This operation is basically an intersection. Consider following lists

List 1	List 2
Adams	Adams
Carter	Anderson
Chin	Andrews
Davis	Bech
Foster	Burns
Garwick	Carter
	Davis
	Dempsey

Initializing : we need to arrange things in such a way that the procedure gets going properly.

Getting & accessing the link next time in list : we need simple methods support getting the next list element & accessing it.

Synchronizing : we have to make sure that the current item from one list is never so far ahead of current item on the list that a match will be missed.

Handling end-of-file conditions - when we get to the end of either list1 or list2, we need to halt the program.

Recognising errors : when an error occurs in data, we want to detect it & take some action.

If item(1) is lessthan item(2). we get the next item in list 1;

If item(1) is greater than, we get next item in list 2;

else get next items from both the lists

The key feature of this algorithm is that control always returns to the head of main loop after every step of operation.

We can setup an unbounded no. of different lists, each varying length, without creating a large no. of small files using linked lists. We could redefine our secondary index so it consists of records with two fields - a secondary key field & a field containing relative rrn of first corresponding primary key reference (label 90) in the inverted list.

---

① Explain matching procedure of 2 sequential process with algorithm/ pseudocode.

Ans

```
int Match (char *List1Name, char *List2Name, char
           *OutputListName)
{  int MoreItems;
   InitializeList(1, List1Name);
   InitializeList(2, List2Name);
   InitializeOutput(OutputListName);
   MoreItems = NextItemInList(1) && NextItemInList(2);
   while (MoreItems)
   {   if (Item(1) < Item(2))
           MoreItems = NextItemInList(1);
       else if (Item(1) == Item(2))
       {   ProcessItem(1);
           MoreItems = NextItemInList(1) &&
                       NextItemInList(2);
       }
       else
           MoreItems = NextItemInList(2);
   }
   FinishUp();
   return 1;
}
```

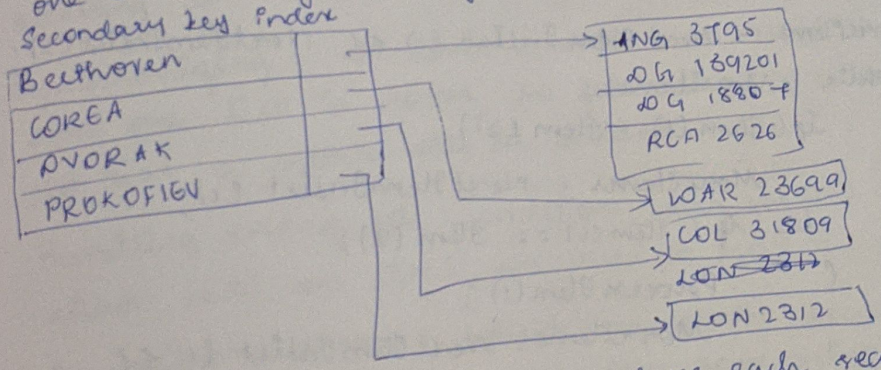
② Explain ways to improve secondary indexing.

Beethoven	ANG3795	DG139201	DGL8807	RCA2626

a) Array of references: With this, major contribution of this revised index structure is its help in solving the problem of rearranging index file every time a new record is added to data file. Consider a new record being added PROKOFIEV ANG 36193 LON2312,

Since we aren't adding another record to index, there is no need to rearrange any records. All that is required is rearrangement of fields in existing recording for PROKOFIEV.

b) Inverted list
List containing set of secondary key which leads to one or more primary key is called inverted list



This is an ideal situation where each secondary key point to diff lists of primary key. The list could contain hundreds of references, if needed, while still requiring only one instance of a secondary key. On the other han if a list requires only one element, then no space lost to internal fragmentation.

What is 2° indexing? What are the advantages?

Ans The search for records by multiple key can be done on multiple indexes, with the combination of index entries defining the records matching the key combination. If two keys are to be combined, a list of entries from each key is retrieved.

- for an "or" combination of keys, lists are merged ie, any entry found in either list matches the search.
- for an "and" combination of keys, the lists are matched, ie. only entries found in both lists match the search.

