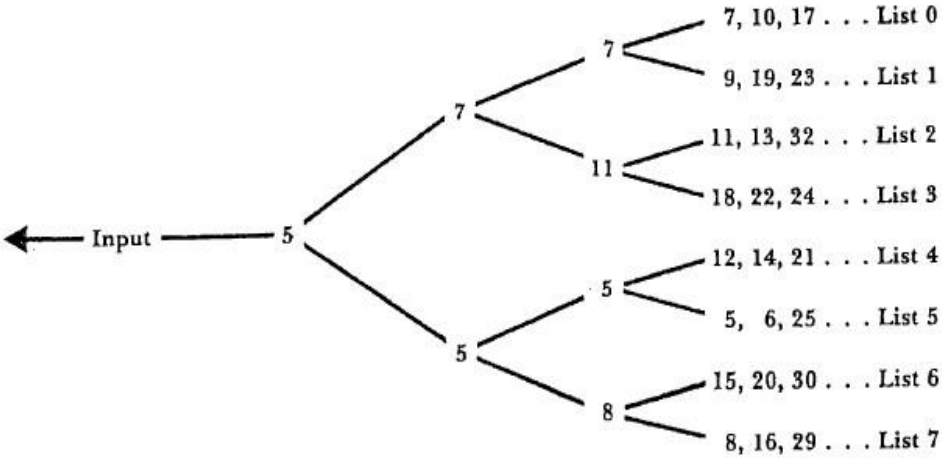


Internal Assessment Test 2 – May 2023
QP SCHEME

Sub:	File Structures					Sub Code:	18IS61	Branch:	ISE	
Date:	23/05/2023	Duration:	90 min's	Max Marks:	50	Sem/Sec:	VI A, B & C	OBE		
Answer any FIVE FULL Questions								MARKS	CO	RBT
1.	Explain all the operations required to maintain an indexed file with class declarations. <ul style="list-style-type: none"> ✓ Operations Required to Maintain an Indexed File ✓ Create the original empty index and data files, ✓ Load the index file into memory before using it, ✓ Rewrite the index file from memory after using it, ✓ Add data records to the data file, ✓ Delete records from the data file, ✓ Update records in the data file, ✓ Update the index to reflect changes in the data file. <pre> template <class RecType> class TextIndexedFile { public: int Read (RecType & record); // read next record int Read (char * key, RecType & record); // read by key int Append (const RecType & record); int Update (char * oldKey, const RecType & record); int Create (char * name, int mode=ios::in ios::out); int Open (char * name, int mode=ios::in ios::out); int Close (); TextIndexedFile (IOBuffer & buffer, int keySize, int maxKeys = 100); // access protected members ~TextIndexedFile (); // close and delete protected: TextIndex Index; // name correlation b/w index and data file BufferFile IndexFile; TextIndexBuffer IndexBuffer; RecordFile<RecType> DataFile; char * FileName; // base file name for file int SetFileName(char * fileName, char *& dataFileName, char *& indexFileName); }; // The template parameter RecType must have the following method // char * Key() </pre> <p>Figure 7.7 Class TextIndexedFile</p>					10	CO2	L2		
2.a	Explain the importance of secondary keys for searching and illustrate with suitable example <ul style="list-style-type: none"> ✓ Two or more Secondary key can be used to retrieve special subsets of records from the data file. ✓ An example is illustrated in previous figure & one can respond to requests such as: ✓ Find the recording with Label ID COL37483 (Primary Key access); ✓ Find all the recordings of Bethoven's work (Secondary Key composer) and ✓ Find all the Recordings title "Violin Concerto"(Secondary key N title) ✓ Requests can be rephrased as a Boolean "and" operation. 					5	CO2	L3		

