jbugeja 758397

Binary Search Tree Analysis

1 Introduction
A binary search tree (bst) are an effective way to store large amounts of sorted data, if the tree is perfectly balanced the search and insertion operations can be performed in 1.44*O(logn) time, however in worst case this can go up to O(n) time as the tree effectively becomes a link list if sorted data is entered.

1.1 Purpose
The purpose of this report is to analyse the efficiency of different implementations of a bst and the way data can affect search and insertion times.

1.2 Scope
Large amounts of data are important for analysis is as bsts are normally meant to store records in large scale like in dictionaries or any other sort of key lookup databases. A range of different search keys are important in analysis to access different areas of the tree.

1.3 Method
The bst created in this example is one that mimics a yelp database. For Stage 1, a bst was created that handles duplicate keys by inserting them to the left of a key with the same name. For Stage 2, duplicate keys were inserted next to each other through a linked list. The testing was done on a MacBook air with a 1.3Ghz intel i5 and 4gb of RAM.

1.4 Limitations
- The C language must be used to code the program
- Programs must use provided data

1.5 Assumptions
It's been assumed that no entered key will be larger than 64 characters and no data entered larger than 1465 characters. The data entered into the dictionary is assumed to be in .csv format and the keys in .txt with a newline character marking the end of each key to be searched. It is also assumed that all entries into the dictionary are correct.

2. Findings

2.1 Comparison of bsts
The following table summarises the programs run times for randomly entered keys. The same sets of keys were used for business and business_alternative as well as user and user_alternative. Keys consisted of 7 true keys and 2 false ones. Multiple tests were run with different randomized keys and the median times were taken. Runtime includes both insertion and lookup. Output files were as expected with all keys printed out and any keys not found with "NOTFOUND" printed.

Table 1: Comparison of two portable computers

	Stage 1 BST – Ignoring duplicate data (time)	**Stage 2 BST – Adding duplicate data in linked list (time)**
yelp_academic_dataset_business	0m3.381s	0m3.364s
yelp_academic_dataset_business_alternative	2m19.641s	2m17.550s
yelp_academic_dataset_user	0m40.137s	0m39.422s
yelp_academic_dataset_user_alternative	11m24.767s	11m45.875s

2.2 Runtime Analysis

Table 1 shows how the order of data entered will change the behavior of the bst, entering a sorted data into the bst increases the time by a factor of ~40 for the business dataset and a factor of ~20 for the user dataset.
It is expected that the data for sorted takes more time than randomized data since it is effectively being inserted into a linked list with insertion and lookup time of O(n) compared to O(logn) for a bst. Stage 2 data is expected to be faster than Stage 1 as duplicate data is inserted earlier in the list and found with less comparisons, this is shown for the business dataset but not for the user dataset, this is probably because of the sheer amount of data that is being processed compared in the user data compared to business data.
There is close to 10x more data being processed which would increase processing time by a factor of 10, the reason the alternative data may not have increased by exactly that amount could be due to the processor dedicating more power for a longer running task or helping it run more efficiently if it's taking a while.
CPU and RAM usage at the time of testing may also cause variations in time which could have caused the unexpected discrepancy in the alternative user data.

