FINAL PROJECT REPORT
TOPIC: AMAZON CO-PURCHASING ANALYSIS

INSTRUCTOR : DR YINGHUI WU
PRESENTED BY : GAUTAM BHAT KHANDIGE AND ABHI CHADHA
GROUP NUMBER : 16

ABSTRACT

In order to solve the old aging question of slow time performance and poor capabilities of query processing over large metadata the team members took upon the project titled amazon co - purchasing which inculcates developing an analytics engine which performs differently from the conventional form of query processing in order to aim and achieve better processing for a faster output over a defined dataset. As we deep dive into the project we would meet upon the steps which includes pre processing of the data to the algorithm development and finally to the deployment which was achieved over an executed time and gave astonishing results which outlasted our expectations of how the faster processing of data over different pattern sets could be achieved in a significantly lower amount of time.

INTRODUCTION

PROBLEM STATEMENT

- Developing a co-purchasing analytics engine that has a faster indexing time in order to extract the data and present it for lesser time wastage.
- In order to create and deploy better search patterns an algorithm needs to be devised that is faster and better from conventional industry style searching over a large data scale.
- Build an efficient query for a unique search engine that compliments the algorithm build up and helps in efficient data processing.

WHY THIS PROJECT

- The project was purely chosen based on what implementations in the real world it had to offer.
- Time is a huge factor in determining as to why an algorithm is efficient or not
- An efficient algorithm in regards with a query helps us in faster indexing and accessing and gives an output that leads to lesser time wastage which is what we are aiming for through this project.
- The amazon dataverse is a vast database that offers not only millions but billions of bytes of data that are highly customer orientated which helps us in tackling a real-world dilemma.
- Furthermore, the amazon metadata is vast and diverse which helps us in having a project that has global outreach.

APPROACH

- The approach we take upon is highly different from conventional techniques offered.
- Modern development tools that have breached the industry we use **SQL** as the base for building are unique query over the defined database.
- Python with its vast capabilities and functions offered is the one we chose to go ahead with to build our algorithm that would be devised for faster accessing.

RELATION BETWEEN PROJECT AND PREVIOUS WORK

- Learnt about mySQl connector and python through project papers and the theoretical part was taught heavily in class by Professor Wu
- Learning about the structure and performance of structured and unstructured data. Knowledge on data structures was obtained in CSDS233 by Professor Michael Lewicki and Professor Erman Ayday.
- Comprehensive detailing of query processing was taught thoroughly throughout the semester as part of course work.

- Using python as a front end to scrape and store data in dataframes was learnt in CSDS 133 by Dr Orhan Ozguner.
- Basicity of coding and syntax was taught by Professor Dr Harold Connamacher in CSDS 132.

ALGORITHM DEVELOPMENT

THERE WERE MAINLY 4 STEPS WE TOOK UPON TO BUILD AND DEPLOY OUR PROJECT AS A WHOLE WHICH CAN BE SEEN BELOW WITH ITS DETAILED EXPLANATIONS OFFERED

STEP 1

PARSING METADATA THROUGH SQL

- Uploading the data into python in specific to jupyter notebook for its better representation.
- Cleaning and pre-processing the data so that we can better run our code through it for faster and better algorithmic efficiency….operations performed
- Converting the data into semi-structured data into structured data through……