Selective Indexes: Another interesting feature of secondary indexes is that they can be used to divide a file into parts & provide a selective view. For example, it is possible to build a selective index that contains only the title of classical recordings in the record collection. If we have additional information about the recordings in data file such as date the recording was released, we could build selective indexes such as "recordings released prior to 1970" & "recording since 1970". Such selective index information could be combined into Boolean & operations to respond to requests such as "List all the recordings of Beethoven's Ninth Symphony released since 1970". Selective indexes are sometimes useful when contents of a file fall naturally & logically into several broad categories.

```
void main()
{ int ch, flag;
in.initial();
sind.initial();
for(;;)
{ cout << 1-add, 2-display, 3-search, 4-delete, 5-exit";
  cin >> ch;
  switch(ch)
  { case 1: cout << "enter details:";
            s.add();
            in.write();
            sind.swrite();
            break;
    case 2: opener(dfile, datafile, ios::in);
            cout << "indexfile, secondary & datafile";
            s.datadisp();
            break;
    case 3: cout << "To search based on sec key";
            flag = sec_search();
            if(flag == -1)
              cout << "no data record";
            break;
    case 4: flag = sec_search();
            if(flag == -1)
              cout << "no data record found";
            else
            { s.remove();
              in.write();
              sind.swrite();
            }
            break;
    default: exit(0);
  }
  dfile.close();
  ifile.close();
  sfile.close();
}
```

```
for (i = spos ; i < sindsize - 1; i++)
        sid[i] = sid[i+1];
sindsize--;
} else {
    cout << "usn doesn't match";
}

void student :: dataDisp ()
{
    cout << endl << "Index file details :" << endl;
    cout << "\t" << "usn" << "\t" << "address";
    for (i = 0; i < indsize ; i++)
        cout << id[i].usn << id[i].addr ;
    cout << endl;
    for (i = 0; i < sindsize ; i++)
        cout << sid[i].name << sid[i].susn ;
    while (dfile)
    {   unpack ();
        cout << dusn << name << age ;
        cout << branch << sem ;
    }
}
void student :: unpack()
{ dfile.getline (dusn, 15, '|');
  dfile.getline (name, 20, '|');
  dfile.getline (age, 5, '|');
  dfile.getline (branch, 6, '|');
  dfile.getline(sem, 5, '|');

}
```

```
strcpy-s (id[i].usn, dusn);
-itoa-s(k, id[i].addr, 10);
indsize++;
for (i = indsize; i >0; i--)
{ if (strcmp(name, sid[i-1].sname) <0)
        sid[i] = sid[i-1];
  else if ((strcmp(name, sid[i-1].sname) == 0) &&
      (strcmp(dusn, sid[i-1].susn) <0))
        sid[i] = sid[i-1];
  else
      break;
}

strcpy-s (sid[i].sname, name);
strcpy-s (sid[i].susn, dusn);
indsize++;
}
void student::pack()
{  strcpy-s (buffer, dusn);
   strcat-s(buffer, ":| "); strcpy-s(buffer, name)
strcats-(buffer, age); strcat s(buffer, "|");
strcat-s(buffer, branch); strcat-s(buffer, "|");
strcat-s(buffer, sem); strcat-s(buffer, "|");
}
int search (char *susn)
{ int low=0, high=indsize-1, mid;
while (low <= high)
{   mid = (low+high)/2;
   if (strcmp(susn, id[mid].usn == 0)
        return mid;
```

```
void index :: write ()
{ opener (ifile, indexfile, ios :: out);
    for (i=0; i< indsize ; i++)
        ifile << id[i].usn << '|' << id[i].addr << "\n";
}
void index :: rewrite ()
{ opener (rifile, indexfile, ios :: out);
    for (i=0; i < indsize ; i++)
        rifile << id[i].rname << "|" << id[i].usn << "\n";
}
void student :: add ()
{ int k;
  cout << "enter the usn no = "; cin >> dusn;
  if (search dusn >=0)
  { cout << " usn already present";
  for (i = indsize ; i >0; i--)
  { if (strcmp (dusn, id[i-1].iusn)<0)
      id[i] = id[i-1];
    else
      break;
  }
  opener (dfile, datafile, ios :: app);
  cout << "enter name"; cin >> name;
  cout << "age"; cin >> age;
  cout << "branch"; cin >> branch;
  cout << "sem"; cin >> sem;
  pack();  dfile.seekg(0, ios::end);
  k = dfile.tellg();
```

```
class index
{
  public:
        char sname[20], snum[5];
        void initial();
        void write();
} ind, sid[30];
void index::initial()
{
  ifile.open(indexfile, ios::in);
  if (!ifile)
  {
    indsize = 0; return; }
  for (indsize=0; ; indsize++)
  {
    ifile.getline(sid[indsize].snum, 15, '|');
    ifile.getline(sid[indsize].addr, 5, '\n');
    if (ifile.eof())
        break;
  }
  ifile.close();
}
void sindex::initial()
{
  sifile.open(sindexfile, ios::in);
  if (!sifile)
  {
    sindsize = 0;
        return;
  }
  for (sindsize=0; ; sindsize++)
  {
    sifile.getline(sid[sindsize].sname, 20, '|');
    sifile.getline(sid[sindsize].snum, 5, '\n');
    if (sifile.eof())
        break;
  }
  sifile.close();
}
```

