Visvesvaraya Technological University
Belagavi-590 018, Karnataka

A MINI PROJECT REPORT ON
"Indexing for credit card records"
Mini-Project Report submitted in Partial fulfillment of the requirement for the 6th Semester File Structures Laboratory with Mini-Project
[17ISL68]

Bachelor of Engineering
In
Information Science and Engineering

Submitted by

MEGHANA KS [1JT17IS021]

Under the Guidance of
Mr.Vadiraja A
Asst.Professor,Department of ISE

Department of Information Science and Engineering
Jyothy Institute of Technology,
Tataguni, Bengaluru-56008

Jyothy Institute of Technology,

Department of Information Science and Engineering
Tataguni, Bengaluru-560082

CERTIFICATE

Certified that the mini project work entitled **"Indexing"**

carried out by **Meghana K S [1JT17IS021]** bonafide student of Jyothy Institute

of Technology, in partial fulfillment for the award of **Bachelor of Engineering** in

Information Science and Engineering Department of the **Visvesvaraya Technological University, Belagavi** during the year **2020-2021**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said Degree.

Mr.Vadiraja A   **Dr. HarshvardhanTiwari**

Guide,Asst,Professor   Associate Professor andHoD

Dept. OfISE   Dept. OF ISE

External VivaExaminer   Signature with Date:

-
-

ACKNOWLEDGMENT

Firstly, I am very grateful to this esteemed institution Jyothy Institute of Technology for providing me an opportunity to complete my project.

I express my sincere thanks to our Principal Dr. Gopalakrishna K for providing me with adequate facilities to undertake this project.

I would like to thank Dr. Harshvardhan Tiwari, Professor and Head of Information Science and Engineering Department for providing for his valuable support.

I would like to thank my guide Mr. Vadiraja A, Asst. Prof. for his interest and guidance in preparing this work. Finally,

I would thank all our friends who have helped me directly or indirectly in this Project

Meghana K S
1JT17IS021

ABSTRACT

The project is based on " INDEXING" of both Primary Index and Secondary Index

The database used in this project is "CREDIT CARD RECORDS" has been done by using Eclipse IDE with the Windows platform by using "JAVA".

The project main intension is to build indexing using one lakh records, the records is given in the form of "CSV" file format ,In primary Indexing we use primary key has credit card cvv number and in Secondary Index we use credit card holder name has a Secondary key

We are performing Insertion, Deletion, Search and Build index for 1 lakh dataset

The searching is done based on binary search algorithm and Deletion ,Insertion, Build Index is based on primary Key and Secondary Key in the dataset

Primary index is defined on an ordered data file. The data file is ordered on a key field. The key field is generally the primary key of the relation.

Secondary index can be generated from a field which is the candidate key and has a unique value in every record, and the unique key

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

INTRODUCTION

1.1 Introduction to File Structures

A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks. In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by the files creator and user. A File Structure should be according to a required format that the operating system can understand.

 In computing, a file system or file system controls how data is stored and retrieved. Without a file system, information placed in a storage medium would be one large body of data with no way to tell where one piece of information stops and the next begins. By separating the data into pieces and giving each piece a name, the information is easily isolated and identified. Taking its name from the way paper-based information systems are named, each groups of data is called a "file". The structure and logic rules used to manage the groups of information and their name is called a "file system".

There are many different kinds of file systems. Each one has different structure and logic, properties of speed, flexibility, security, size and more. Some file systems have been designed to be used for specific applications. For example, the ISO 9660 file system is designed specifically for optical discs.

 File systems can be used on numerous different types of storage devices that use different kinds of media. The most common storage device in use today is a hard disk drive. Other kinds of media that are used include flash memory, magnetic tapes, and optical discs. In some cases, such as with tmpfs, the computers's main memory (random-access memory, RAM) is used to create a temporary file system for short-term use.

Some file systems are used on local data storage devices; others provide file access via a network protocol(for example, NFS, SMB, or 9P clients). Some file systems are "virtual". Meaning that the supplied "files" (called virtual files) are computed on request(e.g. procfs) or are merely a mapping into a different file system used as a blacking store. The file system manages access to both the content of files and the metadata about those files. It is responsible for arranging storage space; reliability, efficiency, and tuning with regard the physical storage medium are important design

1.2 Introduction to JAVA

• Like any programming language, Java language has its own structure, syntax rules, and programming paradigm. The Java language's programming paradigm is based on concept of OOP, which the language's features support.

• The Java language is a C-language derivative, so its syntax rules look much like C's. For example, code blocks are modularized into methods and delimited by braces and variables are declared before they are used.

• Structurally, Java language starts with packages. A packages is the Java language's namespace mechanism. Within packages are classes, and within classes are methods variables, constants, and more. You learn about the parts of the java language in this tutorial.

1.3 Indexing

Index: A structure containing a set of entries, each consisting of a key field and a reference field, which is used to locate records in a data file.
Key Field: The part of an index which contains keys.

Reference Field: The part of an index which contains information to locate records.
• An index imposes order on a file without rearranging the file.

• Indexing works by indirection.

A Simple Index for Entry-Sequenced Files

An index in which the entries are a key ordered linear list.

• Simple indexing can be useful when the entire index can be held in memory.

• Changes (additions and deletions) require both the index and the data file to

be changed.

• Updates affect the index if the key field is changed, or if the record is moved.

• An update which moves a record can be handled as a deletion followed by an addition.
• Object Oriented Support for Indexed, Entry Sequenced Files

A file in which the record order is determined by the order in which they are entered

• The physical order of records in the file may not be the same as order of entry, because of record deletions and space reuse.

• The index should be read into memory when the data file is opened.

Indexes That are too Large to Hold in Memory

• Searching of a simple index on disk takes too much time.

• Maintaining a simple index on disk in sorted order takes too much time.

• Tree structured indexes such as B-tress are a scalable alternative to simple indexes.

• Hashed organization is an alternative to indexing when only a primary index is needed.

Indexing to Provide Access by Multiple Keys

A search key other than the primary key.

An index built on a secondary key.

• Secondary indexes can be built on any field of the data file, or on combinations of fields.

• Secondary indexes will typically have multiple locations for a single key.

• Changes to the data may now affect multiple indexes.

• The reference field of a secondary index can be a direct reference to the location of the entry in the data file.

• The references fields of a secondary index can also be an indirect reference to the location of the entry in the data file, through the primary key.

• Indirect secondary key references simplify updating of the file set.

• Indirect secondary key references increases access time.

Retrieval Using Combinations of Secondary Keys

• The search for records by multiple keys can be done on multiple index, with the combination of index entries defining the records matching the key combination.

CHAPTER 2
DESIGN

DESIGN

2.1 Algorithm to build a Primary Index and Secondary Index:

Indexing is a way to optimize performance of a file system by minimizing the number of disk accesses required when a query is processed. An index is a data structure which is used to quickly locate and access the data in a disk of the file system.