2.3 Comparison Analysis

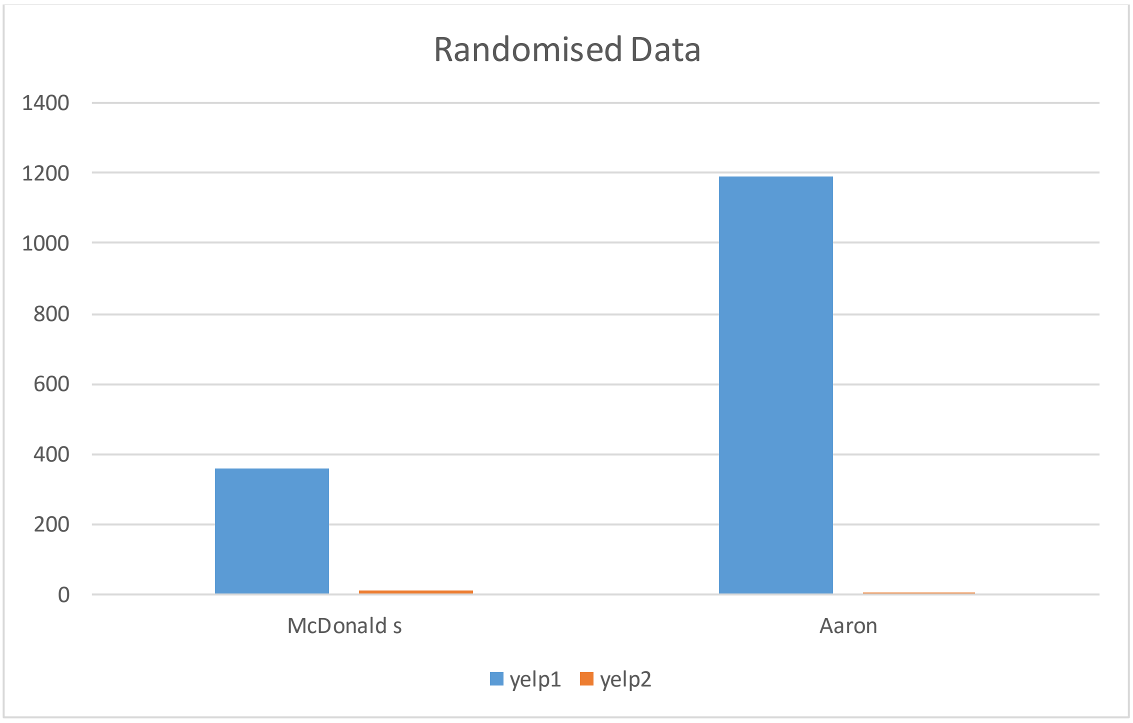
Business Data
Figure 1 shows the output for yelp1 with business set data, the number of comparisons are quite low for most keys except for ones with a high number of duplicates in the dictionary. McDonald s has a comparison number of 361 which is massively greater than the mid 20's median for the keys, Gamestop also has a large comparison number of 40. Comparing these to Figure 5 where duplicate keys are stored in a linked list together the number of comparisons for McDonald s and Gamestop drop to 10 and 22 respectively, this greatly improves the efficiency of the program and helps to make the bst more balanced and

behave more towards the 1.44O(logn) behavior this is shown visually in Graph 1. The data average lookup of ~20 corresponds exactly to a 1.44O(logn) of base 2 (1.44*log(77445) = 22.73). This is shown by how the number of comparisons between almost all of the keys drop between Figure 1 and 5, meaning the tree height is becoming smaller due to the lack of extra branches.

Business Data Alternative

The effect of having data placed in sorted order is seen clearly between Figure 2 and Figure 1. The number of comparisons grows massively in Figure 2 since a sorted dataset has been inserted into the dictionary instead of randomized keys. The lookup time grows massively from ~3 seconds to ~2 minutes. This is also seen between Figures 5 and 6 with similar computation times. The comparisons number of the keys instead corresponds directly to the data's alphabetical order, with Build a Bear at 8447 comparisons since it starts with 'B' and University of Pittsburgh at 54280 since it starts with 'U' this tree behaves more like a linked list than a bst and it's behavior is linear of O(n). The difference between the simple bst in Figure 1 and the one in Figure 2 is small since they are both effectively a linked list however McDonald s drops from 32525 comparisons to 32181 respectively, the height of the tree may drop a small amount due duplicate keys being stored separately but the difference is miniscule due to the large number of entries.