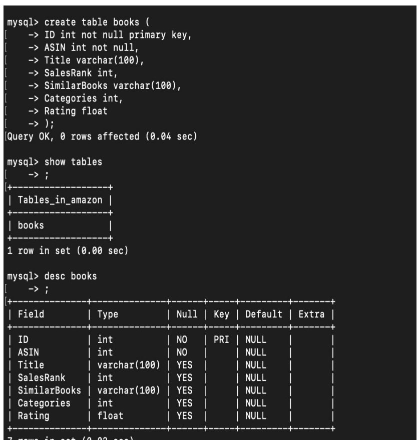
```
import pandas as pd

fh = open('amazon-meta.txt','r')
Ids = []
ASIN = []
title =[]
group =[]
salesrank = []
similar = []
categories = []
rating = []

for i in fh.readlines():
    if "Id" == i[:2]:
        if i.isdigit:
            Ids.append((int(i[6:])))
    if "ASIN" == i[:4]:
        if i.isdigit:
            ASIN.append(i[6:].strip("\n"))
    if i.find("title") >= 0:
        title.append(i[i.find("title") + 6:])
    if i.find("group") >= 0:
        group.append(i[i.find("group") + 6:].strip("\n"))
    if i.find("salesrank") >= 0:
        salesrank.append(int(i[i.find("salesrank")+10:].strip("\n")))
    if i.find("similar") >= 0:
        similar.append(i[i.find("similar")+8:].strip("\n"))
    if i.find('categories') >= 0:
        categories.append(int(i[i.find("categories")+12:].strip("\n")))
    if i.find("avg rating") >= 0:
        rating.append(float(i[i.find("avg rating")+12:].strip("\n")))
amazon_database = {'ID':Ids[1:], 'ASIN':ASIN[1:], 'Title': title, "Group": group, 'Sales_Rank':salesrank,'Similar_Book
print(df)
```

```
      ID        ASIN                                                Title  \
0      1  0827229534          Patterns of Preaching: A Sermon Sampler\n
1      2  0738700797                      Candlemas: Feast of Flames\n
2      3  0486287785   World War II Allied Fighter Planes Trading Ca...
3      4  0842328327   Life Application Bible Commentary: 1 and 2 Ti...
4      5  1577943082   Prayers That Avail Much for Business: Executi...
..   ...         ...                                                ...
146  147  0743202554                The New Jewish Wedding, Revised\n
147  148  0812550749                                    Firebird\n
148  149  B00000IC80                Pot O' Gold/Made for Each Other\n
149  150  0764118218   Vocabulary Builder: French : Mastering the Mo...
150  151  B00000IC82            Laurel & Hardy - Flying Deuces/Utopia\n

     Group  Sales_Rank                                        Similar_Books  \
0     Book      396585  5  0804215715  156101074X  0687023955  068707...
1     Book      168596  5  0738700827  1567184960  1567182836  073870...
2     Book     1270652                                                   0
3     Book      631289  5  0842328130  0830818138  0842330313  084232...
4     Book      455160  5  157794349X  0892749504  1577941829  089274...
..     ...         ...                                                ...
146   Book       11398  5  1580231942  0805060839  0963575309  074321...
147   Book       77008  5  0756401615  0886778905  067187750X  075640...
148    DVD       51310                3  B00006AUGQ  B00005REKR  B0002KPHZ6
149   Book      966856                                                   0
150    DVD       44671                                           1  B00009YXEW

     No_of_Categories  Rating
0                   2     5.0
1                   2     4.5
2                   1     5.0
3                   5     4.0
4                   2     0.0
..                ...     ...
146                 4     4.5
147                 3     4.0
148                20     0.0
149                 4     0.0
150                16     3.0

[151 rows x 8 columns]
```