Indexes are created using some columns in a file:

• The first column is the search key that contains a copy of the primary key or candidate key of the table. These values are stored in sorted order so that the corresponding data can be accessed quickly.

• The second column is the block where that particular key value can be forced.

Clustering index is defined on an ordered data file. The data file is ordered on a non-key field. In some cases, the index is created on non-primary key columns which may not be unique for each record. In such cases, in order to identify the records faster, we will group two or more columns together to get the unique values and create index out of them. This method is known as clustering index. Basically, records with similar characteristics are grouped together and indexes are created for these groups.

The data is sorted according to the search key. It includes sequential file organization. The primary key in data frame is used to create the index. As primary keys are unique and are stored in sorted manner, the performance of searching operation is quite efficient
.
Algorithm:

```
Int insert Sorted (int arr [], int n, int key, int capacity)
If (n >= capacity)
Return n;
Int I;
 For (1=n-1; (I >=0 && arr[i] >key); i--)
    arr [i+1] = arr[i];
    arr [i+1] = key;
Return (n+1);
```

2.2 Algorithm to Sort the Index based on Primary key:

The Insertion sort, although still o (n2) o (n2), works in a slightly different way. It always maintains a sorted a sub_ list in the lower positions of list. Each new item is then "inserted"

Back into the pervious sub -list such that the sorted sub-list is one item larger. We begin by assuming that a list with one item (position 00) is already sorted. On each pass, one for each item 1 through n-1n-1, the current item is checked against those in the already sorted sub-list. As we look back into the already sorted sub -list, we shift those items that are greater to the right when we reach a smaller item or the end of the sub -list, the current item can be inserted.

The implementation of insertion Sort shows that there we are again n-1n-1 passes to sort n items. The iteration starts at position 1 and moves through position n-1n-1, as these are the items the need to be inserted back into sorted sub-lists. Line 8 performs the shift operation that moves a value up one position in the list, making room behind it for the insertion. Remember that this not a complete exchange as was performed in the previous algorithms.

The ,maximum number of comparisons for an insertions sort is the sum of the first n-1n-1 integers. Again this is o(n2)o(n2). However , in the best case, only one comparison needs to be done on each pass. This would be the case for an already sorted list .One note about shifting versus exchanging is also important. In general, a shift operations requires approximately a third of the processing work of an exchange since only one assignment is performed . In benchmark studies, insert ion sort will allow very good performance.

Algorithm:

```
if{ high < low }
 return -1;
  int mid = { low + high} / 2
 if{ key == array[mid] }
 return mid ;

 if{key > array[mid]}
 return binary Search { array, { mid + 1 } , high , key };
return binary Search { array , low, { mid -1} , key};
```

2.3 Algorithm for Searching:

When data items are sorted in a collection such as a list, we say that they hav linear or sequential relationship. Each data items is stored in a postion relative to the others. In java lists, these relative positions are the index values of the individual items. Since these index values are ordered , it os possible for us to visit them in sequence . This process gives rise to our first searching technique, the sequential technique, the SEQUENTIAL SEARCH.

The java implantation for this algorithm is function the needs the list and the items we are looking for and returns a Boolean values as to whether it is present. The Boolean variable found is initialized to False and is assigned the values= True f we discover he items in the list.

Algorithm:

SET Lo to 0

    SET Hi to array length – 1

  WHILE Lo < = Hi

      SET Mid to { Lo + Hi } / 2

   IF X < array [Mid] THEN

SET Hi to Mid -1

      ELSE IF X > array[Mid] THEN

    SET Lo to Mid +1

ELSE
        RETURN Mid
ENDIF
ENDWHILE
RETURN -1

2.4 Calculating time Complexity:

The time complexity of an algorithm is the total amount of time required by an algorithm to complete its execution. In simple words, every piece of code we write, takes time to execute. The time taken by any piece of code to rum is known as the time complexity of that code. The lesser the time complexity, the faster the execution.

If you've programmed a bit before, you're probably wondering how this can be of any sue for you because your program was running fin even when you didn't know all of this time complexity stuff, right? I agree with you 100% but there's a catch.

The time for program to run does not depend solely on efficiency of code, It's also dependent on the processing power of a PC . Since time complexity is used to measure the time for algorithm, the type of algorithm you'd use in small program wouldn't really matter because there's hardly any work being carried out by the processor although when we write code in professional life, the code isn't of 200 or 300 lines.

The time for program to run does not depend solely on efficiency of code. It's also dependent on the processing power of a PC. Since time complexity is used to measure the time for algorithm the type of algorithm you'd use in small program wouldn't really matter because there's hardly any work being carried out by the processor although when we write code in professional life,

the code isn't of 200 or 300 lines.

It's usually longer than a thesis written by a professor and in cases like that , a lot of processor power is being used. if your code is efficient in terms of data structures , you might find yourself in a you might find yourself in a rather sticky situation.

CHAPTER 3

IMPLEMENTATION

IMPLEMENTATION

Indexing Orientations

- If n entries have v possible orientations

```
t=0
for i=1 to n
t = t * v
t = t + or[i]
endfor
return t
```

- To extract the individual orientations again from t < vn-1, use the following code:

```
for i = n to 1
or[i] = t mod v
t = t / v
endfor
```

- Usually there is the constraint that the total twist is 0 modulo v, in other words the orientation of the last entry is dependent on the other n-1. In this case just do not encode the orientation of the last piece, which gives a number between 0 and vn-1-1:

- To extract the individual orientations again from t<vn-1-1, use the following code:

```
s = 0
for I = n-1 to 1
or[i] = t mod v
s = s − or[i]
if s < 0 then s = s + v
t = t / v
endfor
or[n] = s
```

- If n entries can be permuted amongst themselves then their permutation can be encoded in a number between 0 and n!-1. Use a fixed numbering for both the entries and the postitons: the n entries positions are numbered from 1 to n, and the entries are also from 1 to n

```
t = 0;
```

```
for I = 1 to n-1
t = t * (n-i+1)
for j=i+1 to n
if pm[i]>pm[j] then t=t+1
endfor
endfor
retrun t
```

- Note that if all entries are in position then it is encoded as 0. To extract the permutation again from a number t<n! use this:

```
for i=n-1 to 1
pm[i]=1 +(tmod(n-i+1))
t=t/(n-i+1)
for j= i+1 to n
if pm[j]>=pm[i] then pm[j]=pm[i]+1
endfor
endfor
```

- Often there is the constraint that the permutation must have even parity. This means that the position of the last two entries is dependent on the other n-1. In this case just do not encode the position of the last two entries, which gives a number between 0 andn!2-1:

```
t=0;
for i=1 to n-2
t=t*(n-i+)
for j=i+1 to n
if pm[i] > pm[j] then t=t+
endfor
endfor
```