Graph 1: The difference between the two types of bsts for randomized data



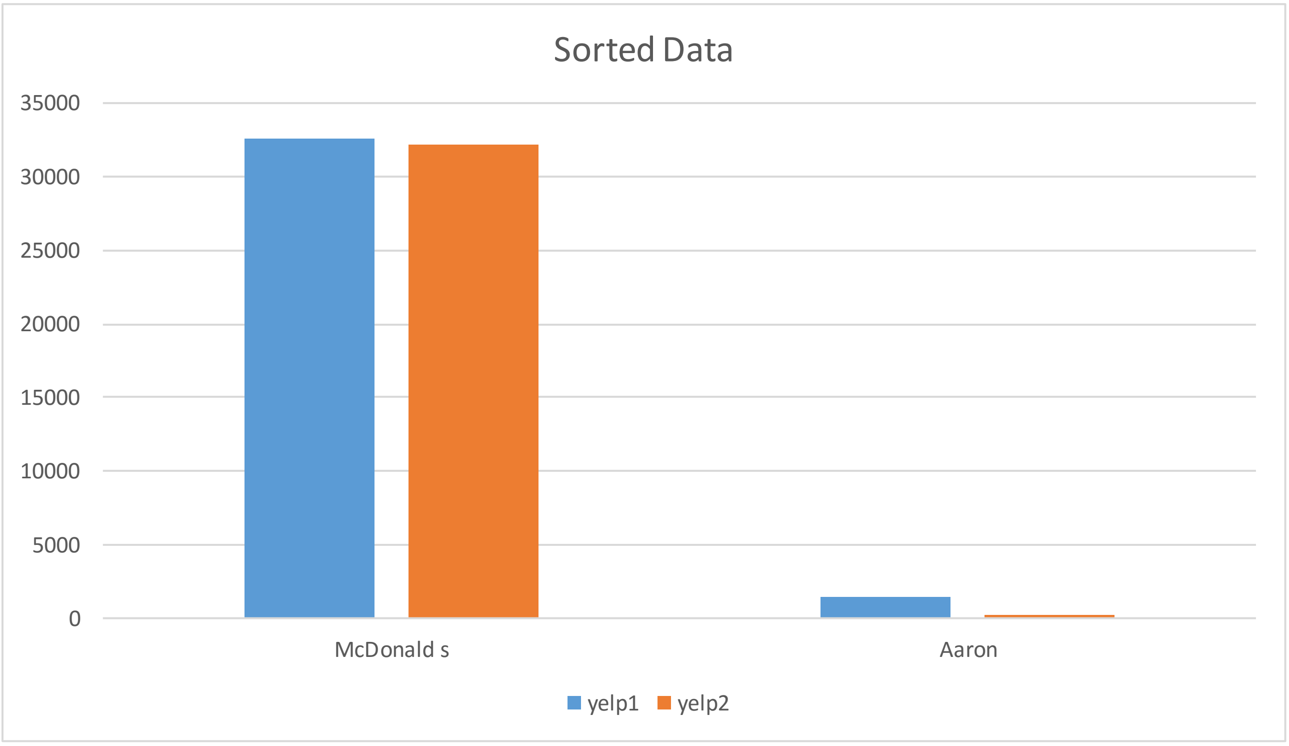
User Data

The user data is filled with usernames instead of people instead of business data, this creates many more duplicates which can be seen with the difference between Figure 3 and Figure 7. All of the name comparisons drop from an average of ~1500 in Figure 3 to ~15 in Figure 7, this improves the program's efficiency 100 fold when duplicate data is stored together rather than throughout the tree. The tree in Figure 3 doesn't behave in the worst case of O(n) but is an inefficient way of storing data when compared to Figure 7. The tree is effectively multiple linked lists when there is a massive amount of data but a relatively small amount of individual keys.

User Data Alternative

The effect of having data placed in sorted order is seen just as clearly in the user data as in the business data. The number of comparisons grow from the mid 1000's in Figure 3 to being reliant on the data's alphabetic order in Figure 4 Russel's comparisons change from 19 to 38991 and Aaron changes from 1193 to 1417, showing that Aaron is higher up in the sorted list than Russel. This corresponds to a worst case behavior of O(n) just like in the business dataset. The difference between the two types of bsts is more noticeable in the user dataset than the previous business dataset due to there being a larger amount of duplicate keys, Aaron drops from 1417 to 283 and Christine from 9650 to 8321. The difference in how the tree handles duplicate data improves the behavior slightly however having a sorted dataset still makes the tree very inefficient and behaving more towards worst case O(n) behavior this is shown visually in Graph 2.