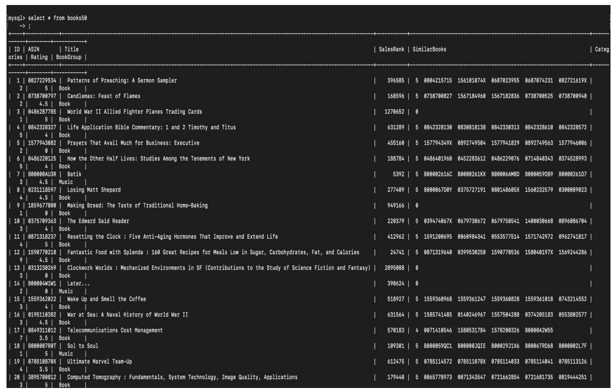
STEP 2

LOADING THE DATA FOR QUERY PROCESSING

- Loading the data from python into SQL and transferring it into a front end load to a back end load using an SQL connector for developing the query and further processing it.
- We use a predefined index to sort out the data into different indexes for faster preprocessing.
- Indexing which is a crucial part of data structures helps us in sorting the data for faster accessing through a defined hash function and key. (picture show table etc. code)



```
mysql> create table books (
    -> ID int not null primary key,
    -> ASIN int not null,
    -> Title varchar(100),
    -> SalesRank int,
    -> SimilarBooks varchar(100),
    -> Categories int,
    -> Rating float
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql> show tables
    -> ;
+-------------------+
| Tables_in_amazon |
+-------------------+
| books            |
+-------------------+
1 row in set (0.00 sec)

mysql> desc books
    -> ;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| ID           | int          | NO   | PRI | NULL    |       |
| ASIN         | int          | NO   |     | NULL    |       |
| Title        | varchar(100) | YES  |     | NULL    |       |
| SalesRank    | int          | YES  |     | NULL    |       |
| SimilarBooks | varchar(100) | YES  |     | NULL    |       |
| Categories   | int          | YES  |     | NULL    |       |
| Rating       | float        | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
```



```
mysql> select * from books50
    -> ;
| ID | ASIN       | Title                                                                                       | SalesRank | SimilarBooks                                                  | Categories | Rating | BookGroup |
|  1 | 0827229534 | Patterns of Preaching: A Sermon Sampler                                                     |    396585 | 5  0804215715  156101074X  0687023955  0687074231  082721619X |  2 |   5 | Book  |
|  2 | 0738700797 | Candlemas: Feast of Flames                                                                  |    168596 | 5  0738700827  1567184960  1567182836  0738700525  0738700940 |  2 | 4.5 | Book  |
|  3 | 0486287785 | World War II Allied Fighter Planes Trading Cards                                            |   1270652 | 0                                                             |  1 |   5 | Book  |
|  4 | 0842328327 | Life Application Bible Commentary: 1 and 2 Timothy and Titus                                |    631289 | 5  0842328130  0830818138  0842330313  0842328610  0842328572 |  5 |   4 | Book  |
|  5 | 1577943082 | Prayers That Avail Much for Business: Executive                                             |    455160 | 5  157794349X  0892749504  1577941829  0892749563  1577946006 |  2 |   0 | Book  |
|  6 | 0486220125 | How the Other Half Lives: Studies Among the Tenements of New York                           |    188784 | 5  0486401960  0452283612  0486229076  0714840343  0374528993 |  5 |   4 | Book  |
|  7 | B00000AU3R | Batik                                                                                       |      5392 | 5  B00002616C  B0000261KX  B00006AM8D  B000059OB9  B000026107 |  3 | 4.5 | Music |
|  8 | 0231118597 | Losing Matt Shepard                                                                         |    277409 | 5  B000067D0Y  0375727191  080148605X  1560232579  0300089023 |  4 | 4.5 | Book  |
|  9 | 1859677800 | Making Bread: The Taste of Traditional Home-Baking                                          |    949166 | 0                                                             |  1 |   0 | Book  |
| 10 | 0375709363 | The Edward Said Reader                                                                      |    220379 | 5  039474067X  0679730672  0679750541  1400030668  0896086704 |  3 |   4 | Book  |
| 11 | 0871318237 | Resetting the Clock : Five Anti-Aging Hormones That Improve and Extend Life                 |    412962 | 5  1591200695  0060984341  0553577514  1571742972  0962741817 |  4 |   5 | Book  |
| 12 | 1590770218 | Fantastic Food with Splenda : 160 Great Recipes for Meals Low in Sugar, Carbohydrates, Fat, and Calories |     24741 | 5  0871319640  0399530258  1590770536  158040197X  1569244286 |  9 | 4.5 | Book  |
| 13 | 0313230269 | Clockwork Worlds : Mechanized Environments in SF (Contributions to the Study of Science Fiction and Fantasy) |   2895088 | 0                                                             |  3 |   0 | Book  |
| 14 | B00004W1W1 | Later...                                                                                    |    390624 | 0                                                             |  2 |   0 | Music |
| 15 | 1559362022 | Wake Up and Smell the Coffee                                                                |    518927 | 5  1559360968  1559361247  1559360828  1559361018  0743214552 |  3 |   4 | Book  |
| 16 | 0195110382 | War at Sea: A Naval History of World War II                                                 |    631564 | 5  1585741485  0140246967  1557504288  0374205183  0553802577 |  3 | 4.5 | Book  |
| 17 | 0849311012 | Telecommunications Cost Management                                                          |    570183 | 4  0071410546  1580531784  1578200326  B0000A2W55             |  7 | 3.5 | Book  |
| 18 | B000007R0T | Sol to Soul                                                                                 |    109301 | 5  B000059QC1  B00000JQIE  B00029J1X6  B0006TRO68  B000002L7F |  1 |   5 | Music |
| 19 | 078510870X | Ultimate Marvel Team-Up                                                                     |    612475 | 5  0785114572  078511078X  0785114033  0785114041  0785113126 |  4 | 3.5 | Book  |
| 20 | 3895780812 | Computed Tomography : Fundamentals, System Technology, Image Quality, Applications          |    179448 | 5  0865778973  0071343547  0721662854  0721681735  0819444251 |  3 |   5 | Book  |
```