- To extract the even permutation again from a number t>n!/ use this

```
pm[n-1] =1
s=0
for i=n-2 to 1
pm[i]=1 + { t mod (n-i+1)}
s=s+pm[i]-1
t=t/{n-i+1}
for j=i+1 to n
if pm[j] >=pm[i] then pm[j] = pm[j] +1
```

```
      endfor
      endfor
      if s mod 2=1 then swap pm[n],pm[n-1]

      Indexing combinations

      t=0,r=m
      t=0
      r=m
      for i=n-1 to 0
      if cm[i+1] =p then
      t=t+ comb(i,r)
      r=r-1
      endif
      next
      return t
      r=m
```
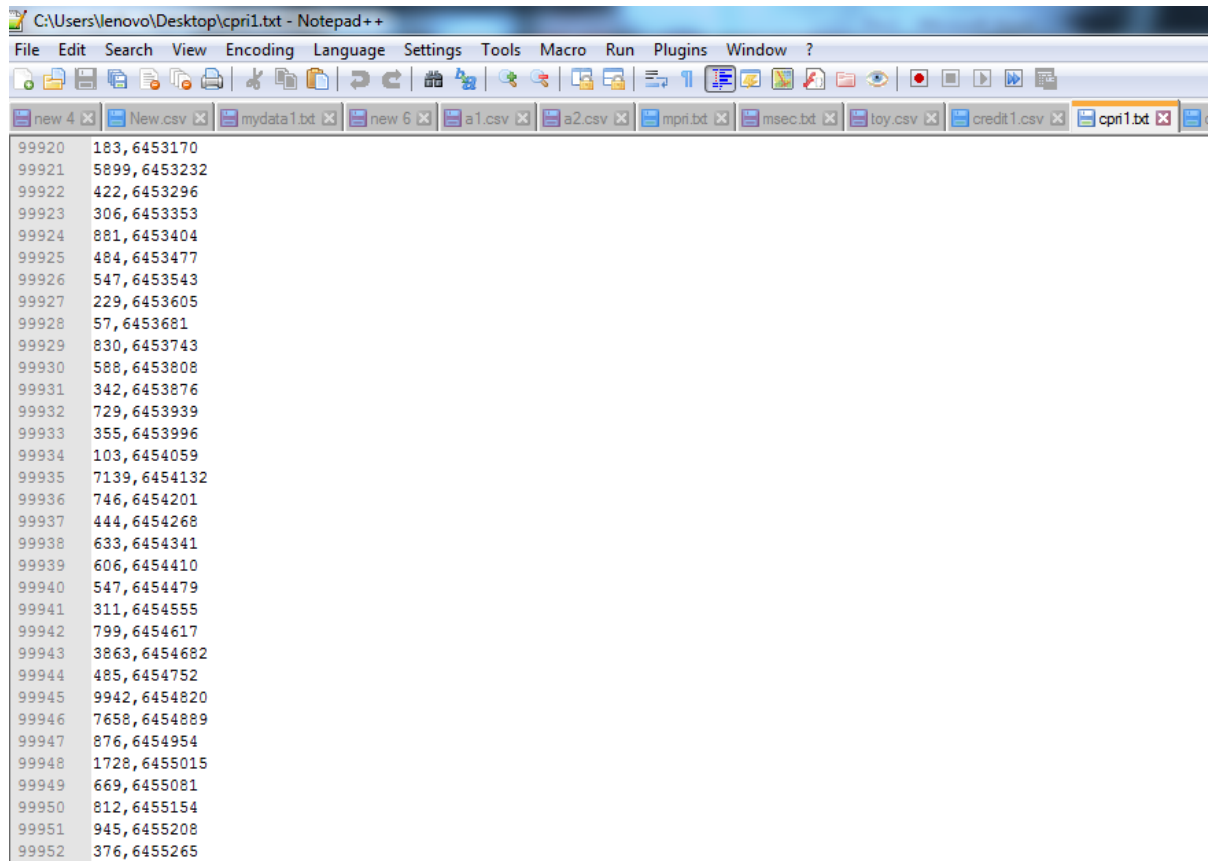
CHAPTER 4

RESULT AND ANALYSIS

RESULTS AND SNAPSHOTS


```
C:\Users\lenovo\Desktop\credit1.csv - Notepad++
File  Edit  Search  View  Encoding  Language  Settings  Tools  Macro  Run  Plugins  Window  ?

new 4   New.csv   mydata1.txt   new 6   a1.csv   a2.csv   mpri.txt   msec.txt   toy.csv   credit1.csv

99978   7681,TeresaDMorin,AmericanExpress,344000000000000,1032,,142100,/n
99979   909,BerniceBorel,MasterCard,5160000000000000,1004,,19500,/n
99980   775,ElsieWChang,DinersClubInternational,30200000000000,9108,,23500,/n
99981   453,JacksonVPehrson,JapanCreditBureau,3570000000000000,2613,,45600,/n
99982   819,JeffreyMCastillo,DinersClubInternational,30300000000000,3022,,198200,/n
99983   895,WaldoIPurdy,DinersClubInternational,30500000000000,5894,,21900,/n
99984   336,AlvinHamilton,JapanCreditBureau,3550000000000000,4350,,122300,/n
99985   689,RaymondHassett,JapanCreditBureau,3540000000000000,9641,,49400,/n
99986   967,CatherineLewis,MasterCard,5170000000000000,6168,,94200,/n
99987   1210,EugenaUStrock,AmericanExpress,349000000000000,6264,,89500,/n
99988   222,SheilaBerry,Discover,6230000000000000,6321,,130100,/n
99989   387,MauroMolinar,Discover,6010000000000000,6992,,12100,/n
99990   401,ChrisRichardson,Visa,4040000000000000,6823,,104900,/n
99991   286,RoderickMarkert,Discover,6010000000000000,8677,,27600,/n
99992   201,JuneStephens,Visa,4900000000000000,8449,,72600,/n
99993   483,DerekEKeffer,MasterCard,5450000000000000,5005,,11100,/n
99994   157,LuisPGray,MasterCard,5300000000000000,2092,,105700,/n
99995   990,ByronBBurrus,JapanCreditBureau,3580000000000000,7444,,79100,/n
99996   470,PaulineIMacias,JapanCreditBureau,3540000000000000,930,,183600,/n
99997   3320,LouisaBurrows,AmericanExpress,371000000000000,3719,,69400,/n
99998   326,ErinZCastillo,MasterCard,5100000000000000,9987,,134300,/n
99999   369,EzequielVBischof,Discover,6010000000000000,4150,,190600,/n
100000  477,CarolynACooper,MasterCard,5440000000000000,2133,,64700,/n
100001  2463,HaroldGoldsmith,AmericanExpress,378000000000000,8648,,16500,/n
100002  489,GlenLBrown,AmericanExpress,376000000000000,2088,,48700,/n
100003  *301,MajorSpeciale,AmericanExpress,348000000000000,9686,,87200,/n
100004  626,JoseChang,Discover,6230000000000000,2508,,172800,/n
100005  483,DebbieLetchworth,Discover,6010000000000000,9068,,156700,/n
100006  370,JaneORiggs,DinersClubInternational,30500000000000,8692,,183900,/n
100007  106,AlexisVMunford,MasterCard,5150000000000000,8783,,153900,/n
100008  3898,VeronicaOFlynn,AmericanExpress,372000000000000,991,,124500,/n
100009  37,PamellaFGuarino,Discover,6010000000000000,8795,,83500,/n
100010  693,RichardTHudson,Visa,4870000000000000,6475,,154600,/n
```

Fig4.1: CREDITCARD MANAGEMENT SYSTEM

The above dataset consists of cvv, holder name, card type full name, card number, card pin, credit limit. This is the dataset which is used and further get indexed.