Graph 2: The difference between the two types of bsts for sorted data

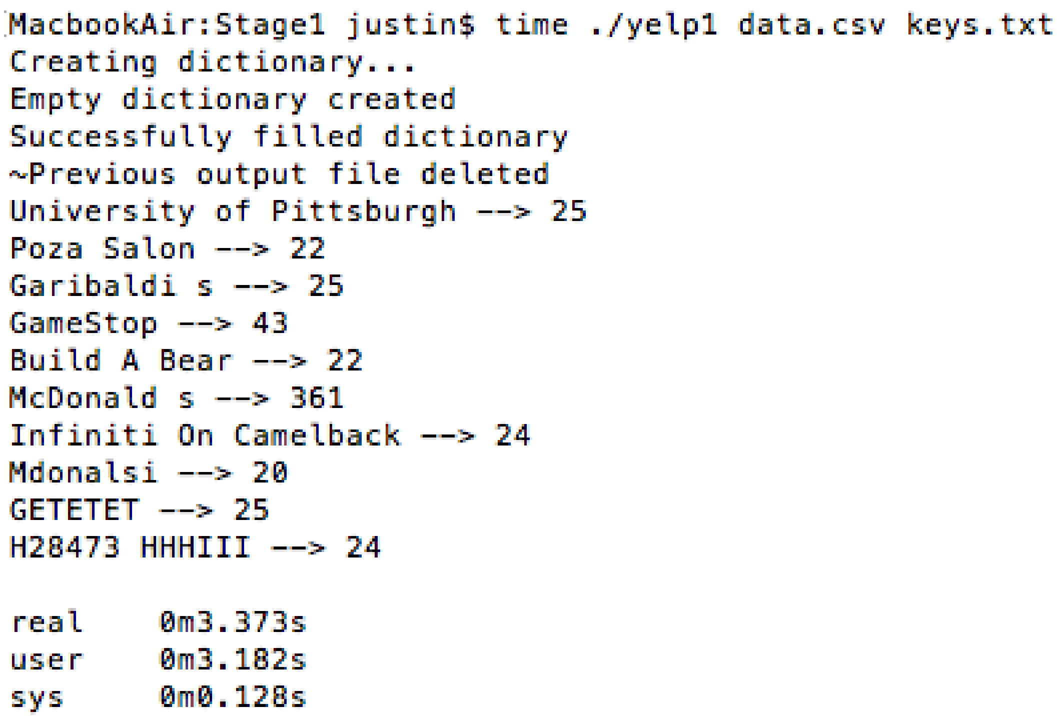


3. Conclusion

It can be concluded that a binary search tree is a very efficient method for storing data only if the data is not already sorted. Data can be looked up in a matter of less than 4 seconds for a dataset consisting of close to 80,000 entries when entries are not sorted but that grows to over 2 minutes when data is entered in sorted order, the similar happens with a dataset of close to 400,000 entries lookup moving from ~40seconds to ~12minutes. The type of bst used also improves behavior immensely when duplicate data is stored within a linked list together than when it's scattered throughout the trees, this helps the tree to become more balanced as well as improving efficiency. However, the difference is not as evident when sorted data is entered, reinforcing how important the way data is entered into the tree is to insertion and sort times. The behavior clearly changes from O(logn) to O(n) in both types of bsts, the best case and worst case behavior respectively which agrees with previous theory studied. One way that this could be fixed would be by balancing the tree through rotations every time a new key is inserted.

jbugeja 758397

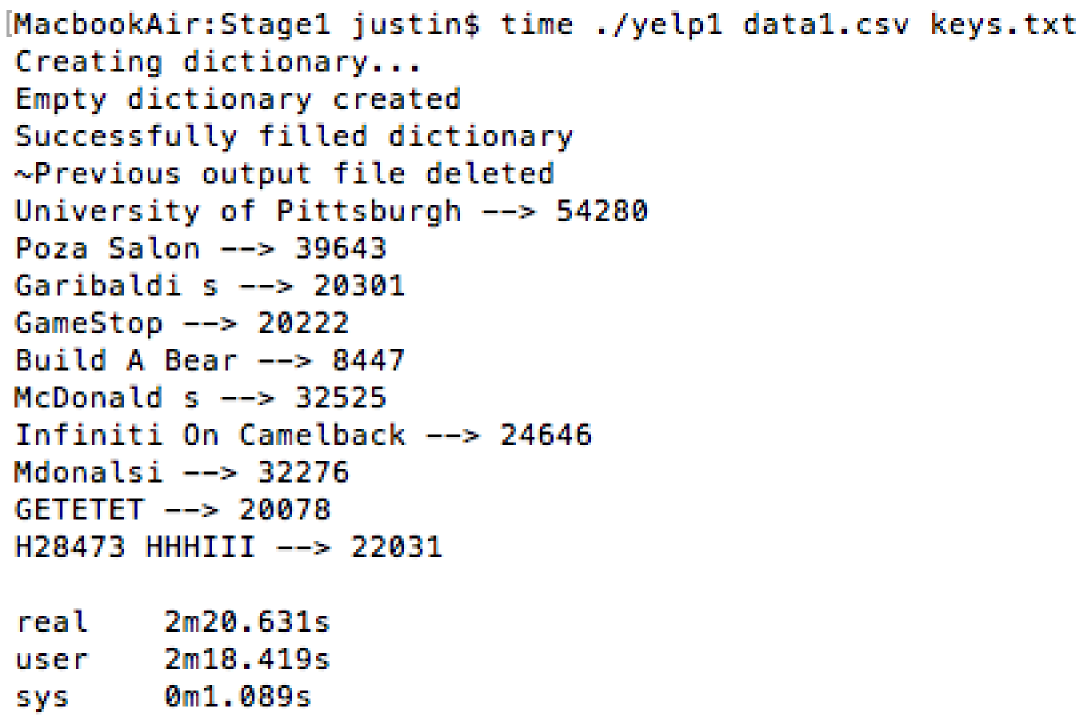
Figure 1: yelp1_academic_dataset_business



```
MacbookAir:Stage1 justin$ time ./yelp1 data.csv keys.txt
Creating dictionary...
Empty dictionary created
Successfully filled dictionary
~Previous output file deleted
University of Pittsburgh --> 25
Poza Salon --> 22
Garibaldi s --> 25
GameStop --> 43
Build A Bear --> 22
McDonald s --> 361
Infiniti On Camelback --> 24
Mdonalsi --> 20
GETETET --> 25
H28473 HHHIII --> 24

real	0m3.373s
user	0m3.182s
sys	0m0.128s
```

