Shazaib Dawood
10/21/2024
CS 211

- Compare the linear search and binary search algorithms, in terms of the different assumptions about the sorted nature of the data, different algorithmic approaches (iterative vs. recursive), different time cost, different big-O time complexity, etc.
    - linear search
        - iterative approach
        - best case: O(1) if element at beginning
        - worst case: O(N) if element is not in the list
        - list doesn't have to be sorted
    - binary search
        - can be iterative or recursive
        - best case: O(1) if element is in the middle
        - worst case: O(log N) if element is not in the list
        - list has to be sorted
- Compare the following sorting algorithms, in terms of the overall algorithmic procedures, the overall algorithmic approach (iterative vs. recursive), time cost using big-O time complexity, etc.
    - selection (min-seek)
        - iterative
        - Loops through every index in the list, finds the minimum element in the unsorted part of the list and swaps it with the element at that current index and then move on to then next index
        - O(N^2)
    - insertion sort
        - iterative
        - loop through every element and insert it into its sorted position
        - Best case: O(N) when list is almost sorted
        - Worst case: O(N^2)
    - bubble sort
        - iterative
        - keep swapping elements that are next to each other if they are in the wrong order
        - O(N^2)
    - mergeSort
        - recursive

        - split the list in half, recursively sort each half, then merge the sorted halves back together
        - O(NlogN)
    - quickSort
        - recursive
        - partition the array into elements greater than a pivot and less than a pivot, then recursively sort each partition
        - Best case: O(NlogN)
        - Worst Case: O(N^2) when the pivot chosen is an extreme of the list
- How to use the time.h library functions to effectively analyze execution time of program components: time(), difftime(), clock(), etc.
    - time()
        - returns the current time
    - difftime()
        - calculates the difference between 2 times
    - clock()
        - returns the processor time used by the program
    - used to measure the execution time of a program by storing the start and end times of the program and then calculating the difference
- How to generate and use pseudo-random numbers in C
    - rand()
        - generates a random number
    - srand()
        - seeds the random number generator
- How to use gprof to effectively analyze execution time of program components: compiling for gprof, program execution with gprof, analyzing timing output from gprof.
    - compiling
        - Gcc -pg main.c
    - shows time spent on each function
- Practice gprof system timing using the Sample zyLab Workspace in Section 13.12 and/or Section 13.13 of the zyBook. Modify the code and view the effect on execution timing. Document your practice and briefly describe the modifications you made and the resulting effects on execution timing. Some ideas include:
    - add computations in the functions level1(), level2(), and level3();
        - Increase in execution times due to the extra operations being performed
    - add an array input to be analyzed (find the sum, max, min, etc.) in functions level1(), level2(), and level3();
        - added more time consumption
    - build an array of random values in the functions level1(), level2(), and level3();
        -

- sort an array in the functions level1(), level2(), and level3();
    -
- have each of the functions level1(), level2(), and level3() do something different.
    -