**Karanja Rachel Wangechi**

**ICS3B – 113851**

**Revision Questions Assignment**

**File Storage and Organization**

***Section A***

1. Primary storage refers to the main storage of the computer which is the main memory particularly the Random Access Memory(RAM) while secondary storage refers to storage that is done outside the computer and is done in external devices and stores data for long term especially in USB drives.
2. This is because store data for a long time and thus prevents data loss as compared to RAM which is volatile and prone to data loss.
3. -Disk – it is a random access addressable device.

-Disk pack – this refers to several magnetic surfaces connected to a rotating spindle.

- track – refers to a circle within a disk.

- Block – refers to a portion of a track or an arc of a track.

- Cylinder – this is a combination of many tracks that have the same diameter on various surfaces.

- Sector – it is the same as a block.

- Interblocking gap – this is the gap that exists between the blocks in the track which contain coded control information written during disk initialization.

- Read/write head – this is a component that reads and writes information onto the correct track in a disk.

1. In the case of reading data into a disk, the read-write head has to be placed in the correct track so that reading data into the block can be easier and more efficient.
2. Components of the disk block address include cylinder number(collection of same tracks with the same diameter on various surfaces), track number(within the cylinder) and block number(within the track)
3. Disk block access is expensive because it is time consuming. For disk access, the total time it takes to access a disk block involves the seek time which refers to the time it takes to place the read-write head onto the correct path, the rotational delay which refers to the average time it takes to locate the correct block and also the block transfer rate which refers to the time taken to transfer data.
4. It improves block access time whereby data in one buffer is being processed while the next set of data is read into the other buffer.
5. Using variable-fixed length records will minimize on wastage of space because records of different lengths can be stored therefore unspanned blocking can be used. The separator characters that are used are ? or %.
6. Contiguous allocation – each file occupies a set of consecutive addresses on a disk. It normally does not leave additional space therefore it does not support overflows.

* Linked allocation – each data block contains the block address of the next block in the file. Each directory entry contains file name, block address which contains a pointer to the first block address and also has a pointer to the last block.
* Indexed allocation – store all pointers together in an index table.

1. **Advantages of unordered files**

* Record insertion is quite efficient.
* New records are usually inserted at the end of the file.
* Primary keys are used to identify unique value in the records.

**Disadvantages of unordered files.**

* Reading records in order of a particular field requires sorting the file record.
* It would take b/2 average time to search for particular records in the field which is quite inefficient.

**Advantages of ordered files.**

* Reading records in order of the ordering field is quite efficient.
* Binary search can be used to search for a record on its ordering field value.

**Disadvantages of ordered files.**

* Insertion is expensive because records must be inserted in the correct order.
* Insertion is good in unordered files because records and fields are stored in heaps and insertion is also efficient in ordered files only if they are in order. For unordered files, field and record search is expensive while in ordered files, the search time is less and more efficient.

***Section B***

1. Record size = (9+20+20+1+10+35+12+9+4+4) + 1 = 125bytes

Bfr = (floor)B/R = 2400/125 = 19 records per block

B = r/bfr = 30000/19 = 1579 blocks

1. Wasted space = B – (R\*bfr) = 2400 – (125\*19.2) = 25 bytes
2. Transfer rate(tr) = B/btt = 2400/1 = 2400bytes per millisecond

Bulk transfer rate = B/btr = tr \* (B/(B+G))

= 2400 \*(2400/(2400+600)

= 1920bytes/msec

**File Structures, Indexing and Hashing**

***Section A***

1. – indexing field – a field that is used to create an index value for the records in the main file.

* Primary key field – a record in the file that is unique and is used to identify all the other records in the file.
* Clustering field – this is a non-key value that is used to identify records in the same cluster and requires the ordering field of the data file to have a unique value.
* Secondary key index – it is a key that may be on a field that is a candidate key and has a unique value in every record.
* Block anchor – the first block on an indexing file.
* Dense index – an index that requires one to create an index key value for each and every record in a file.
* Sparse index- an index record that appears for only some values of the values in the index file.