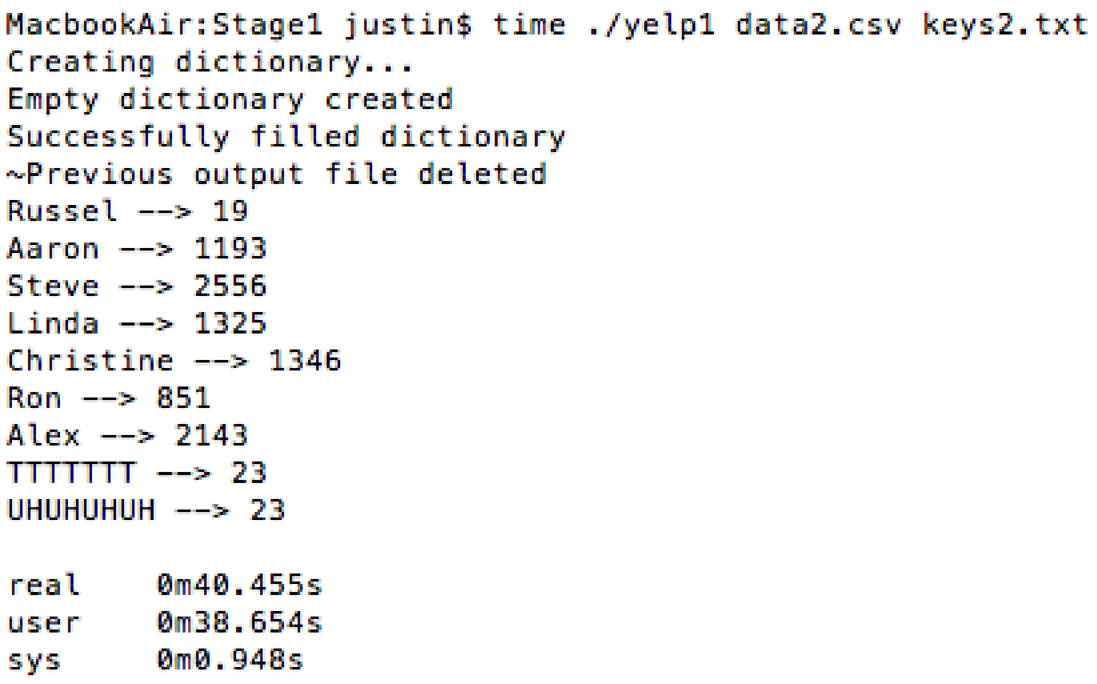
Figure 2: yelp1_academic_dataset_business_alternative



```
MacbookAir:Stage1 justin$ time ./yelp1 data1.csv keys.txt
Creating dictionary...
Empty dictionary created
Successfully filled dictionary
~Previous output file deleted
University of Pittsburgh --> 54280
Poza Salon --> 39643
Garibaldi s --> 20301
GameStop --> 20222
Build A Bear --> 8447
McDonald s --> 32525
Infiniti On Camelback --> 24646
Mdonalsi --> 32276
GETETET --> 20078
H28473 HHHIII --> 22031

real	2m20.631s
user	2m18.419s
sys	0m1.089s
```

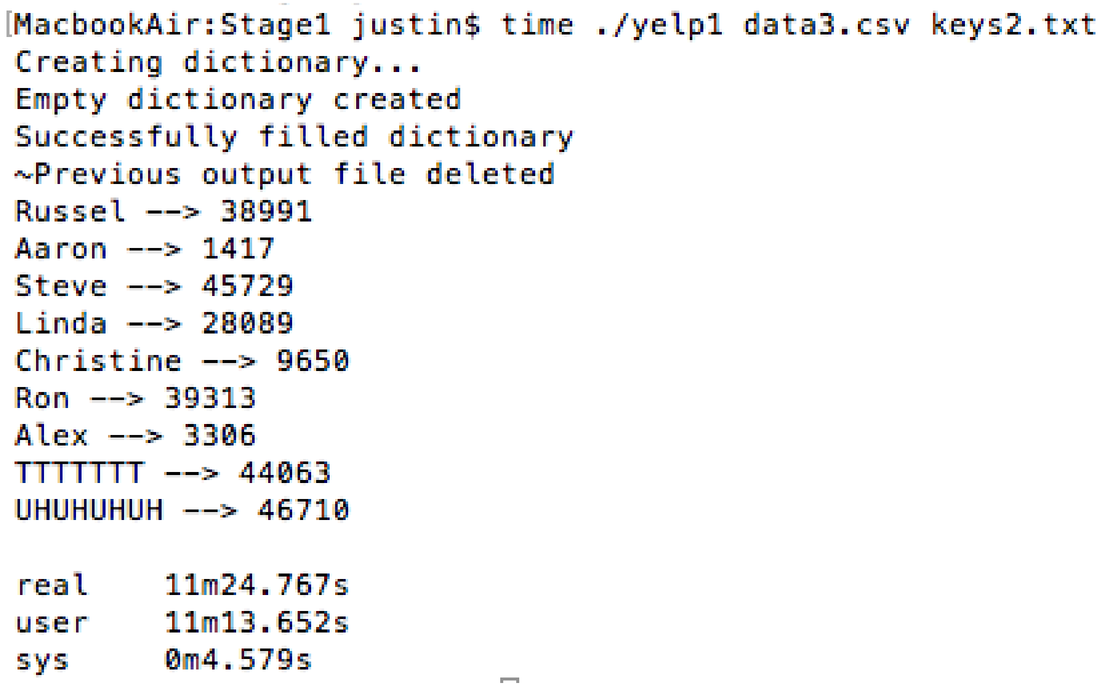
Figure 3: yelp1_academic_dataset_user



```
MacbookAir:Stage1 justin$ time ./yelp1 data2.csv keys2.txt
Creating dictionary...
Empty dictionary created
Successfully filled dictionary
~Previous output file deleted
Russel --> 19
Aaron --> 1193
Steve --> 2556
Linda --> 1325
Christine --> 1346
Ron --> 851
Alex --> 2143
TTTTTTT --> 23
UHUHUHUH --> 23

real    0m40.455s
user    0m38.654s
sys     0m0.948s
```

