Complete the table below to indicate the number of comparisons needed to determine whether the given number is in the list.

| Query | Number of comparisons |
|---|---|
| 2 | 1 |
| 7 | |
| 15 | |
| 22 | |
| 37 | |
| 92 | |
| 100 | |
| 40 (or any number not in the list) | |

    b. Find the average number of comparisons from your table above. Note that the average of a set of values is the sum of those values, divided by the total number of values. (Note: your answer to this question is the average number of comparisons needed to check whether an element is in a list of this size, assuming that there is a 50% chance that the element is in the list, and if so, that it is equally likely to be anywhere in the list).

    c. Assume that a list has *n* elements. Find the average number of comparisons needed to find a number that is *in* the list. Recall that the sum of the integers 1 through n is $\frac{n(n+1)}{2}$. If there are 50 numbers in the list, on average how many comparisons need to be made to find a number that *is* in the list?

3. Again consider the following list, which is in sorted order: 2, 7, 15, 22, 37, 92, 100. Answer these questions using *binary search*. [10 points]

    a. Draw the binary search tree that visualizes the comparisons needed for this specific list, following the example in our notes. Note that the first comparison checks the middle element, which is 22.

    b. What comparisons are needed to check if the number 37 is in the list (In other words, what numbers from the list do we need to check)?

    c. What comparisons are needed to check if the number 13 is in the list?

    d. What numbers in the list can be found with only 2 comparisons?

4. Consider the list containing the elements: 1, 2, 3, 4, 5, 6, 7, 8, 9. Answer these questions using *binary search*. [10 points]

    a. Construct a binary search tree that visualizes the comparisons needed for this specific list, following the example in our notes.

    b. What comparisons are needed to check if the number 8 is in the list?

    c. What is the maximum number of comparisons needed to determine if a number is *not* in the list?

5. For question (4), how many comparisons would be needed if *sequential search* was used to search for a number that was not in the list? Which algorithm – sequential search or binary search – requires fewer comparisons? [3 points]

6. The table below specifies the running time for sequential search and binary search, where the algorithms are run on different computers and the computer running sequential search is faster (as a result, the running times are identical when n = 10). Note that the running time of sequential search is $\theta(n)$ while the running time of binary search is $\theta(log_2(n))$. Complete the table below, specifying the running time in minutes and seconds. [6 points]

| ***n*** | ***Sequential search*** | ***Binary search*** |
|---|---|---|
| 10 | 0 minutes and 30 seconds | 0 minutes and 1 second |
| 20 | | |
| 40 | | |
| 80 | | |
| 160 | | |

7. Challenge Problems (these are *optional*)

   a. Implement the binary search algorithm in Python, following the algorithm in our notes. Note: in order to "round down", you can use integer division, e.g. 5 // 2 uses integer division and is equal to 2. Your algorithm should assume that the list is sorted.

   b. For binary search, the order of magnitude in the worst case scenario (when the element is not in the list) is $log_2(n)$. How would you modify the binary search algorithm so that the order of magnitude is <u>constant</u> if the query element is not in the list.