Lab program 6

```
#include <fstream>
#include <iostream>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#define datafile "stu6.txt"
#define index "prg6.txt"
#define tempfile "tmp6.txt"
using namespace std;
fstream dfile, ifile, tfile;
int i, indsize, tindsize;
char buffer[100], skey[20];
void opener (fstream &file, char *fn, int mode)
{ file.open (fn, mode);
  if (!file)
  { cout << "unable to open file";
    getch();
    exit(1);
  }
}
class student
{ char usn[15], name[20], age[5], branch[6], sem[
  public: void add();
          void pack();
          void remove();
          friend int search (char *);
          void data disp();
          void unpack();
};
class index
{ public: char iusn[15], addr[5];
          void inread();
          void iwrite();
} id[50], tid;
```

e) Explain all the operations required to maintain an index file

a) Create original empty index & datafile.
   index (fixed length) datafile (variable length)

b) Load index to main memory
   Load (read), Store (write), pack, unpack

c) Rewriting index file from memory.

d) Record Addition (Requires some rearrangement of index).

e) Record Deletion
   Mark deleted record in both.

f) Record Updation
   - update changes the value of key field
   - Update doesn't affect key field

① What are the advantages of primary indexing?

a) Associates a fixed length record to variable length

b) Sorting & maintaining index files is less expensive as compared to maintaining a data file

c) Handles pinned records

```
case 3: cout << " Enter USN to be searched";
        cin >> susn;
        flag = search (susn);
        if (flag == -1)
            cout << " record not found ";
        else
            s. recDisp (flag);
            break;
case 4: cout << " Enter usn "
        cin >> susn;
        pos = search (susn);
        if (pos == -1)
        {cout << " Usn No not found in "; }
        else { s. remove (pos); }
        in. write ();
    }
    break;
    default : exit (0);
    }
    stdfile. close ();
  }
}
```

```
    stdfile.getline(branch, 6, '|');
    stdfile.getline(sem, 5, '|');
}
void Student::dataDisp()
{   while(1)
    {   unpack();
        if(usn[0] == '$')
        {   cout<<"record deleted";}
        else
            cout <<usn<< name << age << branch<< sem;
            if(stdfile.eof())
                break;
    }
}
void main()
{   int ch, pos, flag; char usn[15];
    Student S;
    fin.initial();
    for(;;)
    {   cout << endl << "1.Rec entry \n 2. Display \n
        3. search \n 4. Delete \n 5. Exit \n";
        cin >> ch;
        switch(ch)
        {   case 1: cout << "Enter Details";
            S.add()
            fin.write()
            break;
            case 2: opener(stdfile, datafile, ios::in);
            S.dataDisp();
            cout<< "Order file details";
            for(i=0; i< index; i++)
            {   cout << id[i].usn << id[i].addr;
                break;
```

```
    if (strcmp (usn, id[mid].usn) == 0)
        return mid;
    else if (strcmp (usn, id[mid].usn) > 0)
        low = mid + 1;
    else
        high = mid - 1
    }
    return -1;
}
void Student :: recDisp (int pos)
{ opener (stdfile, datafile, ios::in);
  stdfile.seekg (atoi(id[pos].addr), ios::beg);
  unpack();
  if(usn[0] == '$')
  { cout << "This record has been deleted !! \n";
    return;
  }
  cout << usn << name << age << branch << sem << endl;
}
void Student :: remove (int pos)
{ opener (stdfile, datafile, ios::in | ios::out);
  stdfile.seekg (atoi (id[pos].addr), ios::beg);
  stdfile.put ('$');
  for (i=pos; i<indsize; i++)
  { id[i] = id[i+1]; indsize--;
}  }
Void Student :: unpack ()
{ stdfile.getline (usn, 15, '|');
  stdfile.getline (name, 20, '|');
  stdfile.getline (age, 5, '|');
```

```
opener (stdfile, datafile, ios::app);
cout << " Name:"
cin >> name;
cout << " age";
cin>> age;
cout<<" branch ",
 cin >> branch;
cout << " sem";
  cin >> sem;
pack();
stdfile.seekg(0, ios::end);
k = stdfile.tellg();
stdfile << buffer << endl;
strcpy_s( id[i].usn, dusn);
_itoa_s( k, id[i].addr, 10);
indsize++;

}
void student::pack()
{     strcpy_s(buffer, dusn);
      strcat_s(buffer, "|");
      strcat_s(buffer, name);
      strcat_s(buffer, "|");
      strcat_s(buffer, age);
      strcat_s(buffer, "|");
      strcat_s(buffer, sem);
      strcat_s(buffer, "|");
      strcat_s(buffer)

}
int search(char *fusn)
{ int low=0, high = indsize-1;
  int mid;
  while(low <= high)
  {   mid = (low + high)/2;
```

```
for (indsize = 0; ; indsize++)
{   indfile.getline(id[indsize].usn, 15, '|');
    indfile.getline(id[indsize].addr, 5, '\n');
    if (indfile.eof())
        break;
} indfile.close();
}

void opener(fstream &sfile, char *fn, int mode)
{ sfile.open(fn, mode);
  if (!sfile)
  { cout << "Unable to open file \n";
    exit(1);
  }
}
void index::write()
{ofstream indexfile, pos::out)
   indfile,
  for (i=0; i<indsize; i++)
  {   indfile << id[i].usn << "|" << id[i].addr << "|\n";
      indfile.close();
  }
}
void Student::add()
{ int k,
  cout << "Enter usn no \n";
  cin >>(dusn);
  if (search(dusn) >= 0)
  { cout << "usn present already";
    return;
  }
  for (i = indsize; i>0; i--)
  { if (strcmp(dusn, id[i-1].usn) <0)
        id[i] = id[i-1];
    else
        break;
```