	<p>We begin by recognizing that this request can be rephrased as a Boolean <i>and</i> operation, specifying the intersection of two subsets of the data file:</p> <p>Find all data records with: composer = 'BEETHOVEN' and title = 'SYMPHONY NO. 9'</p> <p>We begin our response to this request by searching the composer index for the list of Label IDs that identify recordings with Beethoven as the composer. This yields the following list of Label IDs:</p> <p>ANG3795 DG139201 DG18807 RCA2626</p> <p>Next we search the title index for the Label IDs associated with records that have SYMPHONY NO. 9 as the title key:</p> <p>ANG3795 COL31809 DG18807</p> <p>Now we perform the Boolean <i>and</i>, which is a match operation, combining the lists so only the members that appear in <i>both</i> lists are placed in the output list.</p> <table border="0"> <thead> <tr> <th>Composers</th> <th>Titles</th> <th>Matched list</th> </tr> </thead> <tbody> <tr> <td>ANG3795</td> <td>ANG3795</td> <td>ANG3795</td> </tr> <tr> <td>DG139201</td> <td>COL31809</td> <td></td> </tr> <tr> <td>DG18807</td> <td>DG18807</td> <td>DG18807</td> </tr> <tr> <td>RCA2626</td> <td></td> <td></td> </tr> </tbody> </table> <ul style="list-style-type: none"> ✓ It is much easier to match in sorted list. ✓ Finally look up the addresses of the data file records. ✓ Then records can be retrieved as: <pre> ANG 3795 Symphony No. 9 Beethoven Guilini DG 18807 Symphony No. 9 Beethoven Karajan </pre>	Composers	Titles	Matched list	ANG3795	ANG3795	ANG3795	DG139201	COL31809		DG18807	DG18807	DG18807	RCA2626					
Composers	Titles	Matched list																	
ANG3795	ANG3795	ANG3795																	
DG139201	COL31809																		
DG18807	DG18807	DG18807																	
RCA2626																			
2. b	<p>Write a C++ code to implement Secondary key.</p> <pre> class SecondaryIndex // An index in which the record reference is a string public: int Insert (char * secondaryKey, char * primaryKey); char * Search (char * secondaryKey); // return primary key ... // searches a secondary key index }; // Then uses primary key to read an Indexed File template <class RecType> int SearchOnSecondary (char * composer, SecondaryIndex index, IndexedFile<RecType> dataFile, RecType & rec) { char * Key = index.Search (composer); // use primary key index to read file return dataFile . Read (Key, rec); } </pre> <p>Figure 7.9 SearchOnSecondary: an algorithm to retrieve a single record from a recording file through a secondary key index.</p>	5	CO2	L3															
3.a	<p>Explain Class Heap and insert () a Heap function</p> <ul style="list-style-type: none"> ✓ The algorithm for heap sort has 2 parts. ✓ First we build the heap; and ✓ Then we output the keys in sorted order. <pre> class Heap{ public: Heap(int maxElements); int Insert (char* newKey); char* Remove(); protected: int MaxElements; int NumElernents; char** HeapArray; void Exchange(int i, int j); // exchange element i and j int Compare (inti, int j) // compare element i and j { return strcmp(HeapArray[i],HeapArray[j]) } // returns -1, if left element is smaller </pre>	5	CO2	L2															

	<pre> }; ✓ Insert method that adds a string to the heap is shown. int Heap::Insert(char * newKey) { if (NumElements == MaxElements) return FALSE; NumElements++; // add the new key at the last position HeapArray[NumElements] = newKey; // re-order the heap int k = NumElements; int parent; while (k > 1) // k has a parent { parent = k / 2; if (Compare(k, parent) >= 0) break; // HeapArray[k] is in the right place // else exchange k and parent Exchange(k, parent); k = parent; } return TRUE; } </pre>			
3.b	<p>What is tree? Explain how the Binary search tree used to reduces search time?</p> <ul style="list-style-type: none"> ✓ B-trees are balanced search tree. ✓ More than 2 children are possible. ✓ B-Tree, stores all information in the leaves and stores only keys and Child pointer. ✓ If an internal B-tree node x contains n[x] keys then x has n[x]+1 children. <p>Statement of the problem</p> <ul style="list-style-type: none"> ✓ Searching an index must be faster than binary searching. ✓ Inserting and deleting must be as fast as searching. <p>Searching:</p> <ul style="list-style-type: none"> ➤ The searching procedure is iterative, loading a page into memory and then searching through the page, looking for the key at successively lower levels of the tree until it reaches the leaf level. ➤ They are iterative and they work in two stages, operating alternatively on entire pages and then within pages. <pre> Template <class keyType> Int Btree<keyType>:: Search (const keyType key, const int recAddr) { BTreeNode<keyType> * leafNode; leafNode = FindLeaf(key); return leafNode->Search (key, recAddr); } Template <class keyType> BTreeNode<keyType> * Btree <keyType>::FindLeaf(const keyTuype key) { int recAddr, level; for (level = 1; level < Height; level++) { recAddr=Nodes[level-1]-Search(key, -1,0); Nodes[level]=Fetch(recAddr); } Return Nodes[level-1]; } </pre>	5	CO2	L2

