Ashley Chang

504449890

CS32 Homework 4

2. This produces a compilation error because the compiler does not know how to compare URLS, despite the fact that they have private member strings that can be compared. The find function, which is indirectly called by insert, needs a custom comparison operator for URLS themselves to be able to determine where the new item should go.

3b. The one-parameter listAll function cannot be solved as a recursive function because the function would not be able to track and print its path at different levels of the list. It would only be able to print to the end, because it is not storing any string to remember the path.

4a. The time complexity is O(N^2). Beginning at the top of the function, the first step that relies on the size of an input is the for loop, which, because m1 has N distinct items, gives a complexity of O(N). When entering the loop, the get function is called, which itself has a complexity of N (for loops inside if else statement). Continuing in the same loop, the contains function is called, which in turn calls the find function; the find function does a linear search, resulting in N for the contains function. The insert function from the if statement gives O(1). Because the get and contains function are inside the same loop, it results in (N+N), which is performed by the loop N times to result in N(N+N). This thus becomes N(2N) for a final complexity of N^2.

4b. The time complexity is O(NlogN). From the top of the function until the first for loop, the complexity is O(1)+O(1), which can be ignored. When we encounter the two for loops that follow, the complexity is O(N) +O(N), followed by the O(N log N) sort algorithm. At this point, we have 2N+NlogN.

The following for loop has a complexity of O(2N), because it deals with a vector size that is twice the unique size and when entering it, the entirety of its body results in O(1). Adding this to the total complexity makes 4N+NlogN. When the function reaches the point in which it deletes excess result nodes, the if statement contains a do while loop whose complexity is O(N). Adding up the totals makes 5N+NlogN, so the function’s final complexity is O(NlogN).

It is better than the implementation in part A because as the input size grows to infinity, NlogN is more efficient than N^2.

5a.

Declare a counter variable.

Using a for loop to traverse first half of the array,

Use another for loop to traverse the second half of the array

If the value of the item in the outer loop is greater than the value of

the item in the inner loop, add to the count.

5b. To count the number of inversions with an O(NlogN) time algorithm, we need to use an algorithm that will recursively halve the array of numbers until it reaches an array of one element. Once it reaches one element, it merges back with its corresponding other half and compares itself with the other number. For every level, the function must perform some comparisons: If the first value on the left is larger than the first on the right, then the function adds to a count the total number of items on the right and pops the two values that were compared. Otherwise, the value on the left is popped and the value on the right is left alone. It then moves on to the next value in each half that was not popped and performs the same comparison. Once this is completed, it remembers this count, merges back one level, performs the same operations, and adds the counts together.

6. To sort the array in time O(Nlog(logN)), we need to build a binary search tree and insert the values of the array. Since inserting has a complexity of O(logN) for N distinct values, and we have log N distinct values, inserting then has a complexity of O(log(logN)). We need to insert N times for N elements in the array, so this becomes O(Nlog(logN)). To actually sort the items, we travers through the tree with an in-order traversal and process each node, which has a complexity of O(N), but this can be ignored.