Binary Search Tree

3D5

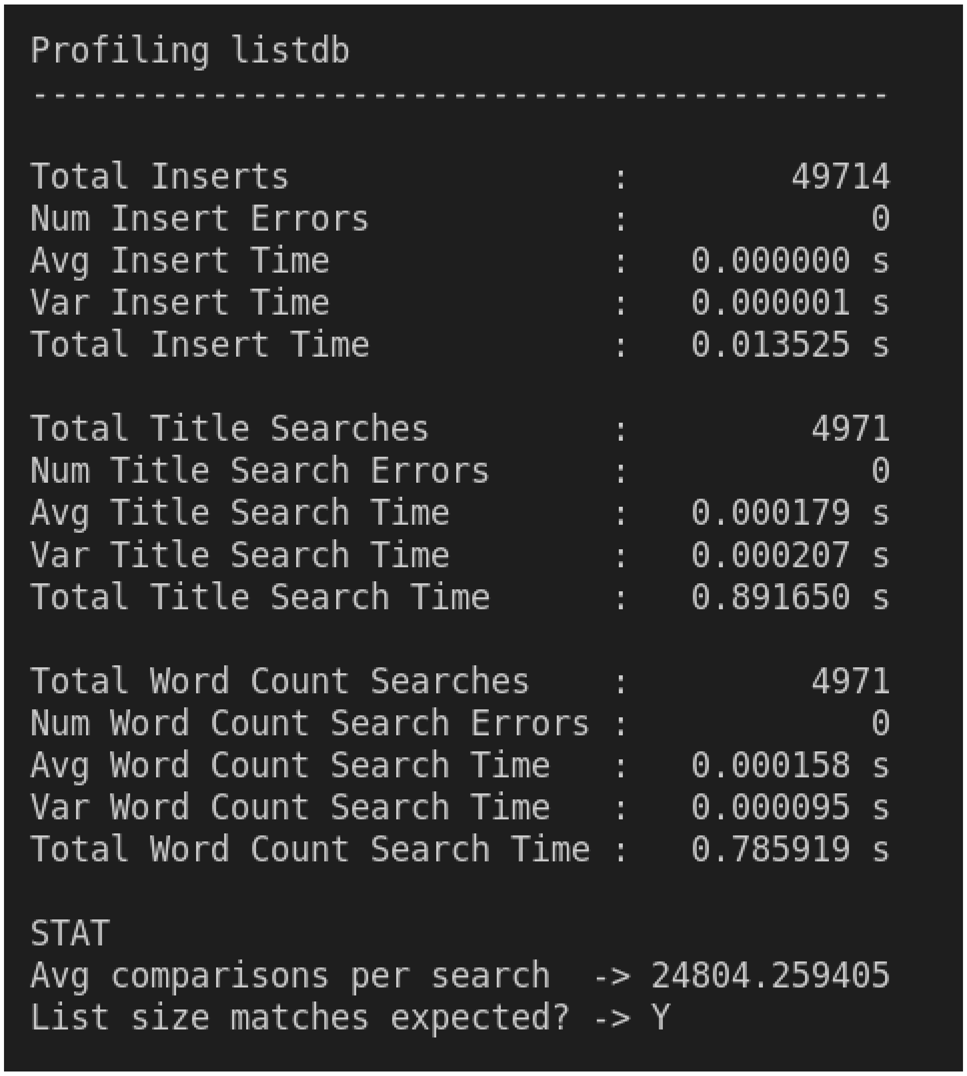
Name: Oisín Cullen |Student number: 20332292 | Date: 17/11/2022

Task 1

The first task was to build a simple binary search tree which would store it based on the ascii value of the character and assumed it was nonrepeating. All functions I implemented were found to work well and mostly al worked by recursively calling that function. The binary search tree made here was unbalanced as it was not required for this task.