<p>4.</p>	<p>Apply the concept of k-way merge and explain selection tree with example.</p> <ul style="list-style-type: none"> ✓ The K-way merge works nicely if K is no larger than 8 or so. ✓ Merging a larger number of lists, & finding the key becomes expensive. ✓ For practical reasons the use of sequential comparison is a good strategy. ✓ If there is a need to merge more than 8 lists replace the loop of comparisons with a selection tree (tournament tree). <div style="text-align: center;">  </div> <p>Figure 8.15 Use of a selection tree to assist in the selection of a key with minimum value in a <i>K</i>-way merge.</p>	<p>10</p>	<p>CO2</p>	<p>L3</p>
<p>5.a</p>	<p>Analyze the requirement of co-sequential model for implement General Ledger Program.</p> <ul style="list-style-type: none"> ✓ Class CosequentialProcessing supports method Match ✓ Class StringListProcess defines the supporting operations for lists. ✓ To use class CosequentialProcessing create a sub class StringListProcess ✓ Main members and methods of a general class for cosequential processing ✓ There are 3 different steps in processing the ledger entries: <ul style="list-style-type: none"> 1. Immediately after reading a new ledger object: <ul style="list-style-type: none"> ✓ Print the header line and initialize the balance for next month from previous month's balance. 2. For each transaction object that matches, update the account balance. 3. After the last transaction for the account, the balance line should be printed. 	<p>5</p>	<p>CO2</p>	<p>L2</p>

```

template <class ItemType>
class MasterTransactionProcess:
public CosequentialProcess<ItemType>
// a cosequential process that supports
// master/transaction processing
{public:
MasterTransactionProcess (); //constructor
virtual int ProcessNewMaster ()=0;
// processing when new master read
virtual int ProcessCurrentMaster ()=0;
// processing for each transaction for a master
virtual int ProcessEndMaster ()=0;
// processing after all transactions for a master
virtual int ProcessTransactionError ()=0;
// no master for transaction

// cosequential processing of master and transaction records
int PostTransactions (char * MasterFileName,
char * TransactionFileName, char * OutputListName);
};

```

Figure 8.12 Class MasterTransactionProcess.

```

while (MoreMasters || MoreTransactions) //Ledges || Journal
if (Item(1) < Item(2)) { // finish this master record
ProcessEndMaster(); //no more transaction in this month, to
MoreMasters = NextItemInList(1); // add to ledger, so
if (MoreMasters) ProcessNewMaster(); // print ledger //
} // account ends // print title line
else if (Item(1) == Item(2)) { // transaction matches master
add transaction // print transaction description
// to account balance // read next journal entry.
ProcessCurrentMaster(); // another transaction for master
ProcessItem (2); // output transaction record
MoreTransactions = NextItemInList(2); // read next journal entry.
}
else { // Item(1) > Item(2) transaction with no master
ProcessTransactionError(); // unmatched journal account.
MoreTransactions = NextItemInList(2);
}
}

```

Figure 8.13 Three-way-test loop for method PostTransactions of class MasterTransactionProcess.

```

int LedgerProcess::ProcessNewMaster ()
{ // print the header and setup last month's balance
ledger.PrintHeader(OutputList);
ledger.Balances[MonthNumber] = ledger.Balances[MonthNumber-1];
}

int LedgerProcess::ProcessCurrentMaster ()
{ // add the transaction amount to the balance for this month
ledger.Balances[MonthNumber] += journal.Amount;
}

int LedgerProcess::ProcessEndMaster ()
{ // print the balances line to output
PrintBalances(OutputList,
ledger.Balances[MonthNumber-1], ledger.Balances[MonthNumber]);
}

```

Figure 8.14 Master record processing for ledger objects.