STEP 3

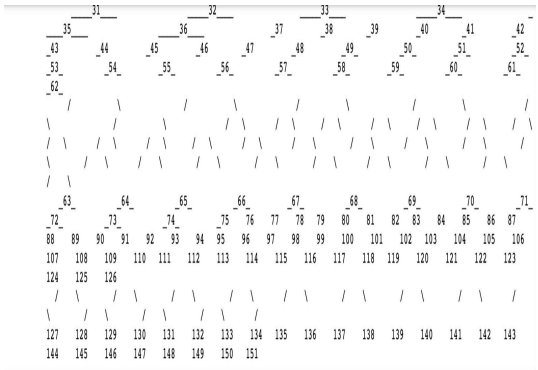
ALGORITHM STRUCTURE

We chose mainly two methods which included developing a hash function implementing and a BST tree parameter to check which out of the two would be better to infuse within the algorithm for faster indexing.

CASE1 : BINARY TREE

- We chose along the binary tree route as it has no ordering constraint.
- Taking upon the in order traversal function of the tree the algorithm developed would help us in faster indexing and searching
- So we add each element to the binary tree, and then we use the inorder traversal to find the ID’s because inorder traversal is the most efficient traversal for a binary tree.

- We later found out that in comparison to the hash function in case two BBinary trees are inefficient.



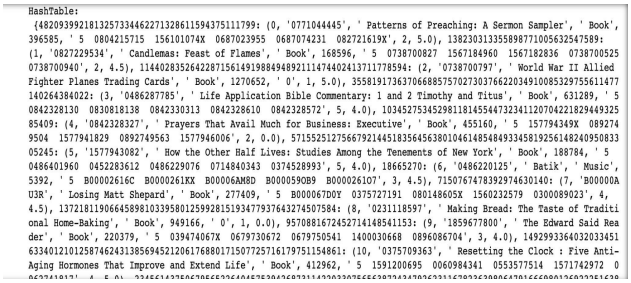
```
from binarytree import build
import time
Idsroot = build(Ids)
# Getting binary tree
print('Binary tree :', root)

# Getting list of nodes
print('List of nodes :', list(root))

# Getting inorder of nodes
print('Inorder of nodes :', root.inorder)
print('Postorder of nodes:',root.postorder)


# Checking tree properties
print('Size of tree :', root.size)
print('Height of tree :', root.height)

# Get all properties at once
print('Properties of tree : \n', root.properties)
```

CASE2 : HASH FUNCTION

With the help of the hashed index based on the primary key of the records we develop an efficient algorithm that combines the efficient data structure to store the prefix, suffix tree.