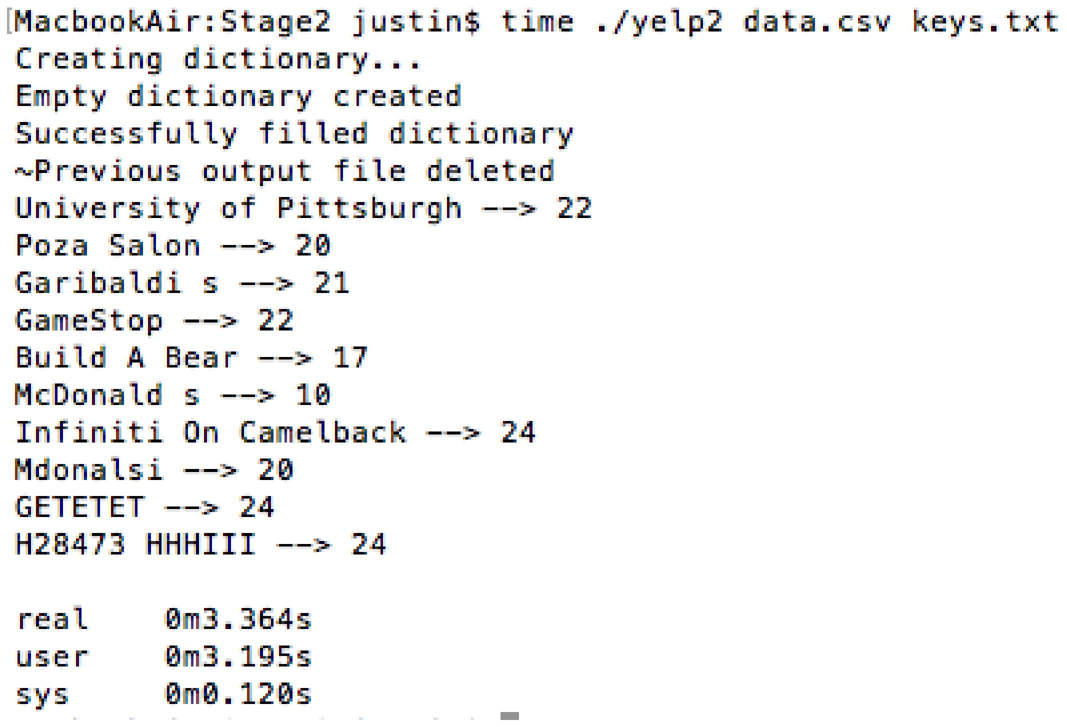
Figure 4: yelp1_academic_dataset_user_alternative



```
MacbookAir:Stage1 justin$ time ./yelp1 data3.csv keys2.txt
Creating dictionary...
Empty dictionary created
Successfully filled dictionary
~Previous output file deleted
Russel --> 38991
Aaron --> 1417
Steve --> 45729
Linda --> 28089
Christine --> 9650
Ron --> 39313
Alex --> 3306
TTTTTTT --> 44063
UHUHUHUH --> 46710

real    11m24.767s
user    11m13.652s
sys     0m4.579s
```