INDEXING ON CREDITCARD MANAGEMENT SYSTEM

Fig 4.2 index build for creditcard dataset using primary key cvv

The index is built for the primary key cvv in the dataset

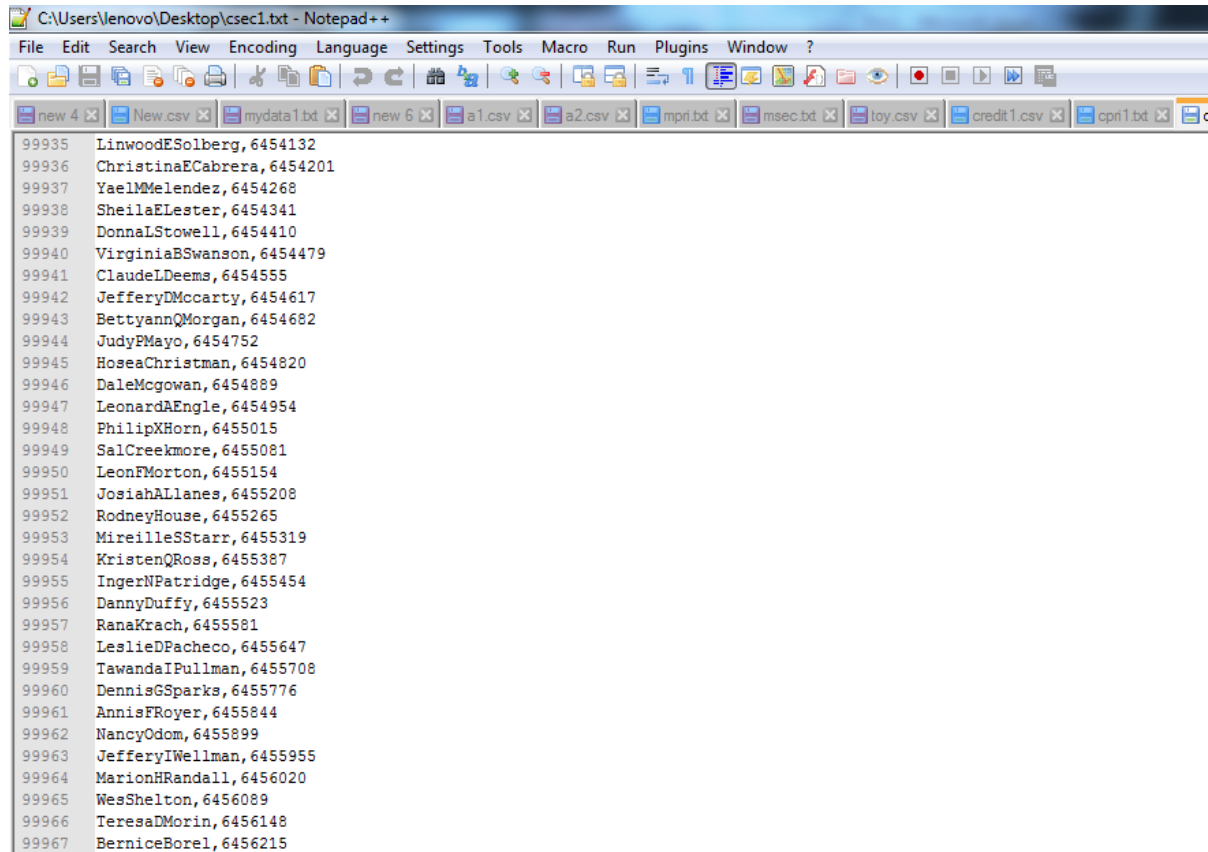# INDEXING ON CREDITCARD MANAGEMENT SYSTEM

Fig 4.3 index build for creditcard dataset using secondary key(holder name)

The index is built for the secondary key holder name in the dataset

```
###########################################
Welcome
_____
Enter your Valuable choice:
###########################################
1>Enter details:
2>Enter the key to Search:
3>To Build Index of my Data:
4>Enter the key to be Deleted:
5>Exit
###########################################
1
Enter the cvv:
202
Enter the primary key:
sneha
Enter the cardtypefullname:
sbi
Enter the cardnumber:
67873509900
Enter the cardpin:
907
Enter the creditlimit:
564778000
total records100002
6381328536msec
###########################################
Welcome
_____
Enter your Valuable choice:
###########################################
```

Fig4.4 Insertion of data in Primary key

```
###########################################
Welcome
_____
Enter your Valuable choice:
###########################################
1>Enter credit card details:
2>Enter the key to Search:
3>To Build Index of my Data:
4>Enter the key to be Deleted:
5>Exit
###########################################
2
Enter the cvv to search:
2463
100001
unqkey: 2463
holdername: GarlandGMaxon
cardtypefullname: AmericanExpress
cardpin: 375000000000000
creditlimit: 8844
16811msec
###########################################
Welcome
_____
Enter your Valuable choice:
###########################################
1>Enter credit card details:
2>Enter the key to Search:
3>To Build Index of my Data:
4>Enter the key to be Deleted:
5>Exit
###########################################
```

Fig 4.5 Searching of primary key in data(CVV)

```
                        _____
Enter your Valuable choice:
###############################################
1>Enter credit card details:
2>Enter the key to Search:
3>To Build Index of my Data:
4>Enter the key to be Deleted:
5>Exit
###############################################
4
total records100001
Enter the cvv to delete:
477
Done
Deleted
total records100000
850331msec
###############################################
Welcome
                        _____
Enter your Valuable choice:
###############################################
1>Enter credit card details:
2>Enter the key to Search:
3>To Build Index of my Data:
4>Enter the key to be Deleted:
5>Exit
###############################################
```