1. For the case of the primary index, it uses the sparse indexing since one has to create an index key for each and every record in the main file and this will require so much space. Insertion and deletion of records is very inefficient because other records might be forced to move since the record files need to be ordered. As for the secondary index, it is usually based on a candidate key that is uniquely identifies records on a file. The index is usually an ordered file with two fields which makes insertion and deletion quite expensive but searching for a record is quite efficient. This index uses the dense index. Lastly, the clustering index is usually based on a non-key value to index files and records and is unordered therefore insertion and deletion is quite efficient. It uses the sparse indexing since every index records reference to one cluster/one block of information or data.
2. A primary index should be unique to records in the file and searching for records will be easier because the primary index is usually a primary key as compared to the secondary index which is a candidate key that is unique and provides a secondary means of accessing a file that already has a primary index.
3. It helps in creation of an index for another index therefore the top-level index will fit into one block and since the first level index is ordered and the key on the index value, the other values can be sparse.
4. Order p of a B-tree simply means that each node has a maximum of p children. Each node contains (p-1) keys, keys are usually arranged in ascending order and when a node is full, the median value of the values is pushed up and the node is split.
5. Order p of B+ tress means that each node has a maximum of p children only that entries in a leaf contain pointers to the data records. The non-leaf nodes form a sparse index on the leaf level and key values may appear more than once in the tree. In B+ trees, only leaf nodes store record pointers, all search values are stored in the leaf node and leaf nodes are usually linked into a sorted list.
6. B+ tree is more efficient because it contains pointers to the values and is more dynamic and can adapt to changes.

***Section B***

1. P = 4; p-1 = keys i.e. 4-1 = 3 keys

**Steps**

1. First, you insert 20,30 and 67

|  |  |  |
| --- | --- | --- |
| **20** | **30** | **67** |

1. **To insert 60, find the median of 20,30,60,70 and then push it up and the node splits.**

|  |
| --- |
| **60** |

|  |  |
| --- | --- |
| **20** | **30** |

|  |  |
| --- | --- |
| **60** | **67** |

1. To insert 46, add it in between 20 and 30.

|  |
| --- |
| **60** |

|  |  |
| --- | --- |
| **60** | **67** |

|  |  |  |
| --- | --- | --- |
| **20** | **30** | **46** |

1. To insert 98, insert it after 67.

|  |
| --- |
| **60** |

|  |  |  |
| --- | --- | --- |
| **60** | **67** | **98** |

|  |  |  |
| --- | --- | --- |
| **20** | **30** | **46** |

1. To insert 46, the median of 20,30,46,46 is got which is 46, it is pushed up and the node splits and the value is retained in the child node

|  |  |
| --- | --- |
| **46** | **60** |

|  |  |  |
| --- | --- | --- |
| **60** | **67** | **98** |

|  |  |  |
| --- | --- | --- |
| **20** | **30** | **46** |

|  |  |
| --- | --- |
| 46 | 46 |

1. To insert 74, find the median of 60,67,74 and 98 which is 74. This value is pushed up and the node splits again.

|  |  |  |
| --- | --- | --- |
| **46** | **60** | **74** |

|  |  |
| --- | --- |
| **60** | **67** |

|  |  |  |
| --- | --- | --- |
| **20** | **30** | **46** |

|  |  |
| --- | --- |
| 46 | 46 |

|  |  |
| --- | --- |
| 74 | 98 |

1. To insert 53, insert it after 46

|  |  |  |
| --- | --- | --- |
| **46** | **60** | **74** |

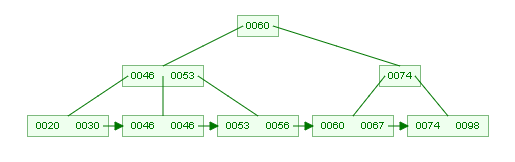
|  |  |
| --- | --- |
| **60** | **67** |

|  |  |  |
| --- | --- | --- |
| **20** | **30** | **46** |

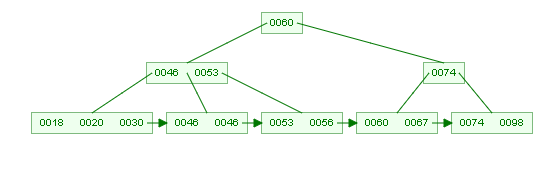
|  |  |
| --- | --- |
| 74 | 98 |

|  |  |  |
| --- | --- | --- |
| 46 | 46 | 53 |

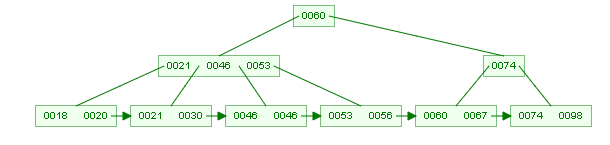
1. To insert 56, find the median of 46,46,53,56 which is 53 so it is pushed up and the node is split but the node is split and the median of 46,56,60,74 is 60 and this time 74 is pushed up and the node is split again.