- The tree used will be a binary tree to make the computation more efficient.
- The hash function that we have developed is based on the pattern of the string, we multiply a prime number 11 with the hash key and add the ascii value of each character in the title, this sum is then, repeatedly added until the whole string is traversed.

```
#HashFunction
hashtable = {}
for i in range(len(title)):
    hashkey = 7;
    for j in range(len(title[i])):
        hashkey = hashkey*11 + ord(title[i][j]);

    hashtable[hashkey] = (Ids[i], ASIN[i],title[i],group[i],salesrank[i], similar[i], categories[i], rating[i])

print('HashTable:\n',hashtable)
```



STEP 4
QUERY DEVELOPMENT AND DEPLOYMENT

- This part mainly involves the deployment of an efficient query over a defined data set.
- This helps us in testing the time of our algorithm for different test cases which would be shown in the result section as we go on further with the project.
- Rather than just having an efficient algorithm, having an efficient query also helps in less time spending.

```
query = "Select Title from books where rating >4"
obj = time.gmtime(0)
epoch = time.asctime(obj)
print("epoch is:", epoch)
mycursor.execute(query)
result3 = mycursor.fetchall()
time_nanosec3 = time.time_ns()
print(time_nanosec3)
print(result)
```

EXPERIMENTAL RESULTS AND FINDINGS

COMPARATIVE RESULT

While comparing we look upon the TIME for different queries upon different defined data with and without deploying the algorithms devised.

QUERY : WITHOUT ALGORITHM

SIZE : 50

We chose this size as it is minimum data to check how the query performs with a small load factor

TIME TAKEN : 1671332524.108181000

SIZE : 100

We chose this as a medium load factor to see how the query would hande medium load factor

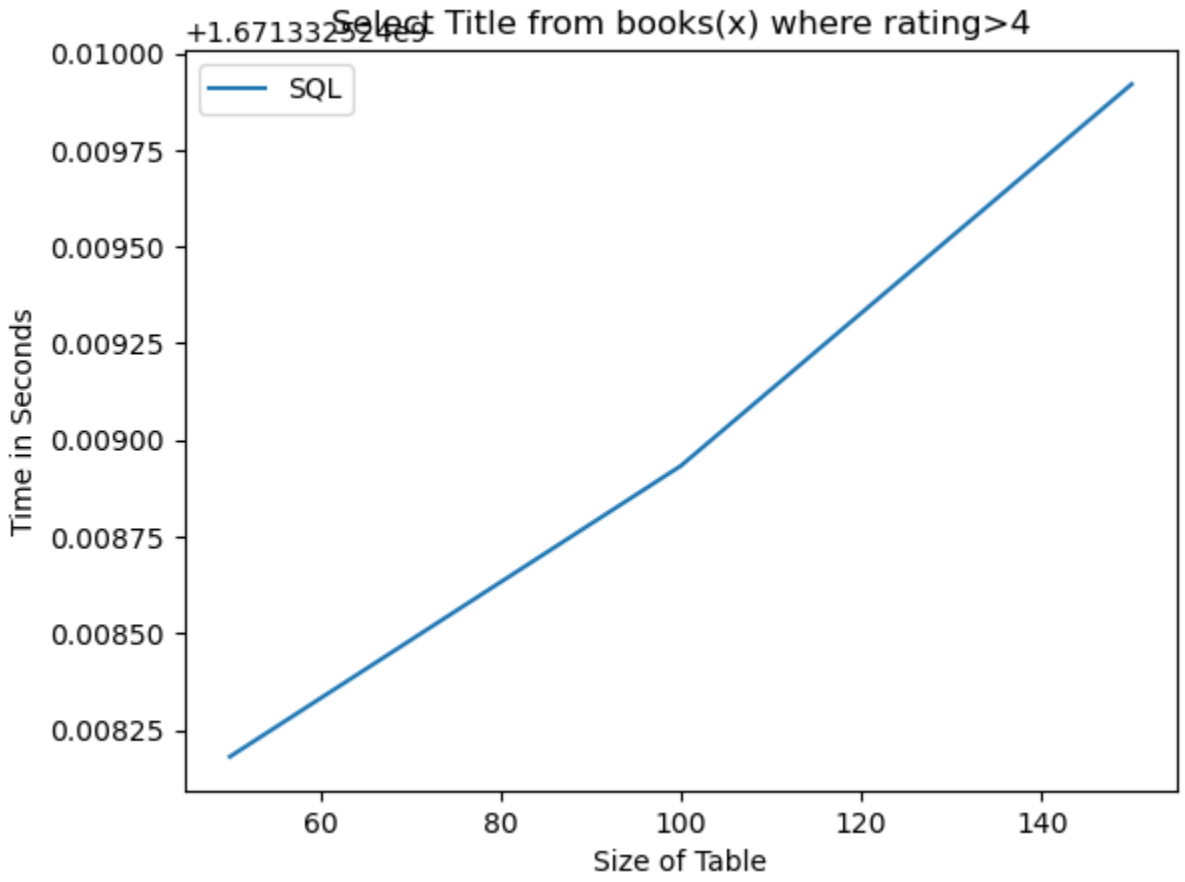
TIME TAKEN : 1671332524.108933000

SIZE :150

We chose this as a large load factor to check the queries performance over a huge load factor

TIME TAKEN : 1671332524.109920000

Below listed is a time performance graph which displays performance over three data sizes defined.



QUERY : WITH ALGORITHM

CASE 1 : BINARY TREE

SIZE : 50

We chose this size as it is minimum data to check how the query performs with a small load factor