Fig 4.6 Deletion of primary key in data(CVV)

```
###############################################
Welcome
                        _____
Enter your Valuable choice:
###############################################
1>Enter credit card details:
2>Enter the key to Search:
3>To Build Index of my Data:
4>Enter the key to be Deleted:
5>Exit
###############################################
3
46592msec
###############################################
Welcome
                        _____
Enter your Valuable choice:
###############################################
1>Enter credit card details:
2>Enter the key to Search:
3>To Build Index of my Data:
4>Enter the key to be Deleted:
5>Exit
###############################################
5
Exiting..
```

Fig 4.7 Building a index for Primary Key

```
   _____
   Enter your Valuable choice:
   ##########################################
   1>Enter credit card details:
   2>Enter the key to Search:
   3>To Build Index of my Data:
   4>Enter the key to be Deleted:
   5>Exit
   ##########################################
   1
   Enter unique key:
   50
   Enter holdername:
   janvi
   Enter cardtypefullname:
   bankofindia
   Enter cardnumber:
   54679357
   Enter cardpin:
   5466
   Enter creditlimit:
   55465657
   total records100001
   406808msec
   ##########################################
   Welcome
   _____
   Enter your Valuable choice:
   ##########################################
   1>Enter credit card details:
   2>Enter the key to Search:
   3>To Build Index of my Data:
   4>Enter the key to be Deleted:
   5>Exit
   ##########################################
```

Fig4.8 Insertion of data in Secondary key

```
   ##########################################
   Welcome
   _____
   Enter your Valuable choice:
   ##########################################
   1>Enter details:
   2>Enter the key to Search:
   3>To Build Index of my Data:
   4>Enter the key to be Deleted:
   5>Exit
   ##########################################
   2
   Enter the holdername to search:
   LouisaBurrows
   cvv: 3320
   prikey: LouisaBurrows
   cardtypefullname: AmericanExpress
   cardnumber: 371000000000000
   cardpin: 3719
   creditlimit: 69400


   156579383msec
   ##########################################
   Welcome
```

Fig 4.9 Searching of Secondary key in data(Holder Name)

```
################################################
Welcome
_____
Enter your Valuable choice:
################################################
1>Enter details:
2>Enter the key to Search:
3>To Build Index of my Data:
4>Enter the key to be Deleted:
5>Exit
################################################
4
total records100000
Enter the holdername name to delete:
DonaldNGallagher
holdername: 795
cvv: DonaldNGallagher
cardtypefullname: Visa
cardnumber: 4370000000000000
cardpin: 1903
creditlimit: 196200


Do You Want To delete This Record ?(y/n)
y
Done
Deleted
```

Fig 4.10 Deletion of Secondary key in data(Holder Name)

```
################################################
Welcome
_____
Enter your Valuable choice:
################################################
1>Enter details:
2>Enter the key to Search:
3>To Build Index of my Data:
4>Enter the key to be Deleted:
5>Exit
################################################
3
265370723msec
################################################
Welcome
_____
Enter your Valuable choice:
################################################
1>Enter details:
2>Enter the key to Search:
3>To Build Index of my Data:
4>Enter the key to be Deleted:
5>Exit
################################################
```

Fig 4.11 Building a index for Secondary Key

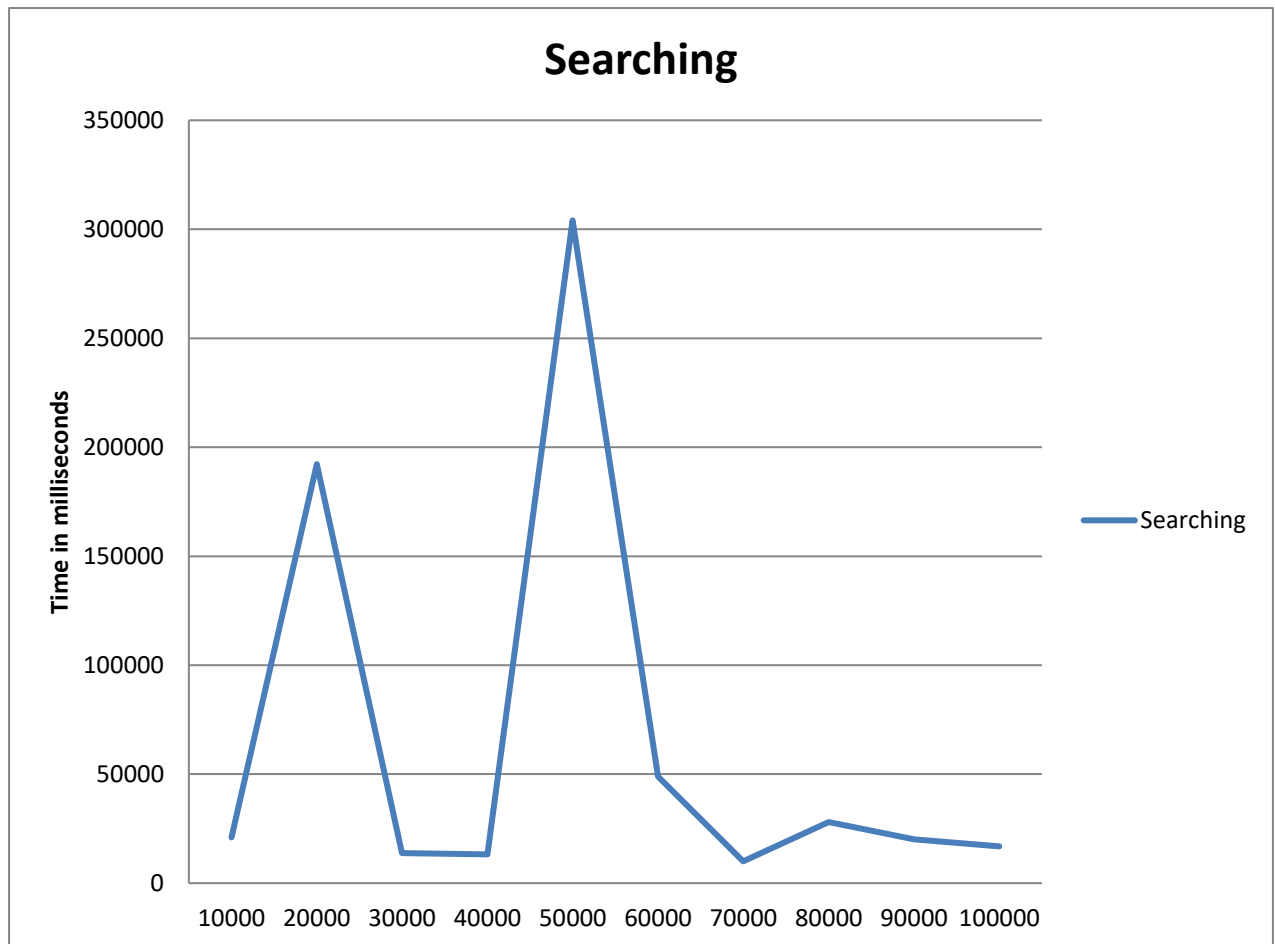# Time Analysis For Inserting the Record in primary key

INSERTION
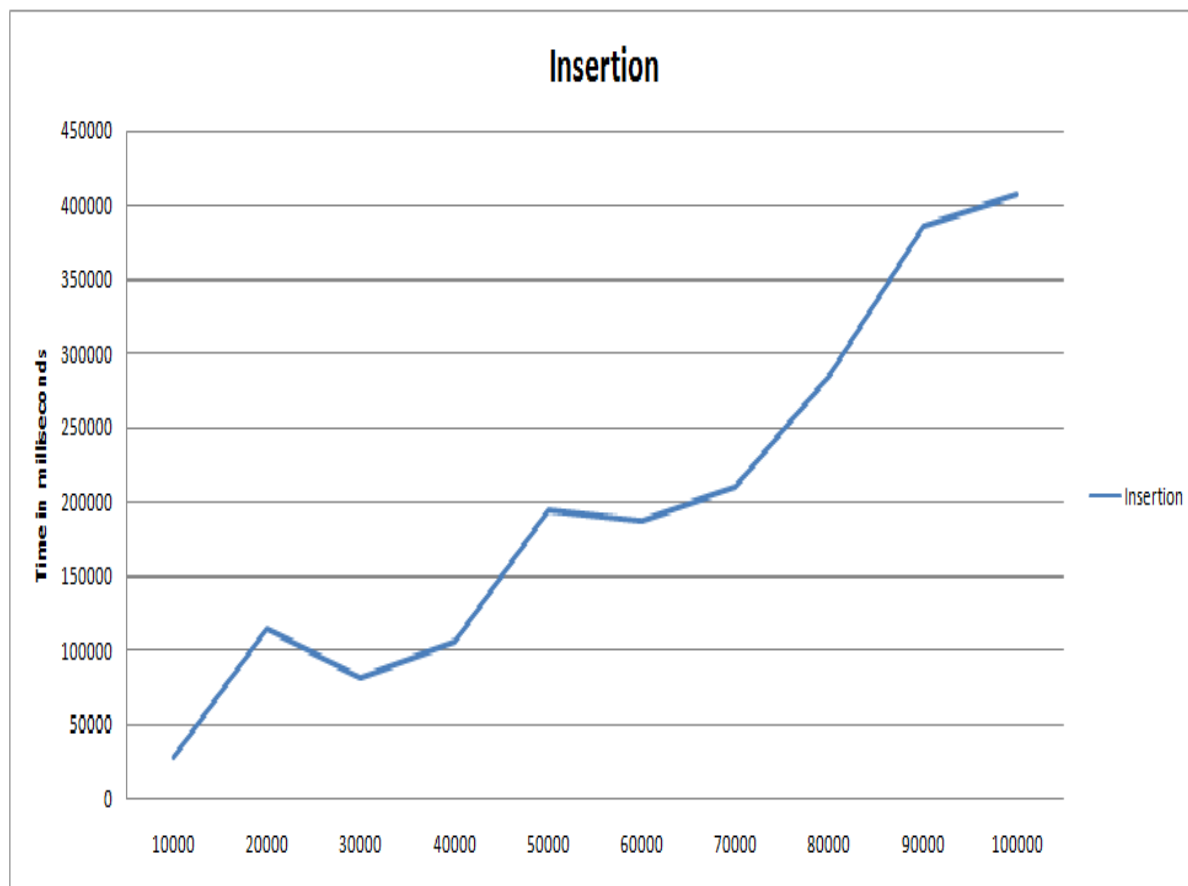
The time taken to insert a record into a data file, as the number of record increases the time also increases.

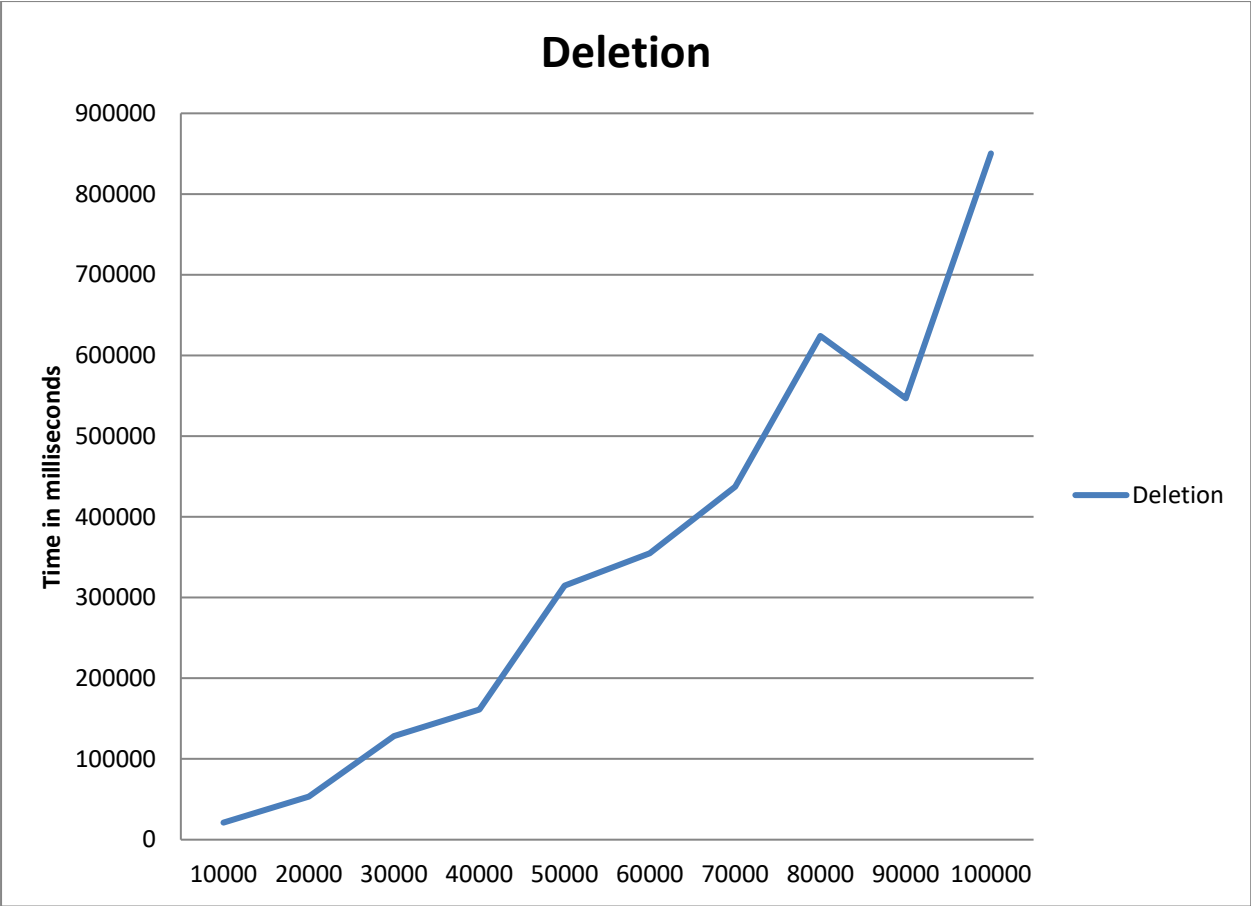# Time analysis for searching a primary key record

SEARCHING

The time taken to search the record in a data file, as the number of record increases the time will delayed to search of record.