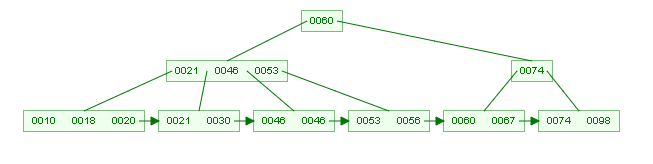
1. To insert 18, place it after before the 20.



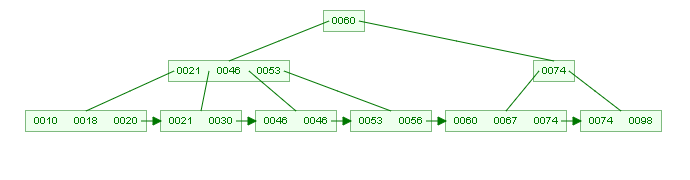
1. To insert 21, find the median of 18,20,21,30 which is 21, it is pushed upwards and the node splits.



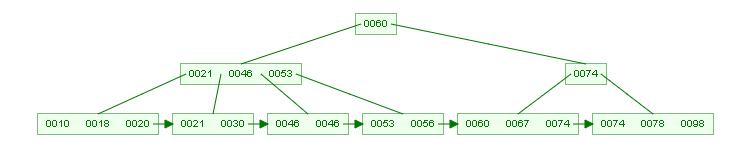
1. To insert 10, place it before 18 and 20



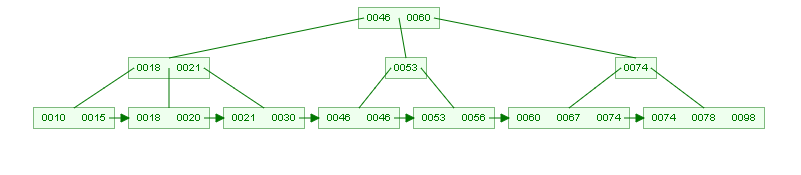
1. To insert 74, place it before 60 and 67.



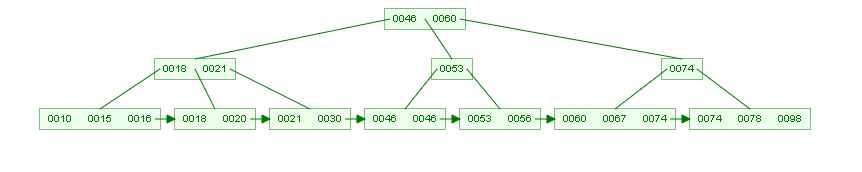
1. To insert 78, place it between 74 and 98.



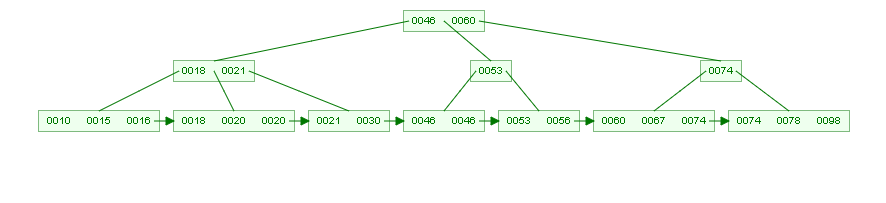
1. To insert 15, get the median of 10,15,18,20 and push 18 up and then the node splits. Since the parent node is full, get the median again of 18,21,48,53 which is 48 then push it up and copy the value in the child node.



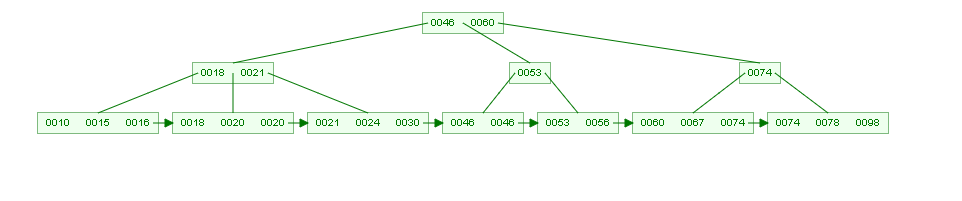
1. To insert 16, place it after 10 and 15.



