1. Introduction:

The Problem is the ability to store a large amount of information and be able to access it quickly, while still maintaining good use of space. There will be many insertions, deletions, and traversing.

The data must be sorted for speed of access and search results must also be sorted in another method

2. Algorithms:

Retrieve in Range:

PsuedoCode

Set a min a max key

Traverse tree inorder with complexity of N

loop

If current node key > min key

Go left

If current node key < max key

Go right

If current node key > min key and current node key < max key

Push to node to heap.

Return heap

Heap function are O(log n)

So the total complexity would be O(N)

Update/Insert Algorithm:

PsuedoCode

Get new data item

If tree Empty

insert data item (0 Log n)

If tree not empty

try find data item with the same key(in order traversal) 0 (log n)

if found O(1)

update data item O (1)

else

insert new data item 0 (log n)

In the case the case of Update / Insert the worst case scenario is if the whole tree is searched and the data item is not found and you still have to insert the item. This brings us to a total of 0 (log n)^2

3. Data Structures:

AVL TREE: the problem require a large amount of insertion, deletion and search which all have a low complexity in an AVL tree.

PRIORITY QUEUE: The results of a search of the data could return large values that needed to be sorted and a heap sort or priority queue used for sorting have a low complexity.

VECTOR: I chose vector for the use of its function, looking back using a string instead wold of reduced complexity.;

4. Complexity analysis: Depending on the level of your code, you would need to

show the complexity of the following operations:

a.

Building a frequency dictionary with use of the update algorithm

Assuming that you also use phrases the Update/insert algorithm is called 3 times for each n word

So 0 (log n)^2 (refer to question 2)

b.

0 (log n)

c. rebuilding would be 0(log n) it is jut a series of insertions

5. Conclusion:

The problem was solved by saving the word frequency in a AVL tree

Using a custom string as a data type for the key

Minimizing insertion and search complexity

Used a priority queue to sort the results of the search in range.

And use a vector to store the list of filenames.