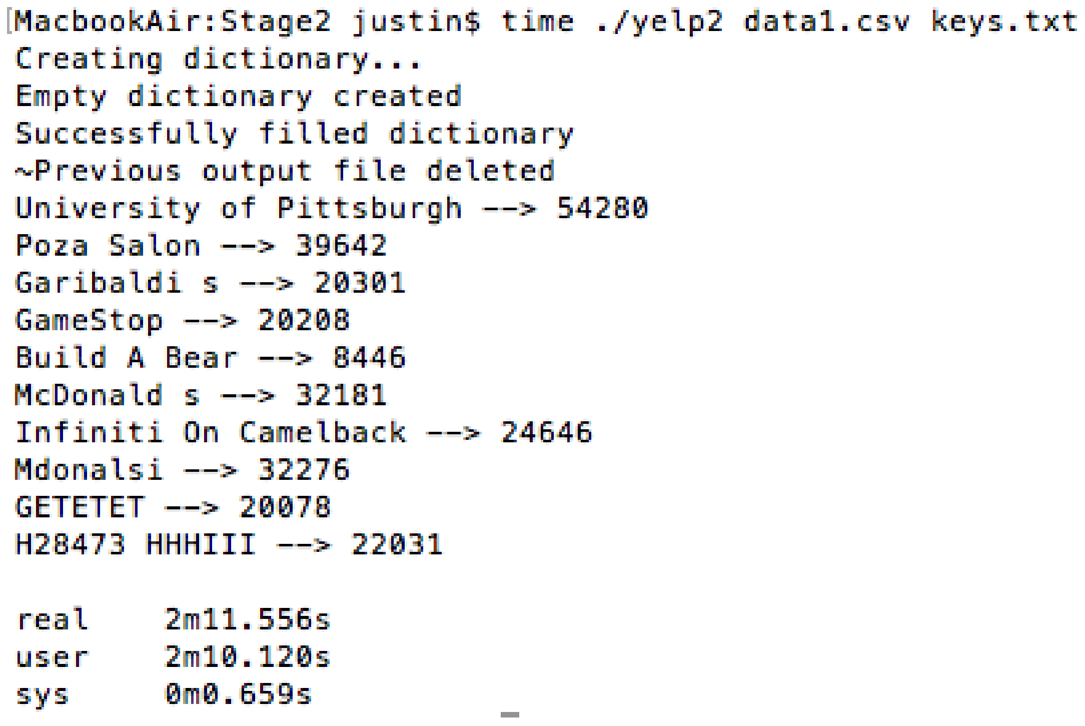
Figure 5: yelp2_academic_dataset_business



```
[MacbookAir:Stage2 justin$ time ./yelp2 data.csv keys.txt
Creating dictionary...
Empty dictionary created
Successfully filled dictionary
~Previous output file deleted
University of Pittsburgh --> 22
Poza Salon --> 20
Garibaldi s --> 21
GameStop --> 22
Build A Bear --> 17
McDonald s --> 10
Infiniti On Camelback --> 24
Mdonalsi --> 20
GETETET --> 24
H28473 HHHIII --> 24

real    0m3.364s
user    0m3.195s
sys     0m0.120s
```

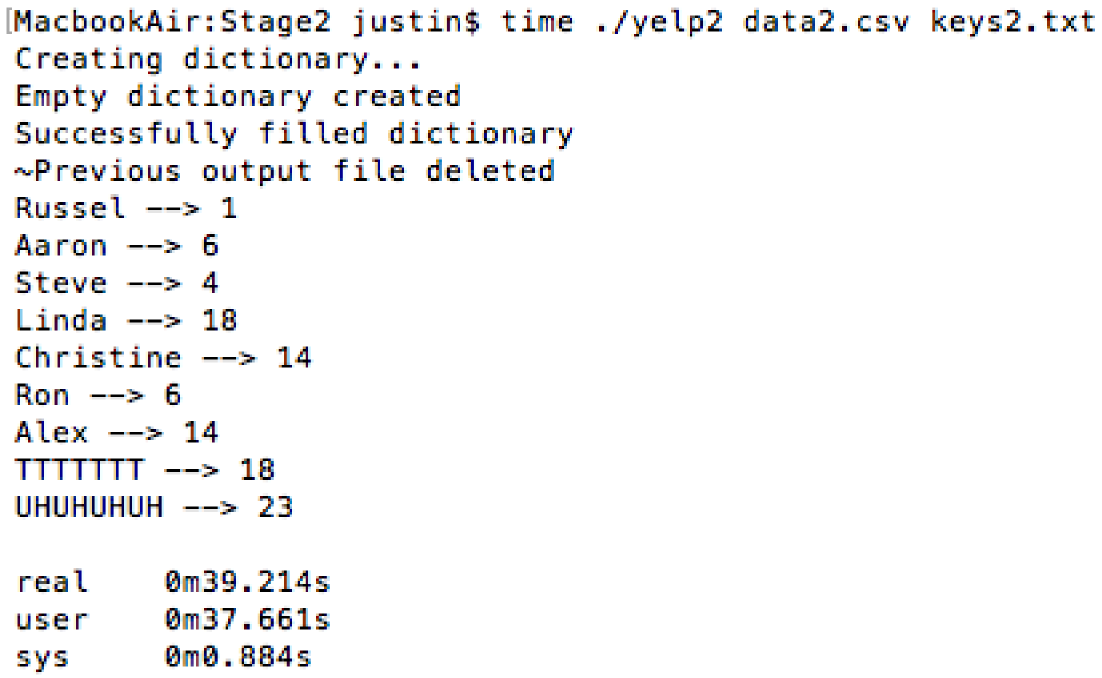
Figure 6: yelp2_academic_dataset_business_alternative



```
[MacbookAir:Stage2 justin$ time ./yelp2 data1.csv keys.txt
Creating dictionary...
Empty dictionary created
Successfully filled dictionary
~Previous output file deleted
University of Pittsburgh --> 54280
Poza Salon --> 39642
Garibaldi s --> 20301
GameStop --> 20208
Build A Bear --> 8446
McDonald s --> 32181
Infiniti On Camelback --> 24646
Mdonalsi --> 32276
GETETET --> 20078
H28473 HHHIII --> 22031

real    2m11.556s
user    2m10.120s
sys     0m0.659s
```