TIME TAKEN : 1671332524.271599000

SIZE : 100

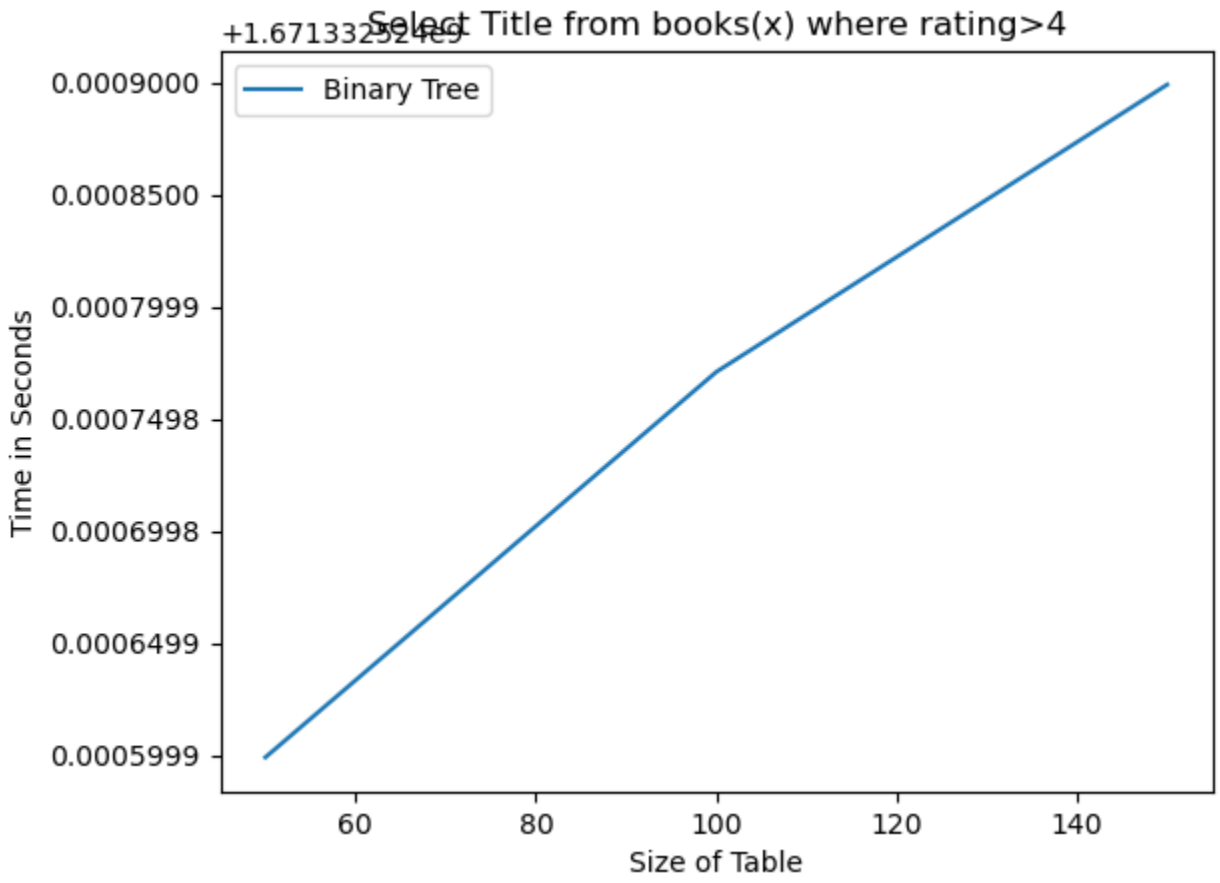
We chose this as a medium load factor to see how the query would hande medium load factor

TIME TAKEN : 1671332524.271771000

SIZE : 150

We chose this as a large load factor to check the queries performance over a huge load factor

TIME TAKEN : 1671332524.271899000



CASE 2: HASH FUNCTION

SIZE : 50

We chose this size as it is minimum data to check how the query performs with a small load factor

