Tuesday, November 19, 2019

Explanation - project 3

Problem 1 - Finding the square root of an integer

In this problem, I apply the binary search to find the integer whose square value is just less than the given number. Starting from number n, the square root value of halved number (n/2)^2 will be computed and compared against n. If squared value is larger than n, then we continue to halve the n number until the squared value is less than n. The binary search will narrow down the scope until a value m, whose m^2 is less than n and m=n-1. The expected time complexity is O(log(n)). 10 normal tests and 2 edge tests are tested by algorithm and all of them passed the tests.

Problem 2 - Finding the square root of an integer

This problem requires the algorithm that has runtime complexity to be O(log(n)). Binary search that bases on divide and conquer principle are ideal to apply in this problem. Since the given list is pivoted at certain point, the first step is to find the pivot point and divide the given list as two sorted sub-list. Binary search algorithm can then be applied to find the value in one of the sub-list, which has time complexity of O(log(n))

Problem 3 - Rearrange array elements

This problem requires the sorting algorithm that has runtime complexity to be O(n*log(n)). Mergesort, quicksort, and heapsort that bases on divide and conquer principle and have time complexity of O(n*log(n)) are ideal to apply in this problem. I use Mergesort algorithm that learned from lecture, and modify it to sort the list from largest to smallest. Then a for loop is used to traverse the sorted list and assign the value to left or right of two values, based on even and odd index position.

Problem 4 - Dutch National Flag Problem

Given the input list consisting of only 0, 1, and 2, it is required to sort the array in a single traversal. The easiest way is to define three empty lists to store 0, 1, and 2, then traverse the given list from beginning to the end and assign the value to its corresponding list. After the traversal, combine three lists in the order of 0, 1, and 2. The time complexity if O(n)

Problem 5 - Tries

Tries is a tree-like data structure that can be used in text prediction. The time complexity of tries is O(m*log(n)), where m is the length of maximum string length, n is the number of keys in tree. If any string characters that share a same prefix characters, these string values will be stored under the prefix character's dictionary or list. Therefore, by using this principle we can find all the suffixes element by giving a prefix. There are two ways to implement the characters' storage, one is dictionary and another is static array. I use static array in problem 5 and dictionary in problem 7.

By using static array to store string character, a length of 26 corresponding to 26 alphabet characters is created for each node. The index corresponding to 26 alphabet characters can be calculated as ord(str)-ord('a'). By traversing each string character, if the character does not exist under previous prefix, it will be stored as a trie node based on its alphabet character and its value will be stored in another static array. Find method and suffixes method based on recursion can be applied to solve the problem.

Problem 6 - Max and Min in an unsorted array

This problem requires the time complexity of O(n). It means I can traverse the n length of unsorted list only once. Since only the max and min value are needed, it is not necessary to first sort the list and find the values. Instead, we can define a large small variable (1000000) and a tiny large variable (0), and then traverse the unsorted list to compare each element against small and large variable. If the element is smaller than small variable, we let small variable equals to that element. Same principle is applied for large variable as well. By using this method, after one traversal (O(n)), the small and large variable should represents the min and max values of that list.

Problem 7 - HTTPRouter using a Trie

This problem is very similar to problem 5 since both of them base on trie data structure. The difference is that instead of storing a single string character, we store the path component split by "/" in tier. Since I use static array to store the element in problem 5, I try dictionary structure in problem 7 to store the elements. The time complexity is same as the one in problem 5, which is O(m*log(n)), where m is the length of maximum string length, n is the number of keys in tree.