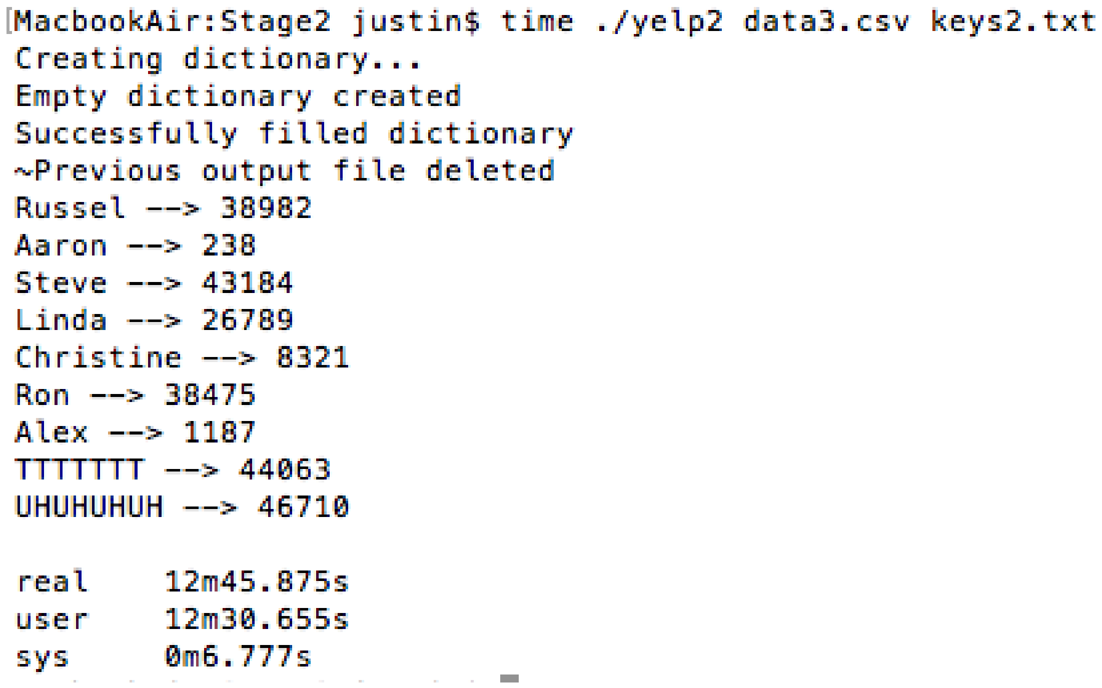
Figure 7: yelp2_academic_dataset_user



```
MacbookAir:Stage2 justin$ time ./yelp2 data2.csv keys2.txt
Creating dictionary...
Empty dictionary created
Successfully filled dictionary
~Previous output file deleted
Russel --> 1
Aaron --> 6
Steve --> 4
Linda --> 18
Christine --> 14
Ron --> 6
Alex --> 14
TTTTTTT --> 18
UHUHUHUH --> 23

real	0m39.214s
user	0m37.661s
sys	0m0.884s
```

Figure 8: yelp2_academic_dataset_user_alternative



```
MacbookAir:Stage2 justin$ time ./yelp2 data3.csv keys2.txt
Creating dictionary...
Empty dictionary created
Successfully filled dictionary
~Previous output file deleted
Russel --> 38982
Aaron --> 238
Steve --> 43184
Linda --> 26789
Christine --> 8321
Ron --> 38475
Alex --> 1187
TTTTTTT --> 44063
UHUHUHUH --> 46710

real	12m45.875s
user	12m30.655s
sys	0m6.777s
```