5.b	What is Runs? Discuss the role runs in sorting large files. ✓ For large files whole file nor keys can be sorted in memory.	5	CO2	L2
-----	---	---	-----	----

- ✓ The multiway merge algorithm provides solution to problem of sorting large files.
- ✓ Create a sorted sub set of full file. This subfile is called **run**.
- ✓ A multiway merge can be used to create a completely sorted file containing all the original records.
- ✓ A schematic VIEW of this run creation and merging process is provided in Fig next.

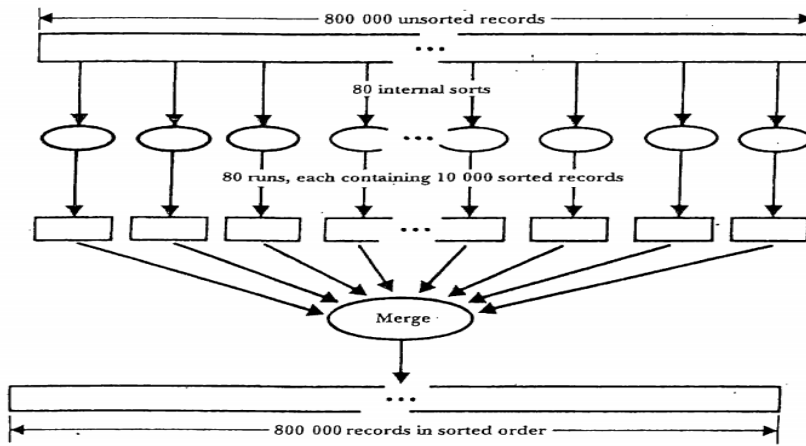


Figure 8.21 Sorting through the creation of runs (sorted subfiles) and subsequent merging of runs.

6. Discuss about the inverted list to improve secondary index structure.

10

CO2

L2

A First Attempt at a solution

- ✓ Change the secondary index structure so it associates an array of references with each secondary key.

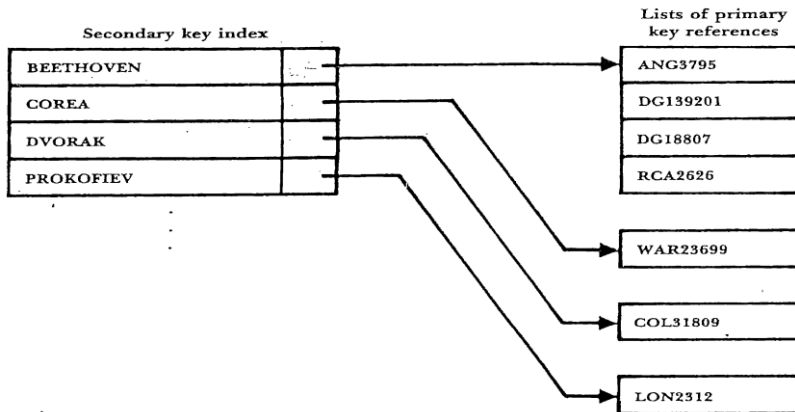
BEETHOVEN	ANG3795	DG139201	DG18807	RCA2626
-----------	---------	----------	---------	---------

- ✓ This helps in solving the problem of rearrange the secondary index file every time a new record is added.

Secondary key	Revised composer index Set of primary key references			
BEETHOVEN	ANG3795	DG139201	DG18807	RCA2626
COREA	WAR23699			
DVORAK	COL31809			
PROKOFIEV	LON2312			
RIMSKY-KORSAKOV	MER75016			
SPRINGSTEEN	COL38358			
SWEET HONEY IN THE R	FF245			

A Better Solution: Linking the List of References

- ✓ Files in which a secondary key leads to a set of one or more primary keys, are called inverted lists.
- ✓ Here list refers to list of primary key references.



- ✓ Improved revision of the composer index with secondary index having 2 fields
- ✓ This new design result in a secondary key file for composers & an associated Label ID file that are organized.

Secondary Index file

0	BEETHOVEN	3
1	COREA	2
2	DVORAK	7
3	PROKOFIEV	10
4	RIMSKY-KORSAKOV	6
5	SPRINGSTEEN	4
6	SWEET HONEY IN THE R	9

Label ID List file

0	LON2312	- 1
1	RCA2626	- 1
2	WAR23699	- 1
3	ANG3795	8
4	COL38358	- 1
5	DG18807	1
6	MER75016	- 1
7	COL31809	- 1
8	DG139201	5
9	FF245	- 1
10	ANG36193	0