TIME TAKEN : 1671332524.193091000

SIZE : 100

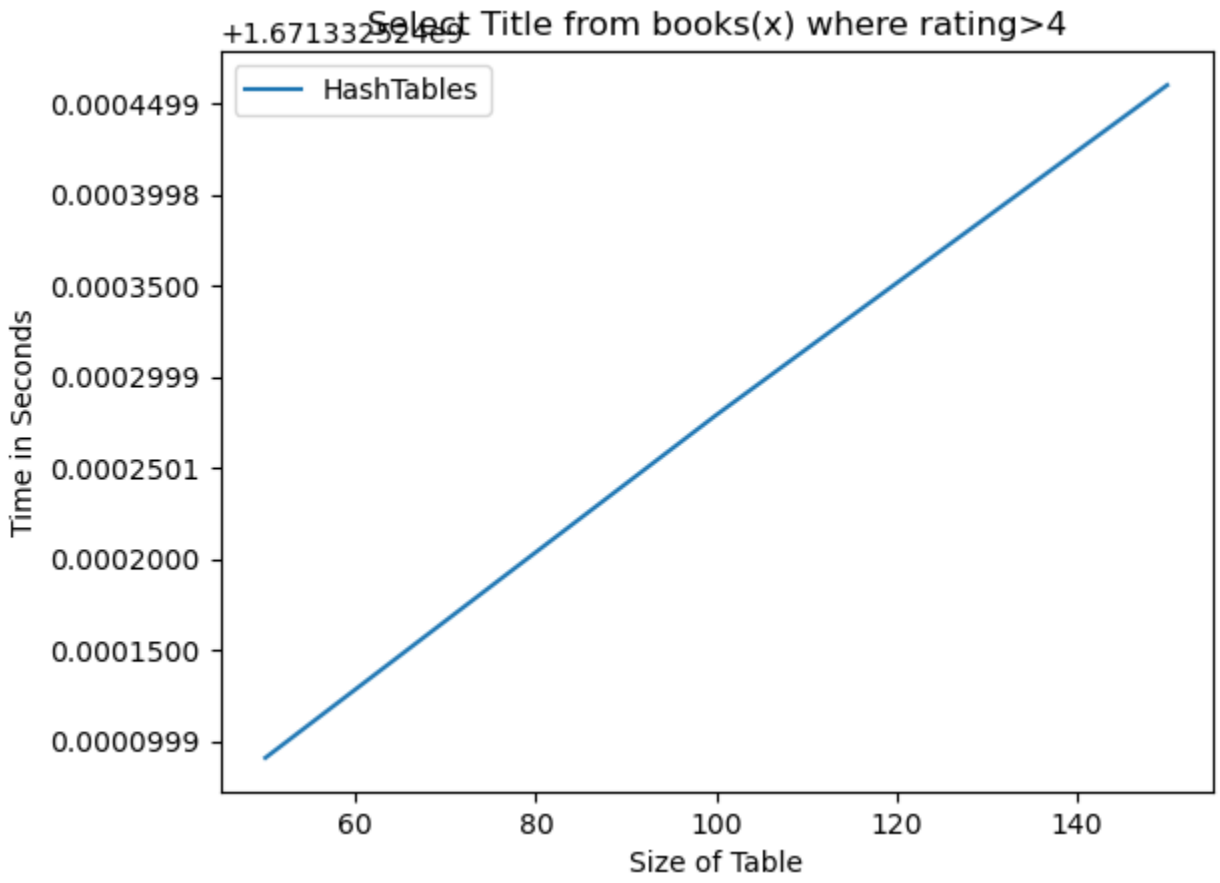
We chose this as a medium load factor to see how the query would hande medium load factor

TIME TAKEN : 1671332524.193091000

SIZE : 150

We chose this as a large load factor to check the queries performance over a huge load factor

TIME TAKEN : 1671332524.193460000



CONCLUSIVE RESULT

Through the comparative results that have been listed above we get to know the performance of the algorithm and the query over a stipulated time.

The query without the algorithm is a very common and conservative way of extracting data from a defined database that has been converted from an unstructured to a structured form that also is being used in the industry for a very long time. We can see from the time listed above how the query performs over a database with a specific search pattern with different loads of data sizes which are namely 50 which is a light load, 100 which is a medium load and 150 which is a heavy load.

The efficient query with the complex algorithm allows room for faster search and indexing which can be seen in the comparative result section from up above through which we get an actual insight what out of the two better gels with the algorithm,binary tree or Hash function with different defined loads of the data set. This offers us a new insight into what can be used in the industry for groundbreaking efficient search with lesser wastage of user data.

CONCLUSION

Through the project we have achieved what we set out for ourselves, a better and efficient algorithm complemented by an enhanced query for faster and better search over a defined dataset. Taking the problem statement that the group came up with and breaking it down into steps provided a clear incentive and goal into what we aimed for and what problem we seeked out to solve. Having developed an algorithm we tested out the search pattern both with and without the algorithm to see how big of a difference it made in our search patterns. Furthermore choosing data loads was highly important for us to see the algorithms performance with heavy sets over a defined time. As we can see the Query with the Binary tree algorithm and by itself offered highly similar results which can be deduced from the graph provided but the performance of the query with the Hash function algorithm was something that behaved and performed well beyond our expectations and offered a search query that was **ONE HUNDRED TIMES** faster than the performance of the query by itself and the performance of the query with the Binary tree. Hence offering us a model that has taken us a step closer to solving the industry's leading problem of time consumption with heavy datasets.