Class 5 LP-5

Write a program to implement simple index on primary key for a file of student objects. Implement add(), search(), delete() using index

```
#include <iostream>
#include <fstream>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#define max 10
#define datafile "Students.txt"
#define indexfile "index5.txt"
using namespace std;
fstream stdfile, indfile;
int i, indsize;
char buffer[80];

class Student
{ char dum[15], age[5], sem[5], branch[5],
  name[20];
  public: void add();
          void pack();
          friend int search(char *);
          void recDisp(int);
          void remove(int);
          void datadisp();
          void unpack();
};
class index
{ public: char inum[5], addr[5];
          void initial(); void write();
} in, id[max];
void index::initial()
{ indfile.open(indexfile, ios::in);
  if(!indfile)
  { indsize=0; return; }
```

Note that

Label ID ~~4005 3745~~ is the first entry in the COL 38358

Label ID COL 38358 is the second entry in index not necessarily in data file which is entry sequenced which means they occur in the order they entered into the file.

Using index to provide access to data file by label ID is a simple matter.

```
int RetrieveRecording ( Recording & recording, char *key,
    TextIndex & RecordIndex, BufferFile & RecordingFile)
{  int result;
   result = RecordingFile.Read ( RecordingIndex.
      Search (key));
   if ( result == -1) return false;
   result = recording.Unpack (RecordingFile.Get
      Buffer());
   return result;
}
```

Advantage:

Keeping index in memory as pgm runs lets us find records by key more quickly with in index file than with a sorted one since the binary searching can be performed entirely in memory. Once byte offset for the data record is found, single seek is all that's required to retrieve the record.

② What are the limitations of key sort & how to overcome that?

Ans Limitations.

i) In key sort we read in the record a second time before we can write out new sorted file

a) Seeking

3) Based on sorting the RRN gets disturbed then we need to rearrange sequences.

To overcome this:
Problem with key sorting is writing records back in sorted order. Instead we can put an index to the original file.

---

① Explain the concept of simple indexing on entry sequenced file. What are the advantages of it?

Ans Index

Key	Reference field
AN63795	152
COL31809	338
COL38358	196

Address of record	Actual data record
196	COL38358 | Nebraska | Springston
152	AN63795 | Symphony 9 | Beet
338	COL | 31809 | Symphony No 9 | Dvorak

In Index, each entry of which contains key corresponding to a certain LabelID in the data file each key is associated with reference field. which is address of first byte of data record respectively.

The structure of the index object is very simple It is a list of pairs of field; a key field & a byte offset field. There is one entry in the index for each record in data file.