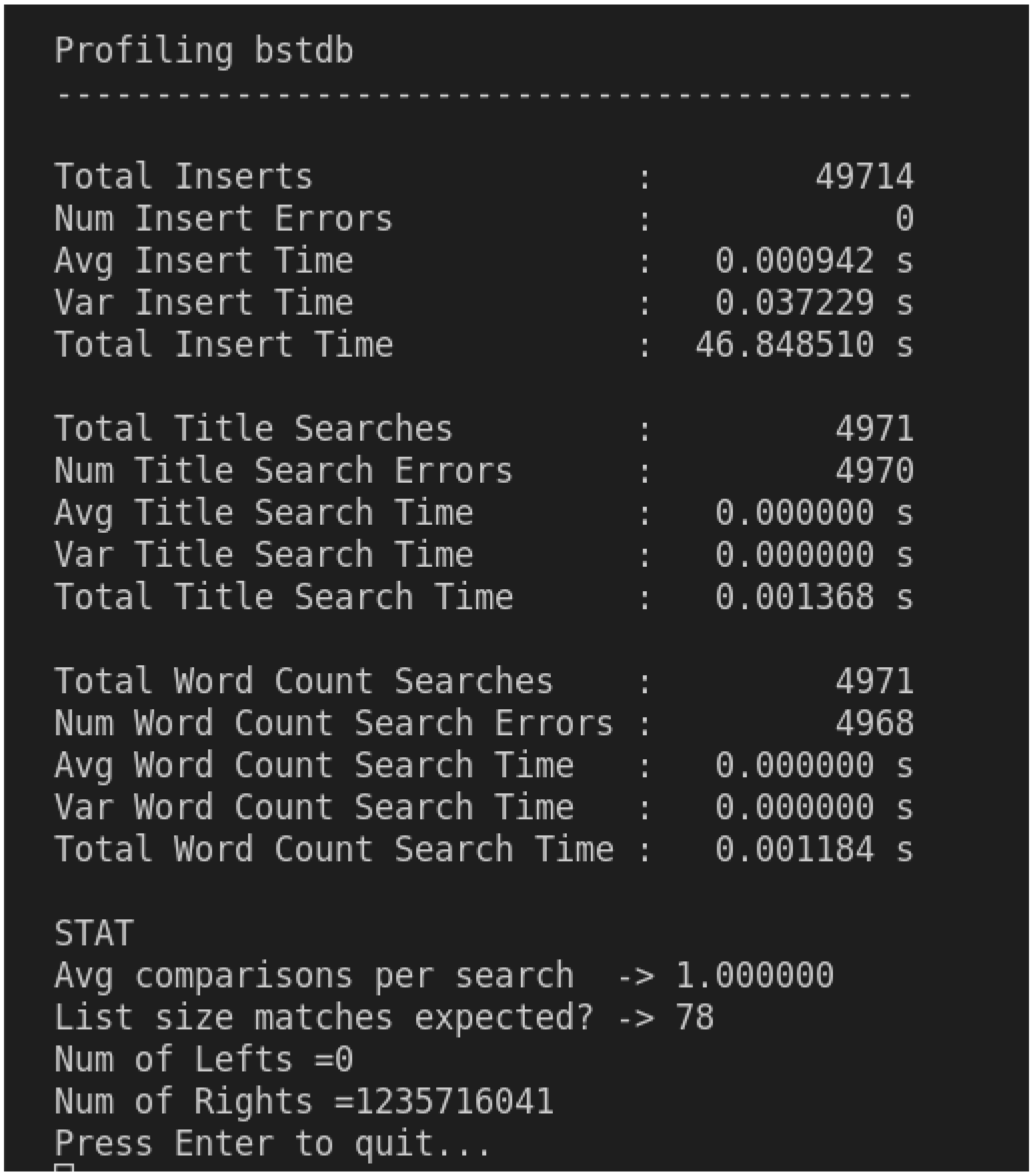
Task 2

In task 2 we were required to design and implement a data base of a binary search tree into a large data base



```
Profiling listdb
-------------------------------------------

Total Inserts                :       49714
Num Insert Errors            :           0
Avg Insert Time              :   0.000000 s
Var Insert Time              :   0.000001 s
Total Insert Time            :   0.013525 s

Total Title Searches         :        4971
Num Title Search Errors      :           0
Avg Title Search Time        :   0.000179 s
Var Title Search Time        :   0.000207 s
Total Title Search Time      :   0.891650 s

Total Word Count Searches    :        4971
Num Word Count Search Errors :           0
Avg Word Count Search Time   :   0.000158 s
Var Word Count Search Time   :   0.000095 s
Total Word Count Search Time :   0.785919 s

STAT
Avg comparisons per search  -> 24804.259405
List size matches expected? -> Y
```

Figure 1 Linked list performance



```
Profiling bstdb
-------------------------------------------

Total Inserts                 :       49714
Num Insert Errors             :           0
Avg Insert Time               :   0.000942 s
Var Insert Time               :   0.037229 s
Total Insert Time             :  46.848510 s

Total Title Searches          :        4971
Num Title Search Errors       :        4970
Avg Title Search Time         :   0.000000 s
Var Title Search Time         :   0.000000 s
Total Title Search Time       :   0.001368 s

Total Word Count Searches     :        4971
Num Word Count Search Errors :        4968
Avg Word Count Search Time   :   0.000000 s
Var Word Count Search Time   :   0.000000 s
Total Word Count Search Time :   0.001184 s

STAT
Avg comparisons per search  -> 1.000000
List size matches expected? -> 78
Num of Lefts =0
Num of Rights =1235716041
Press Enter to quit...
```

Figure 2 BST performance

Whilst the BST had a poor insertion time which was not expected its average search time performed better then the linked list with regards to searching.

Design Decisions. Task 2 Q2

When designing the BST I tried to implement numerous methods to create a balanced Binary search Tree. I first tried to generate an algorithmic number generator which would find the total size of the database (this could be found by looking into profile.c and seeing the generator variables) and take the middle number then take the middle number of that one and find the median left and right of that number until a counter was reached

However I had issues when it came to the 0 case as I would often generate values on the right hand side of the tree which shouldn't be there.

I then tried to implement an Array based approach so that a finite amount of numbers were present so that no overlap of the same numbers being used wouldn't occur, however I kept getting segmentation faults with recursion.

I then attempted to implement a barrel shifter esq approach so that the number would be swapped but also shift the remaining values n places left and right to find the next median number however in testing I had the same issue with the algorithm as I couldn't find a solution which would work for the 0 case.

I decided to then use a simple median finder which puts the array in a semi balanced position so whilst it doesn't guarantee ologn every search time it does guarantee stability and a faster search time compared to linked lists.

Extra Testing

When I was doing extra testing to insure that the bst function I had an insertion counter which read the number of insertions so I could tell the progress of the Bst. I also had counters like search and comparison which helped measure how many nodes it would see to get to a location.

I also added right and left counters to see if the tree was balanced if both values were the same it meant the tree was balanced!