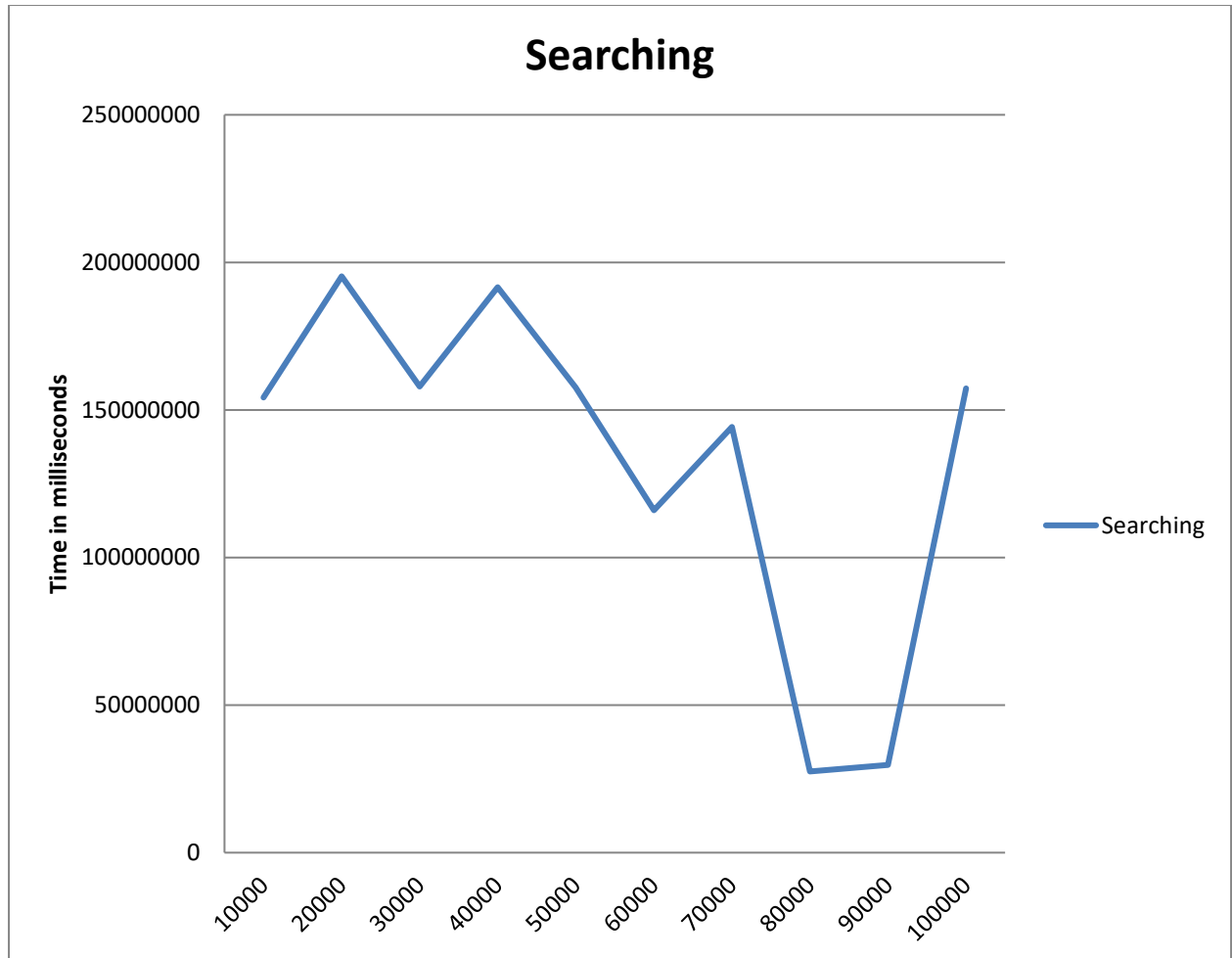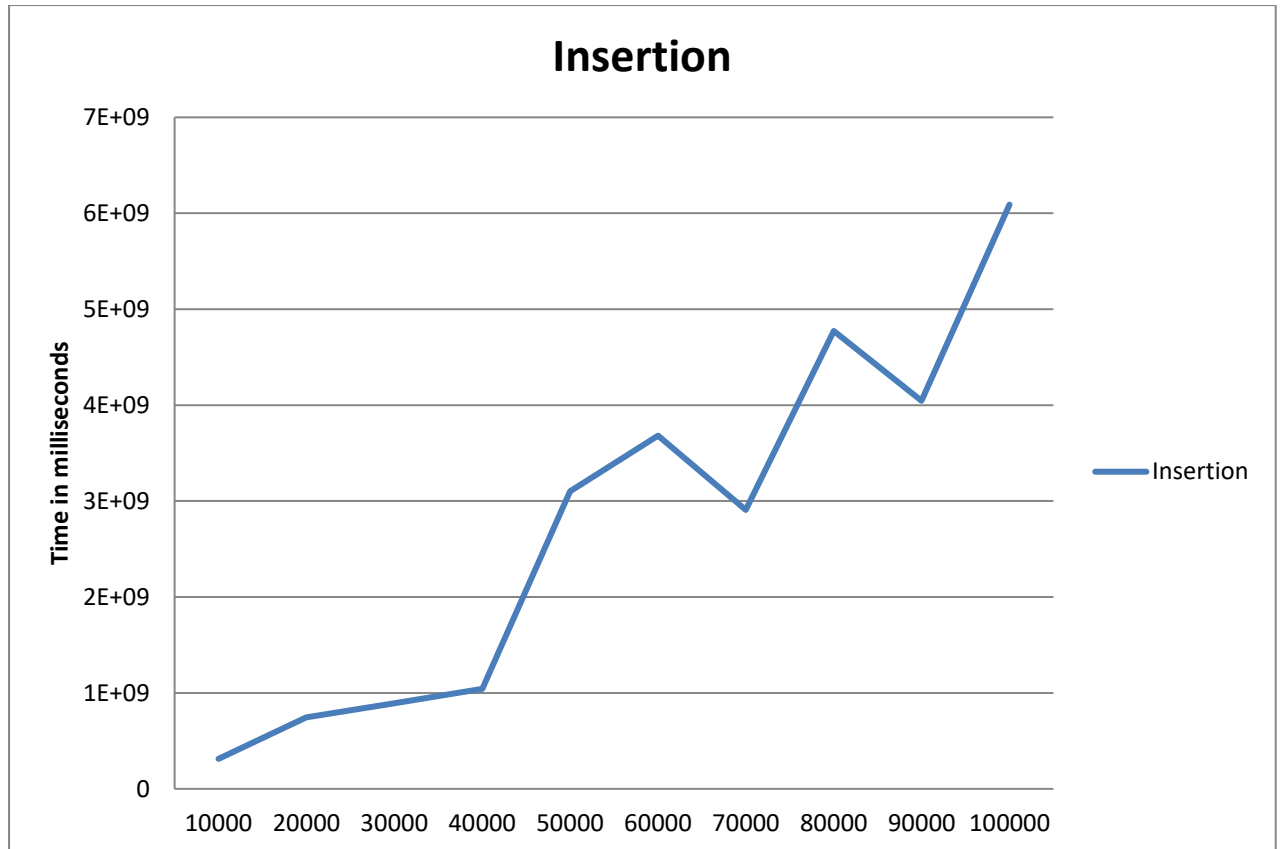# Time Analysis for Deleting a primary key record

DELETION

The time taken to delete from the data file, decreases as the number of record increases.

Time Analysis for searching a secondary key record

SEARCHING

The time taken to search the record in a data file, as the number of record increases the time will delayed to search of record.

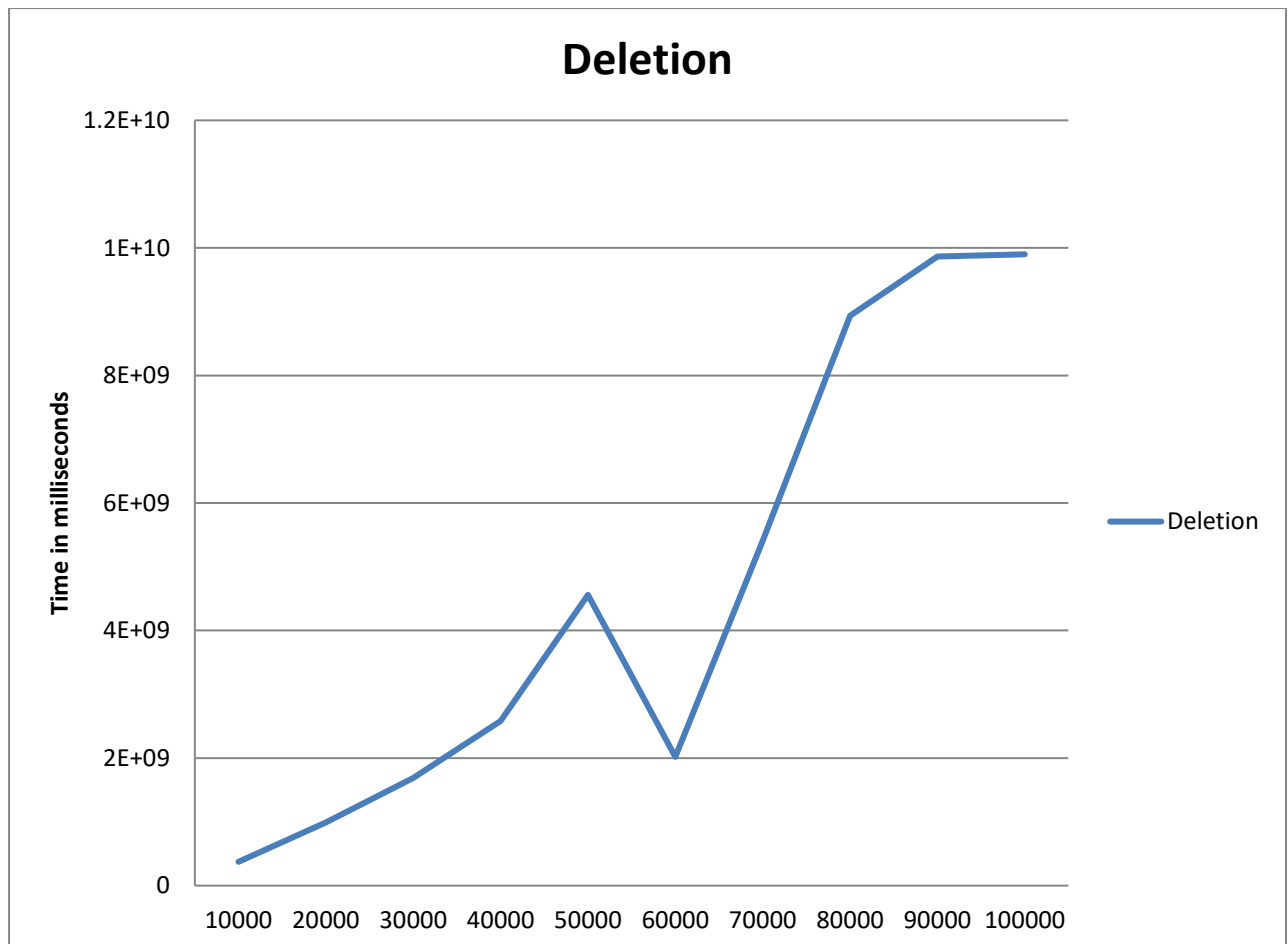## Time Analysis for inserting a secondary key record

INSERTION

Insertion

The time taken to insert a record into a data file, as the number of record increases the time also increases.

Time Analysis for Deleting a secondary key record

DELETION

Deletion

The time taken to delete from the data file, decreases as the number of record increases.

CONCLUSION

We have successfully used various functionalities of JAVA and created the file structures.

- Indexes form an important part of designing, creating and testing information.

- Users search in a hurry for information to help them and give up after two or three tries.

- An index can point the way in harmony with user expectations or not.

- Indexing is an interactive analysis and creative process throughput the entire documentation.

View tables are used to display all the components at once so that user can see all the components of a particular type at once. One can just select the component and modify and remove the component.

Features:

1. Clean separation of various components to facilitate easy modification and revision.

2. All the data is maintained in a separate file to facilitate easy modification.

3. All the data required for different operations is kept in a separate file.

REFERENCES

1,www.geeksforgeeks.org

File structure and its operations

2. https://www.w3schools.com
Basic functions involved in file structure

3. www.tutorialspoint.com
Linking our requirement with the concept

4. https://gateoverflow.in
Insertion and deletion for indexing

5. www.caveofprogramming.com
Searching for content using secondary index

6. www.javacodegeeks.com
Functions implementation

7. Michael Folk, Bill Zoellick, Greg Riccardi File Structures
An Object Oriented approach Overall structure of indexing

8. Raghu Ramakrishnan and Johannes Gehrke: Database Management
Systems, 3rd Edition, McGraw Hill, 2003