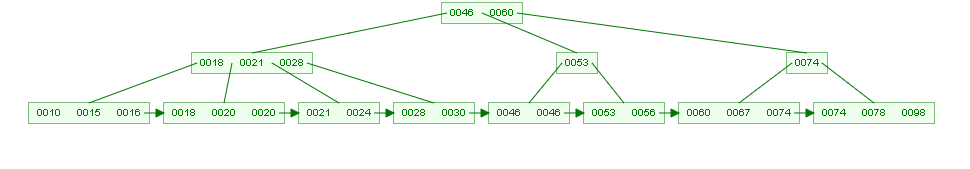
1. To insert 20, put it after 18 and 20.



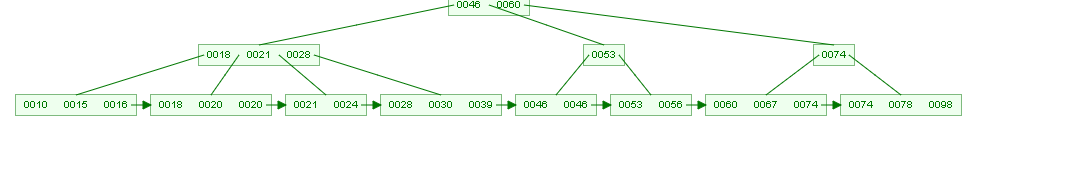
1. To insert 24, place it between 21 and 30.



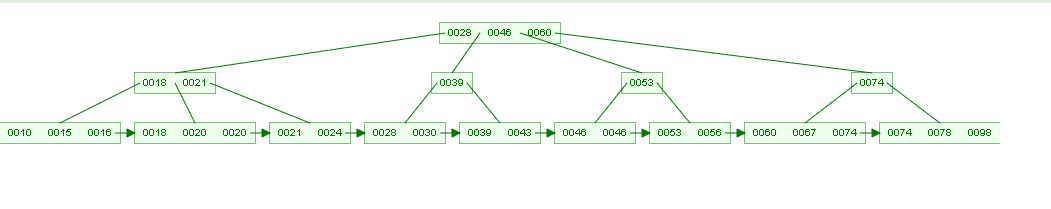
1. To insert 28, find the median of 21,24,28,30 which is 28 thus it is pushed up and the node splits and the value is copied onto the child node.



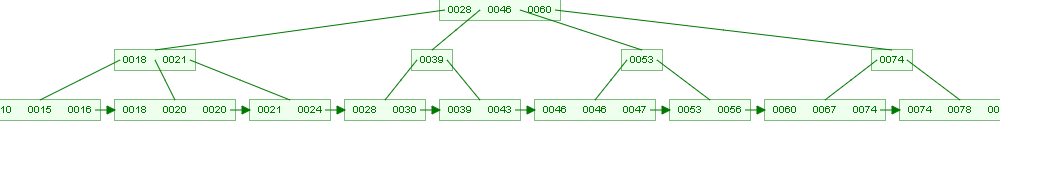
1. To insert 39, place it after 28 and 30.



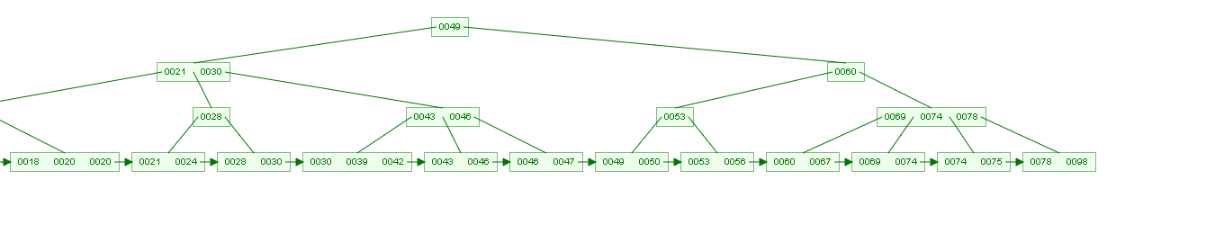
1. To insert 43, get the median of 28,30,39,43 which is 39 so it is pushed up and the value is copied into the child node. Since the parent node is full, the median is got again i.e. 18,21,28,39 which is 28 and 28 is hence pushed up and its value is copied in the child node and the node splits.



1. To insert 47, place it after 46 and 46.



1. To insert 50, find the median of 46,46,47,50 to get 47 which is then pushed up, the value is copied into the child node and the node splits. The final B+ tree will be:



1. In the B-tree, values are not copied into their child nodes. The final B-tree will be:

