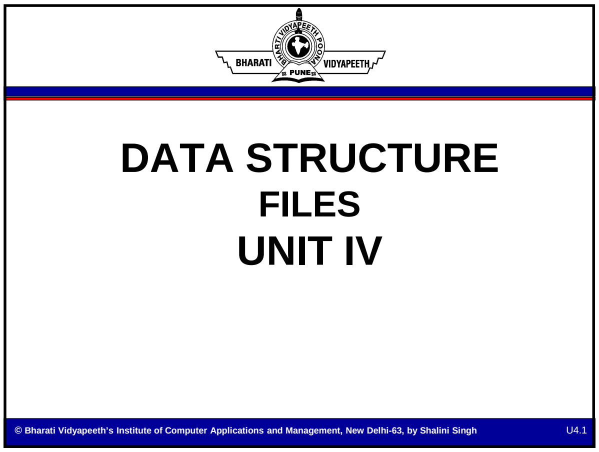
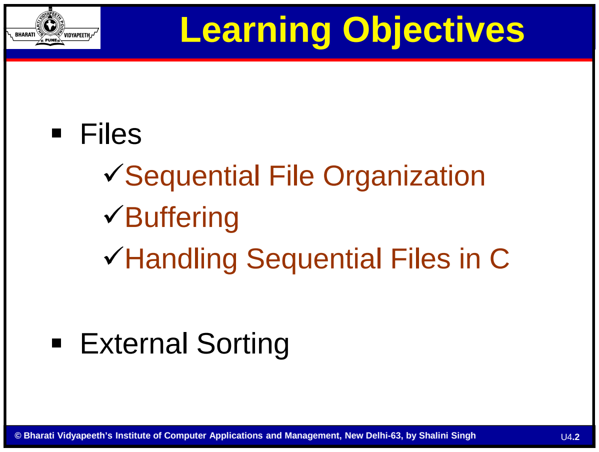


DATA STRUCTURE
FILES
UNIT IV

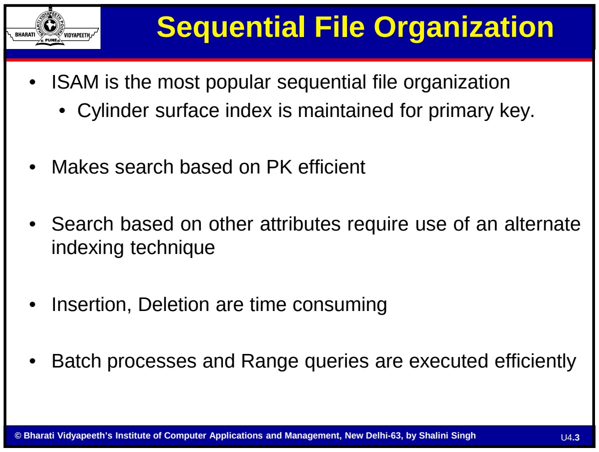
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Shalini Singh U4.1



Learning Objectives

- Files
  - ✓Sequential File Organization
  - ✓Buffering
  - ✓Handling Sequential Files in C
- External Sorting

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Shalini Singh U4.2



Sequential File Organization

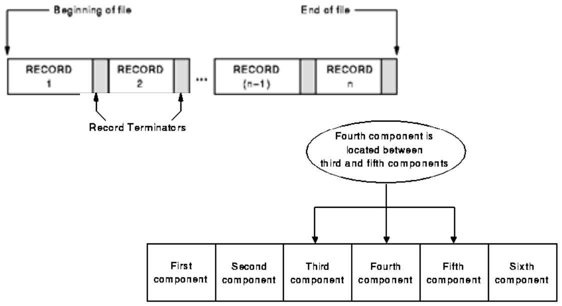
- ISAM is the most popular sequential file organization
  - Cylinder surface index is maintained for primary key.
- Makes search based on PK efficient
- Search based on other attributes require use of an alternate indexing technique
- Insertion, Deletion are time consuming
- Batch processes and Range queries are executed efficiently

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Shalini Singh U4.3

Sequential Files

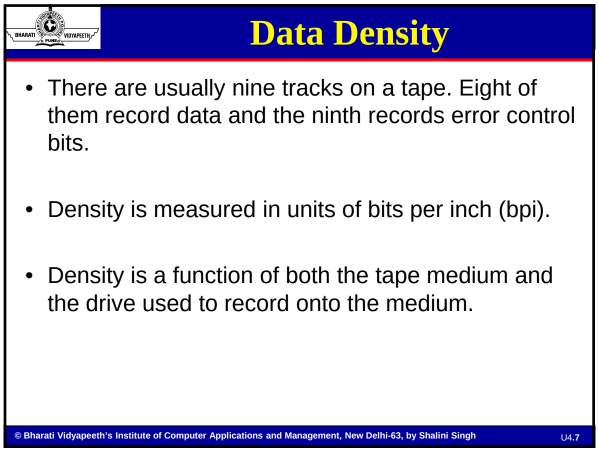
- Sequential files are files where the order of the records in the file is based on the order the records are placed in the file (that is, in arrival sequence)
- The order of the records is fixed.
- Records in these files can only be read or written sequentially i.e. to read Nth record we must first access N-1 records.

Sequential File Organization



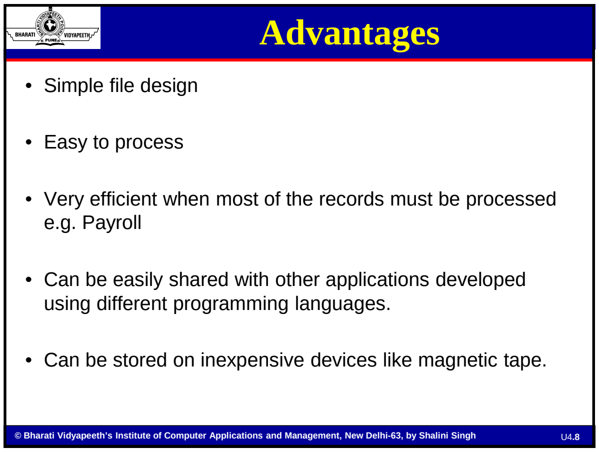
Sequential Files

- One common storage medium used for sequential files is Magnetic Tape.
- Here data is recorded digitally as magnetized spots in the film coating.
- Positive magnetization may represent 1-bit and negative magnetization may represent a 0-bit or vice-versa.
- The magnetized areas are not randomly located on the medium but are arranged in tracks parallel to the edge of the tape.



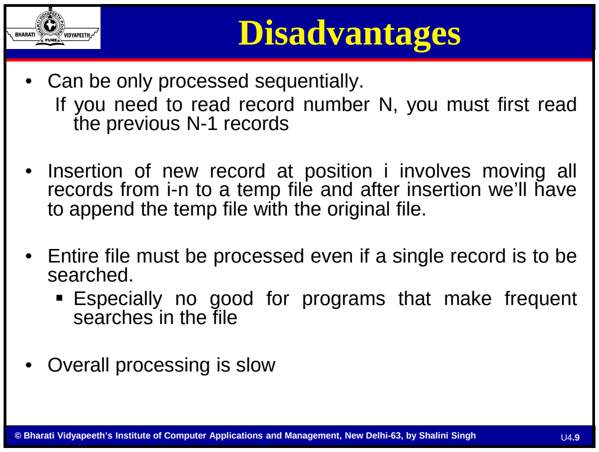
Data Density

- There are usually nine tracks on a tape. Eight of them record data and the ninth records error control bits.
- Density is measured in units of bits per inch (bpi).
- Density is a function of both the tape medium and the drive used to record onto the medium.



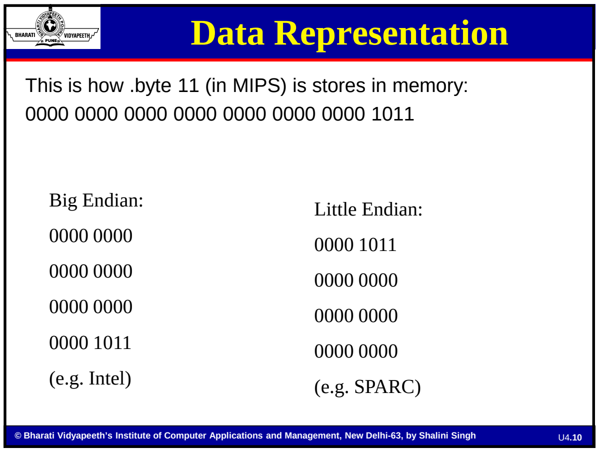
Advantages

- Simple file design
- Easy to process
- Very efficient when most of the records must be processed e.g. Payroll
- Can be easily shared with other applications developed using different programming languages.
- Can be stored on inexpensive devices like magnetic tape.



Disadvantages

- Can be only processed sequentially.
  If you need to read record number N, you must first read the previous N-1 records
- Insertion of new record at position i involves moving all records from i-n to a temp file and after insertion we'll have to append the temp file with the original file.
- Entire file must be processed even if a single record is to be searched.
  - Especially no good for programs that make frequent searches in the file
- Overall processing is slow

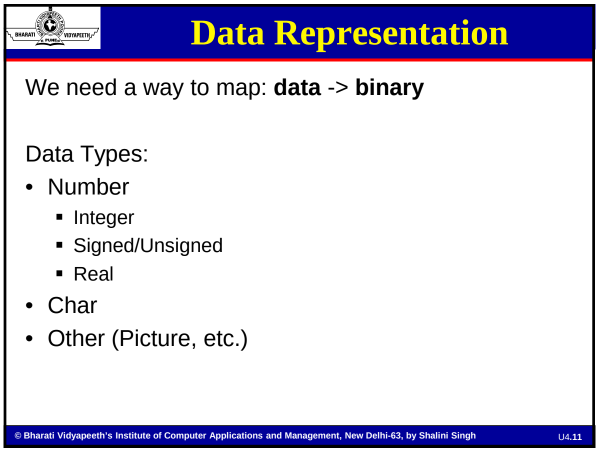


Data Representation

This is how .byte 11 (in MIPS) is stores in memory:
0000 0000 0000 0000 0000 0000 0000 1011

Big Endian:	Little Endian:
0000 0000	0000 1011
0000 0000	0000 0000
0000 0000	0000 0000
0000 1011	0000 0000
(e.g. Intel)	(e.g. SPARC)

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Shalini Singh U4.10



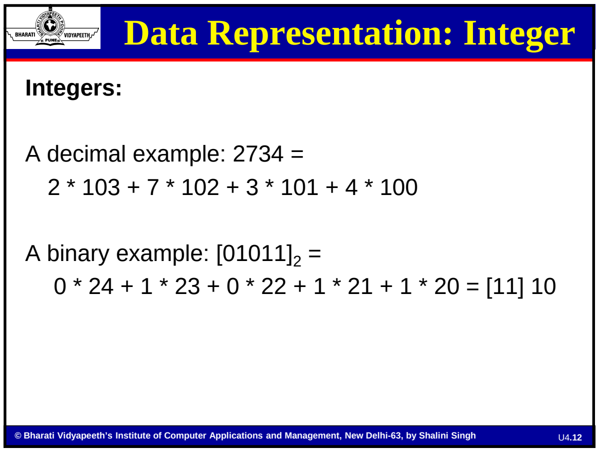
Data Representation

We need a way to map: **data** -> **binary**

Data Types:

- Number
  - Integer
  - Signed/Unsigned
  - Real
- Char
- Other (Picture, etc.)

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Shalini Singh U4.11



Data Representation: Integer

Integers:

A decimal example: 2734 =
2 * 103 + 7 * 102 + 3 * 101 + 4 * 100

A binary example: [01011]$_2$ =
0 * 24 + 1 * 23 + 0 * 22 + 1 * 21 + 1 * 20 = [11] 10

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Shalini Singh U4.12

Data Representation: Unsigned

Unsigned

Unsigned integers are positive (or to be more precise, do not have a sign).

Size	Limit (low, high)
8 bits	0, 255 (255 = 2^8 - 1)
16 bits	0, 64k (64k = 2^{16} - 1)
…	…
N bits	0, $2^N – 1$

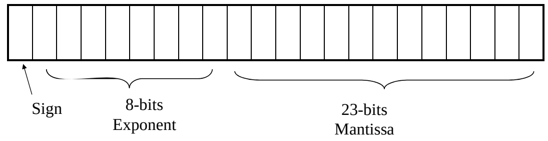
Signed

Represented in 2's complement form

Data Representation: Real Nos.

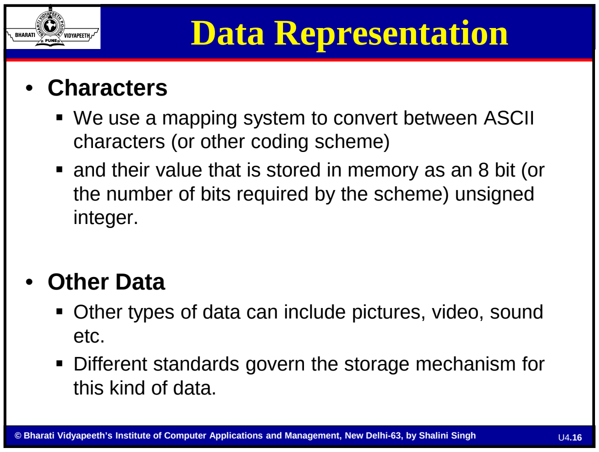
•**Real Numbers**

- There are many standards to represent real numbers. E.g. the IEEE 754 standard.



Data Representation: Real

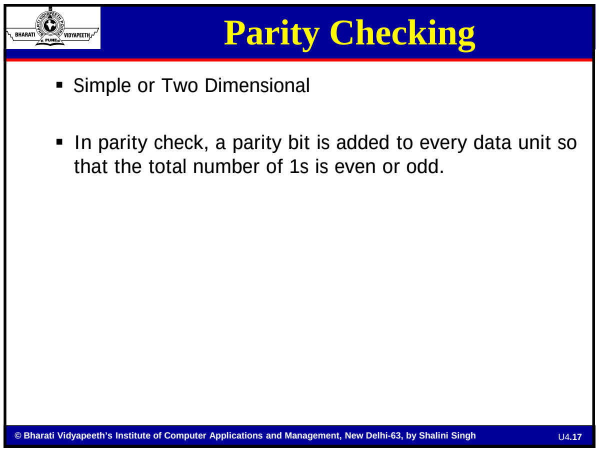
- The first bit is the sign,
- The next 8-bit portion is the Biased Exponent (the exponent of 2 plus 127 to ensure we get a positive number)
- The last 23-bit portion is Mantisa.
- *Note* that there are 2^{24} digits in the fraction part (including the 1 *normalized form* that is not stored in memory) which represents 6 digits.
  - This is the number of significant figures in a floating point number.
- Most CPUs have 6-7 significant figures



Data Representation

- **Characters**
  - We use a mapping system to convert between ASCII characters (or other coding scheme)
  - and their value that is stored in memory as an 8 bit (or the number of bits required by the scheme) unsigned integer.
- **Other Data**
  - Other types of data can include pictures, video, sound etc.
  - Different standards govern the storage mechanism for this kind of data.

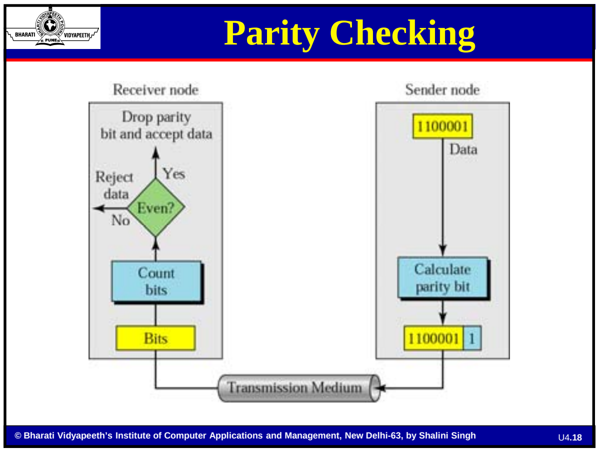
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Shalini Singh U4.16



Parity Checking

- Simple or Two Dimensional
- In parity check, a parity bit is added to every data unit so that the total number of 1s is even or odd.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Shalini Singh U4.17



Parity Checking

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Shalini Singh U4.18

Parity Checking

Suppose the sender wants to send the word *world*. In ASCII the five characters are coded as

1110111 1101111 1110010 1101100 1100100

The following shows the actual bits sent

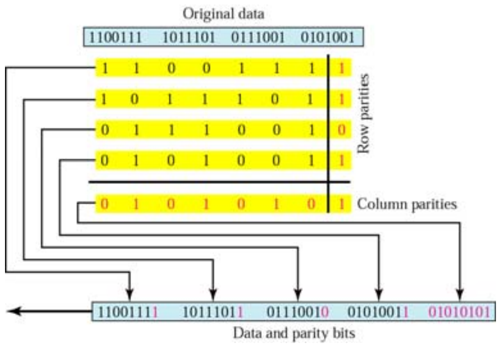
11101110 11011110 11100100 11011000 11001001

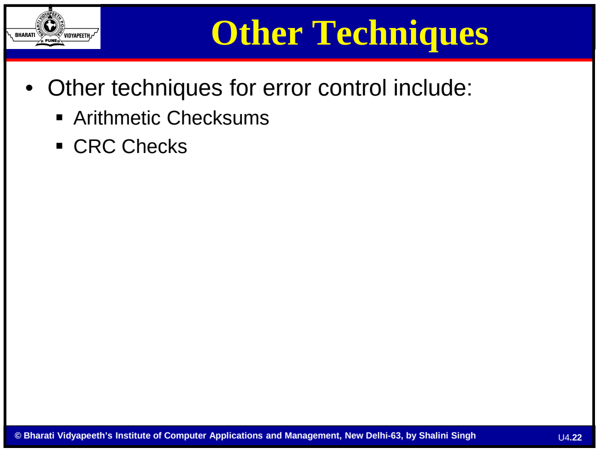
Parity Checking

Two-dimensional parity

- ✓ In two-dimensional parity check, a block of bits is divided into rows and a redundant row of bits is added to the whole block.

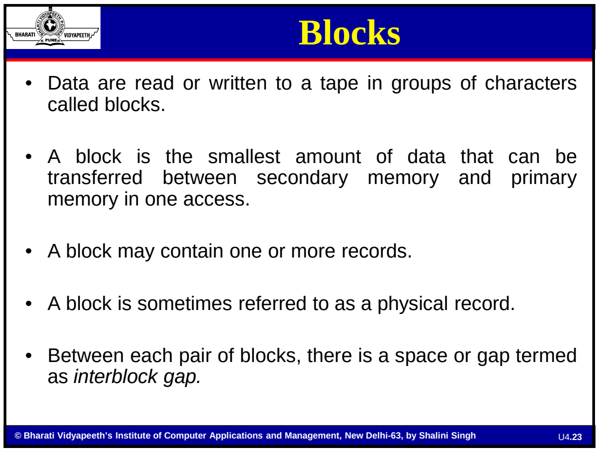
2D Parity Checking





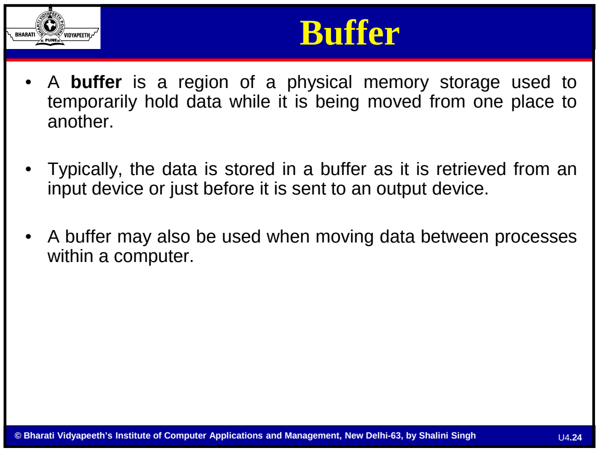
Other Techniques

- Other techniques for error control include:
  - Arithmetic Checksums
  - CRC Checks



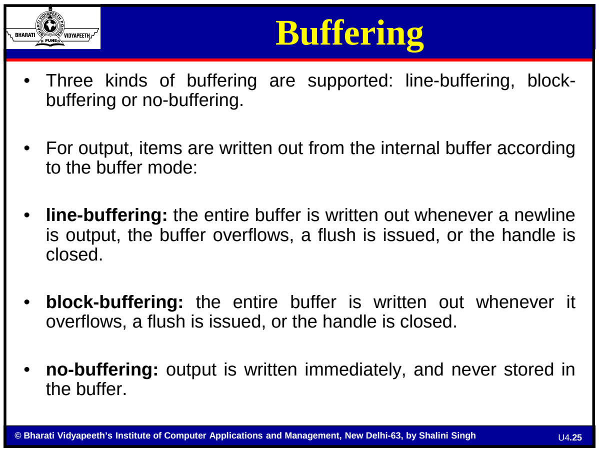
Blocks

- Data are read or written to a tape in groups of characters called blocks.
- A block is the smallest amount of data that can be transferred between secondary memory and primary memory in one access.
- A block may contain one or more records.
- A block is sometimes referred to as a physical record.
- Between each pair of blocks, there is a space or gap termed as *interblock gap.*



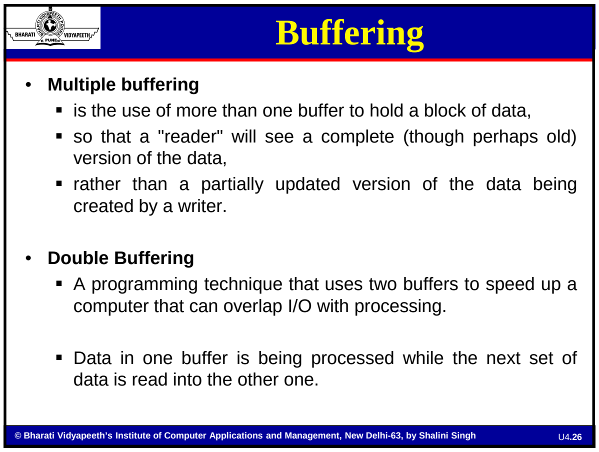
Buffer

- A **buffer** is a region of a physical memory storage used to temporarily hold data while it is being moved from one place to another.
- Typically, the data is stored in a buffer as it is retrieved from an input device or just before it is sent to an output device.
- A buffer may also be used when moving data between processes within a computer.



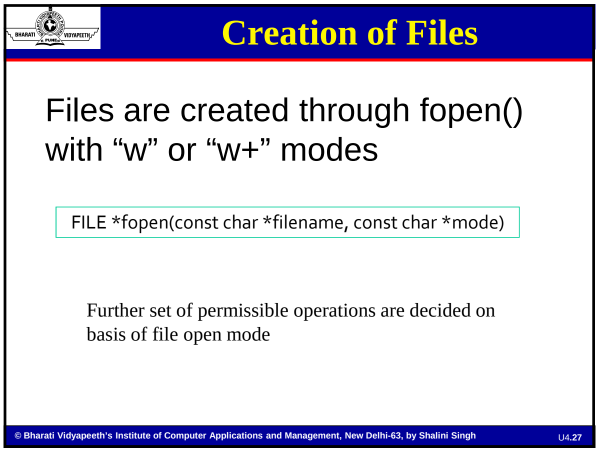
Buffering

- Three kinds of buffering are supported: line-buffering, block-buffering or no-buffering.
- For output, items are written out from the internal buffer according to the buffer mode:
- **line-buffering:** the entire buffer is written out whenever a newline is output, the buffer overflows, a flush is issued, or the handle is closed.
- **block-buffering:** the entire buffer is written out whenever it overflows, a flush is issued, or the handle is closed.
- **no-buffering:** output is written immediately, and never stored in the buffer.



Buffering

- **Multiple buffering**
  - is the use of more than one buffer to hold a block of data,
  - so that a "reader" will see a complete (though perhaps old) version of the data,
  - rather than a partially updated version of the data being created by a writer.
- **Double Buffering**
  - A programming technique that uses two buffers to speed up a computer that can overlap I/O with processing.
  - Data in one buffer is being processed while the next set of data is read into the other one.

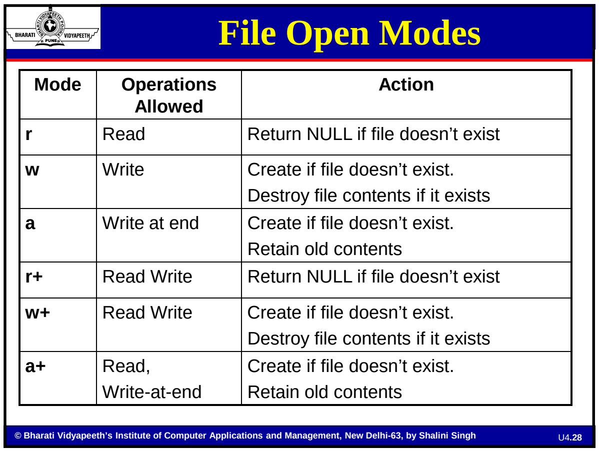


Creation of Files

Files are created through fopen() with “w” or “w+” modes

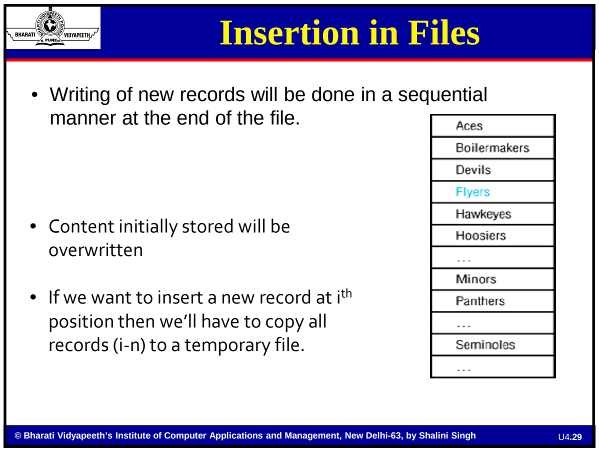
```
FILE *fopen(const char *filename, const char *mode)
```

Further set of permissible operations are decided on basis of file open mode



File Open Modes

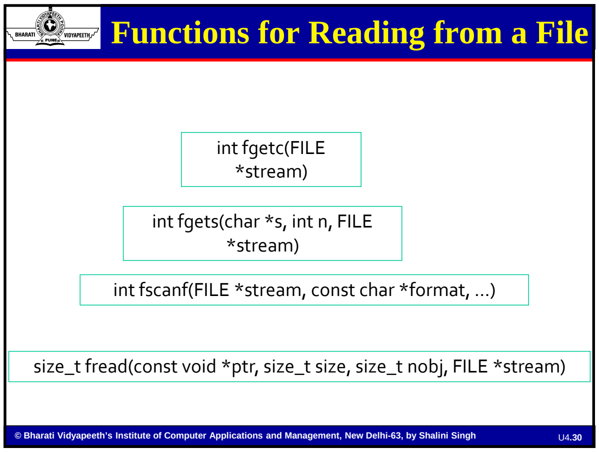
Mode	Operations Allowed	Action
r	Read	Return NULL if file doesn't exist
w	Write	Create if file doesn't exist. Destroy file contents if it exists
a	Write at end	Create if file doesn't exist. Retain old contents
r+	Read Write	Return NULL if file doesn't exist
w+	Read Write	Create if file doesn't exist. Destroy file contents if it exists
a+	Read, Write-at-end	Create if file doesn't exist. Retain old contents



Insertion in Files

- Writing of new records will be done in a sequential manner at the end of the file.
- Content initially stored will be overwritten
- If we want to insert a new record at ith position then we'll have to copy all records (i-n) to a temporary file.

Aces
Boilermakers
Devils
Flyers
Hawkeyes
Hoosiers
. . .
Minors
Panthers
. . .
Seminoles
. . .



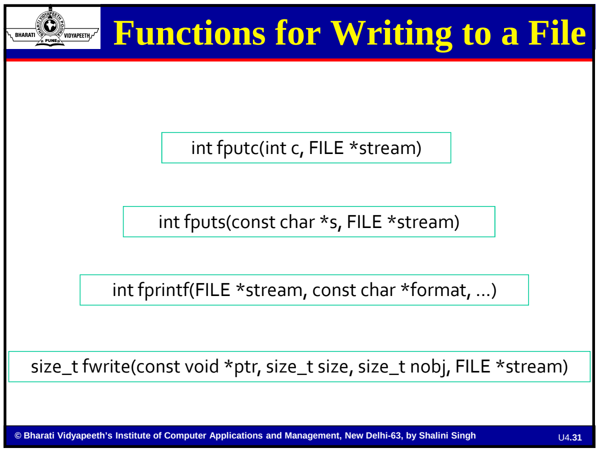
Functions for Reading from a File

int fgetc(FILE *stream)

int fgets(char *s, int n, FILE *stream)

int fscanf(FILE *stream, const char *format, ...)

size_t fread(const void *ptr, size_t size, size_t nobj, FILE *stream)



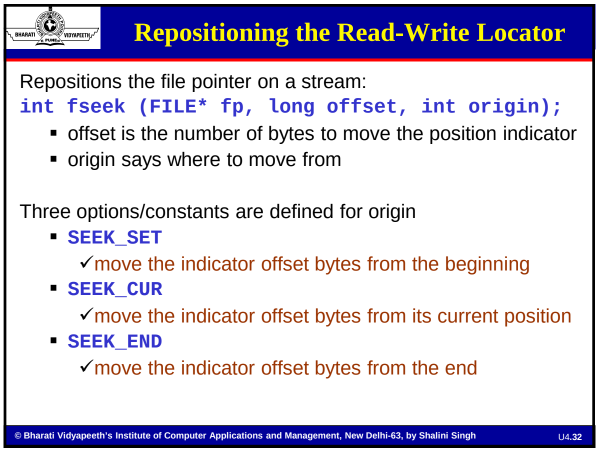
Functions for Writing to a File

```
int fputc(int c, FILE *stream)

int fputs(const char *s, FILE *stream)

int fprintf(FILE *stream, const char *format, ...)

size_t fwrite(const void *ptr, size_t size, size_t nobj, FILE *stream)
```



Repositioning the Read-Write Locator

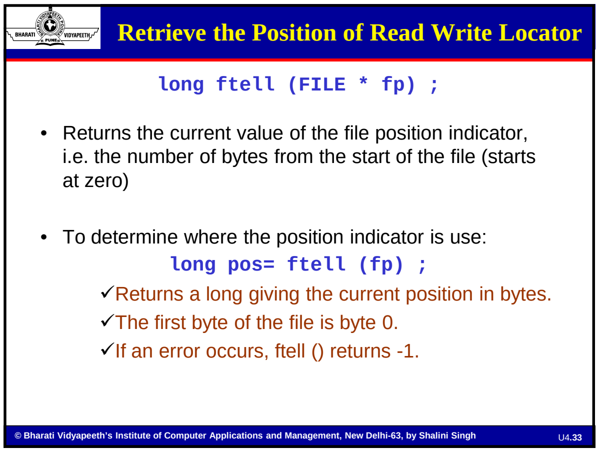
Repositions the file pointer on a stream:

```
int fseek (FILE* fp, long offset, int origin);
```

- offset is the number of bytes to move the position indicator
- origin says where to move from

Three options/constants are defined for origin

- `SEEK_SET`
  - ✓move the indicator offset bytes from the beginning
- `SEEK_CUR`
  - ✓move the indicator offset bytes from its current position
- `SEEK_END`
  - ✓move the indicator offset bytes from the end



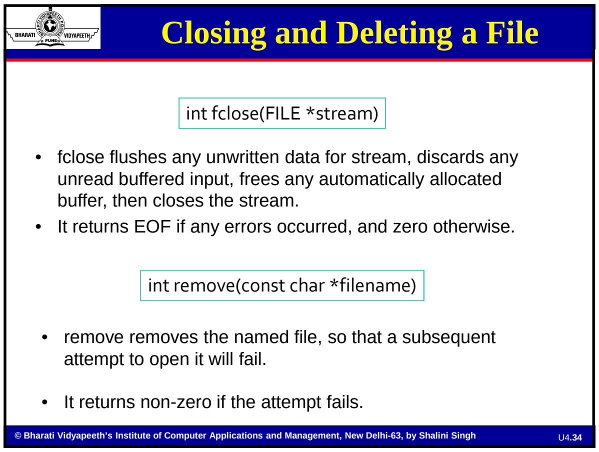
Retrieve the Position of Read Write Locator

```
long ftell (FILE * fp) ;
```

- Returns the current value of the file position indicator, i.e. the number of bytes from the start of the file (starts at zero)
- To determine where the position indicator is use:

```
long pos= ftell (fp) ;
```

  - ✓Returns a long giving the current position in bytes.
  - ✓The first byte of the file is byte 0.
  - ✓If an error occurs, ftell () returns -1.



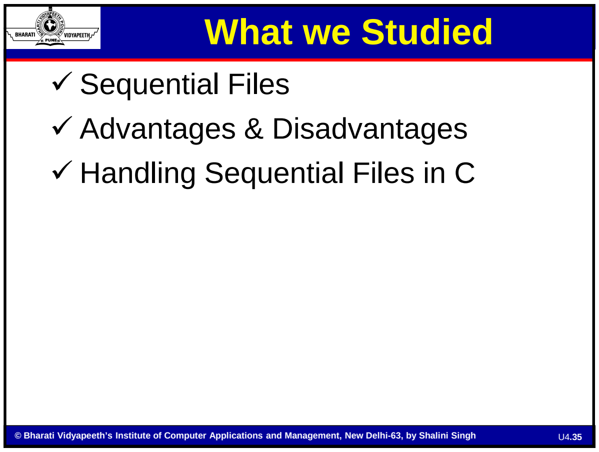
Closing and Deleting a File

```
int fclose(FILE *stream)
```

- fclose flushes any unwritten data for stream, discards any unread buffered input, frees any automatically allocated buffer, then closes the stream.
- It returns EOF if any errors occurred, and zero otherwise.

```
int remove(const char *filename)
```

- remove removes the named file, so that a subsequent attempt to open it will fail.
- It returns non-zero if the attempt fails.

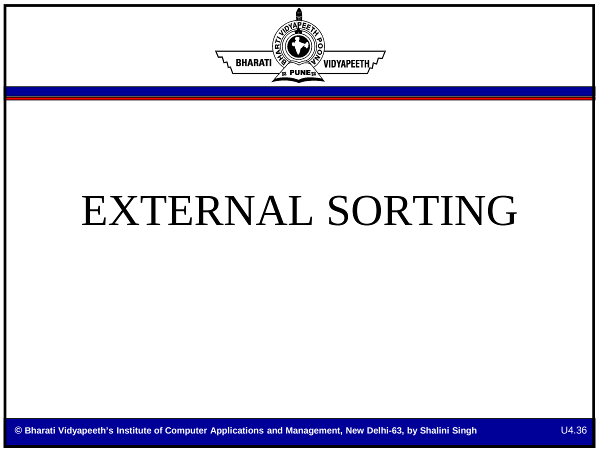


What we Studied

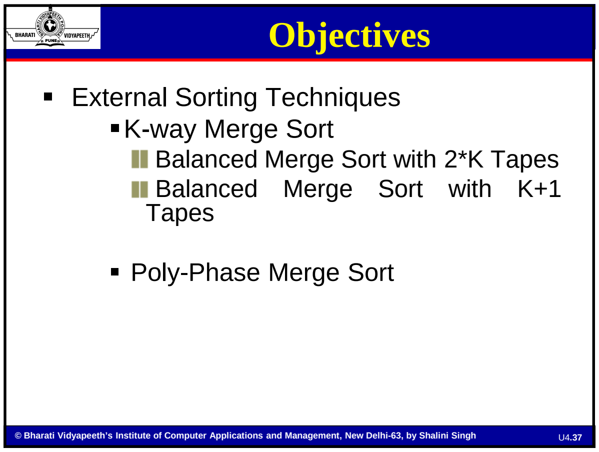
✓ Sequential Files

✓ Advantages & Disadvantages

✓ Handling Sequential Files in C

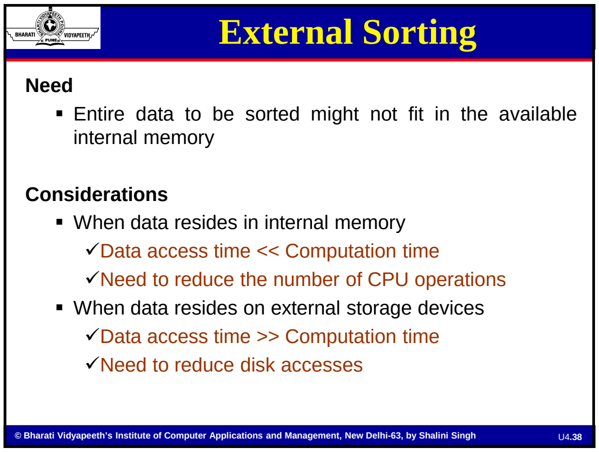


EXTERNAL SORTING



Objectives

- External Sorting Techniques
  - K-way Merge Sort
    - Balanced Merge Sort with 2*K Tapes
    - Balanced Merge Sort with K+1 Tapes
  - Poly-Phase Merge Sort



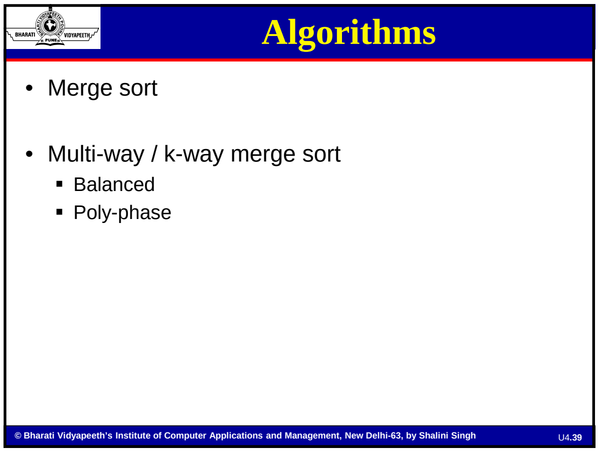
External Sorting

Need

- Entire data to be sorted might not fit in the available internal memory

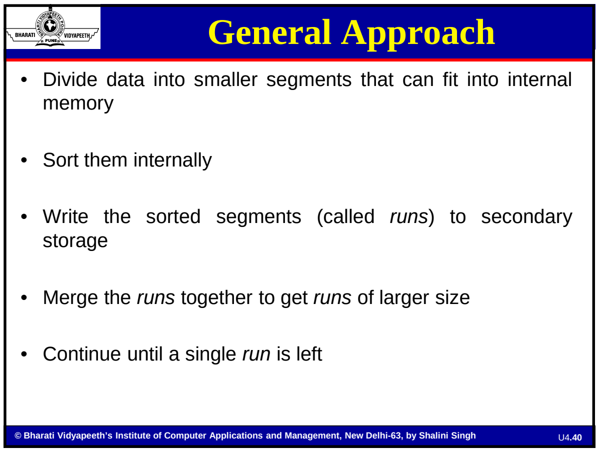
Considerations

- When data resides in internal memory
  - ✓Data access time << Computation time
  - ✓Need to reduce the number of CPU operations
- When data resides on external storage devices
  - ✓Data access time >> Computation time
  - ✓Need to reduce disk accesses



Algorithms

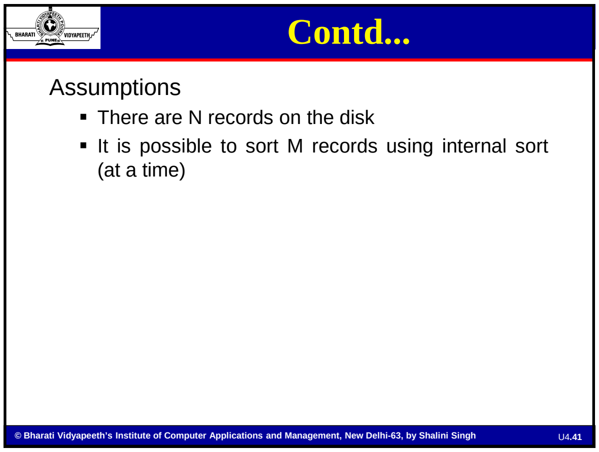
- Merge sort
- Multi-way / k-way merge sort
  - Balanced
  - Poly-phase



General Approach

- Divide data into smaller segments that can fit into internal memory
- Sort them internally
- Write the sorted segments (called *runs*) to secondary storage
- Merge the *runs* together to get *runs* of larger size
- Continue until a single *run* is left

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Shalini Singh U4.40

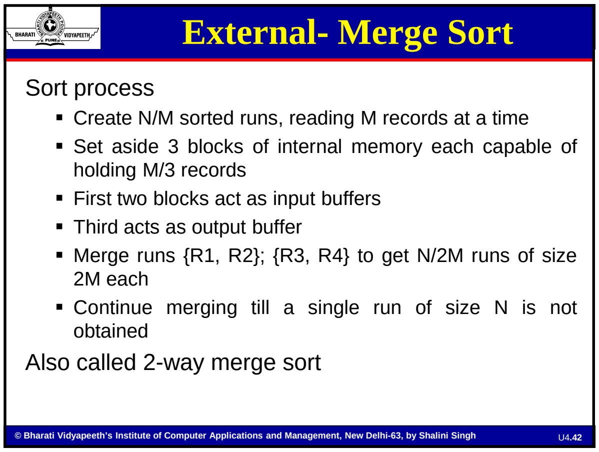


Contd...

Assumptions

- There are N records on the disk
- It is possible to sort M records using internal sort (at a time)

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Shalini Singh U4.41



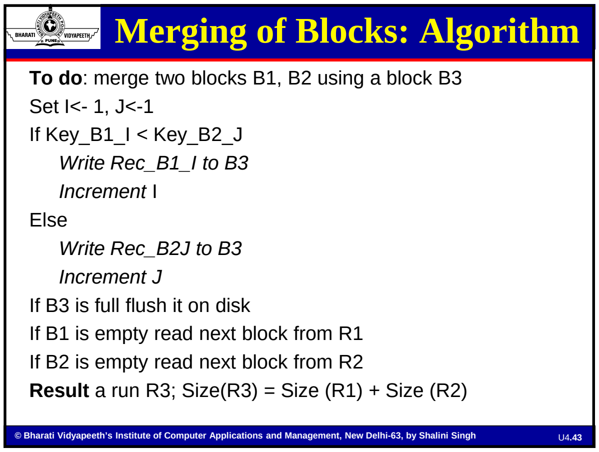
External- Merge Sort

Sort process

- Create N/M sorted runs, reading M records at a time
- Set aside 3 blocks of internal memory each capable of holding M/3 records
- First two blocks act as input buffers
- Third acts as output buffer
- Merge runs {R1, R2}; {R3, R4} to get N/2M runs of size 2M each
- Continue merging till a single run of size N is not obtained

Also called 2-way merge sort

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Shalini Singh U4.42

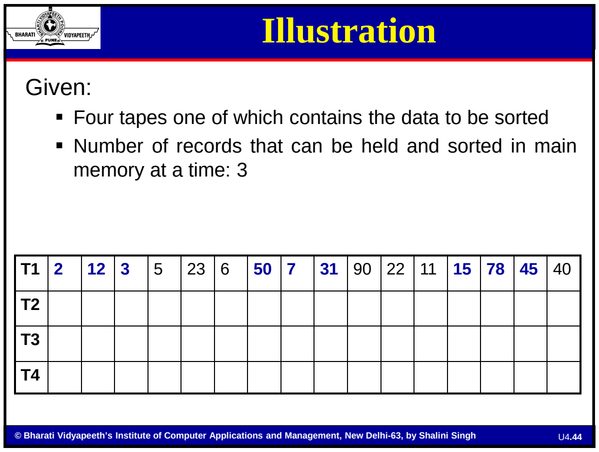


Merging of Blocks: Algorithm

To do: merge two blocks B1, B2 using a block B3

```
Set I<- 1, J<-1
If Key_B1_I < Key_B2_J
    Write Rec_B1_I to B3
    Increment I
Else
    Write Rec_B2J to B3
    Increment J
If B3 is full flush it on disk
If B1 is empty read next block from R1
If B2 is empty read next block from R2
```

Result a run R3; Size(R3) = Size (R1) + Size (R2)

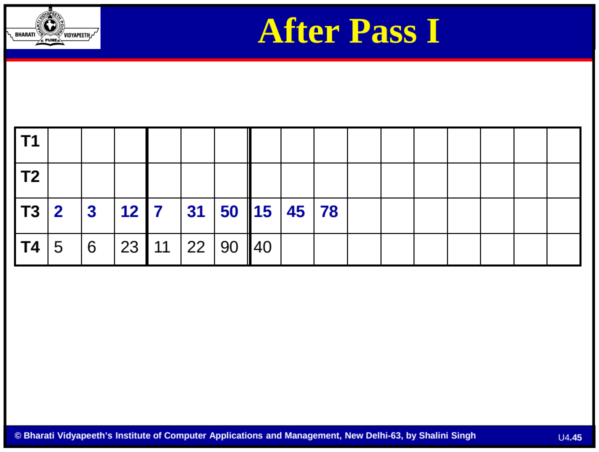


Illustration

Given:

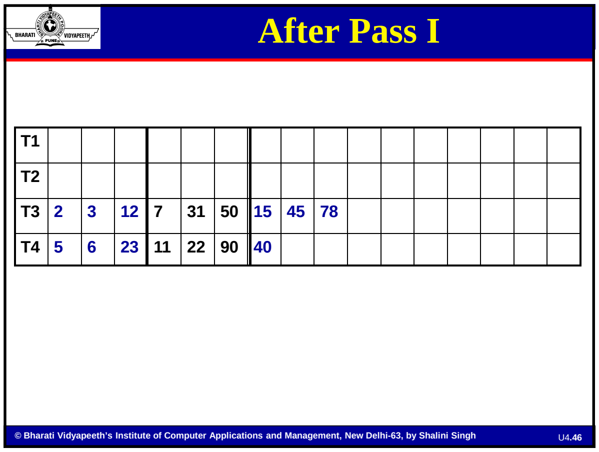
- Four tapes one of which contains the data to be sorted
- Number of records that can be held and sorted in main memory at a time: 3

T1	**2**	**12**	**3**	5	23	6	**50**	**7**	**31**	90	22	11	**15**	**78**	**45**	40
T2																
T3																
T4																



After Pass I

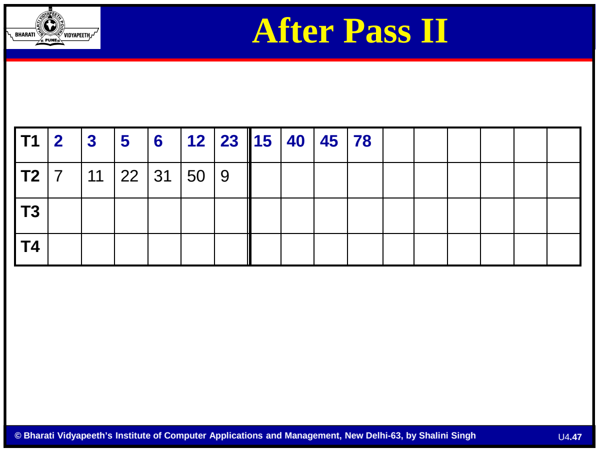
T1																
T2																
T3	**2**	**3**	**12**	**7**	**31**	**50**	**15**	**45**	**78**							
T4	5	6	23	11	22	90	40									



After Pass I

T1																
T2																
T3	2	3	12	7	31	50	15	45	78							
T4	5	6	23	11	22	90	40									

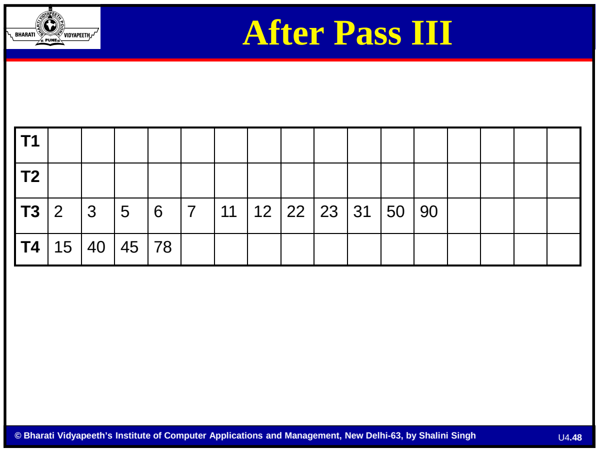
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Shalini Singh



After Pass II

T1	2	3	5	6	12	23	15	40	45	78						
T2	7	11	22	31	50	9										
T3																
T4																

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Shalini Singh

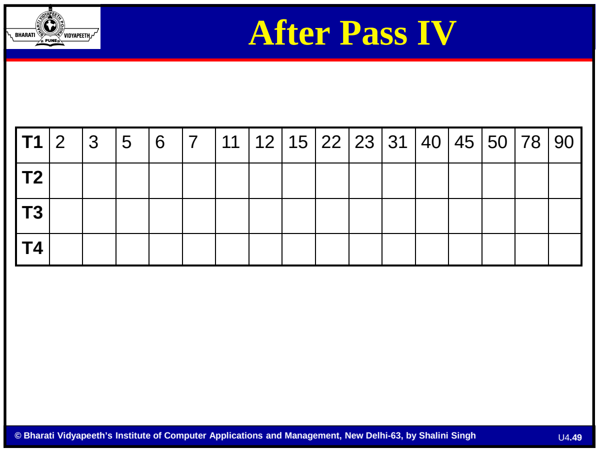


After Pass III

T1																
T2																
T3	2	3	5	6	7	11	12	22	23	31	50	90				
T4	15	40	45	78												

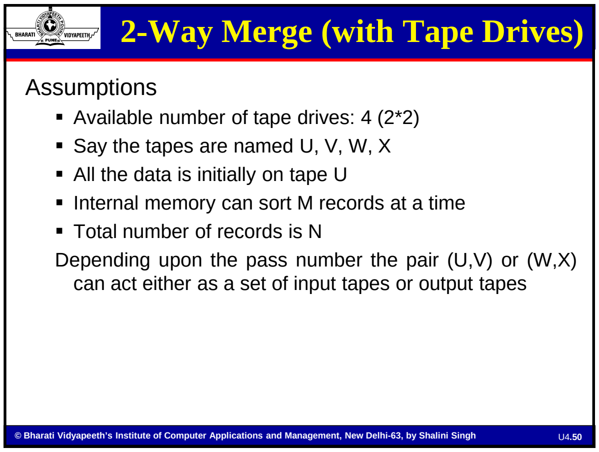
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Shalini Singh

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Shalini Singh



After Pass IV

T1	2	3	5	6	7	11	12	15	22	23	31	40	45	50	78	90
T2																
T3																
T4																

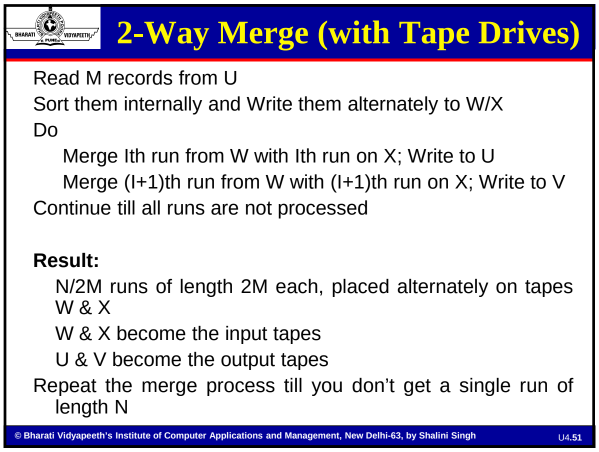


2-Way Merge (with Tape Drives)

Assumptions

- Available number of tape drives: 4 (2*2)
- Say the tapes are named U, V, W, X
- All the data is initially on tape U
- Internal memory can sort M records at a time
- Total number of records is N

Depending upon the pass number the pair (U,V) or (W,X) can act either as a set of input tapes or output tapes



2-Way Merge (with Tape Drives)

```
Read M records from U
Sort them internally and Write them alternately to W/X
Do
    Merge Ith run from W with Ith run on X; Write to U
    Merge (I+1)th run from W with (I+1)th run on X; Write to V
Continue till all runs are not processed
```

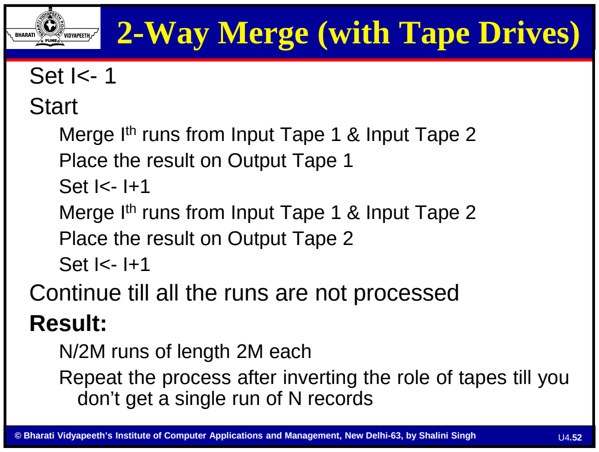
Result:

N/2M runs of length 2M each, placed alternately on tapes W & X

W & X become the input tapes

U & V become the output tapes

Repeat the merge process till you don't get a single run of length N



2-Way Merge (with Tape Drives)

Set I<- 1

Start

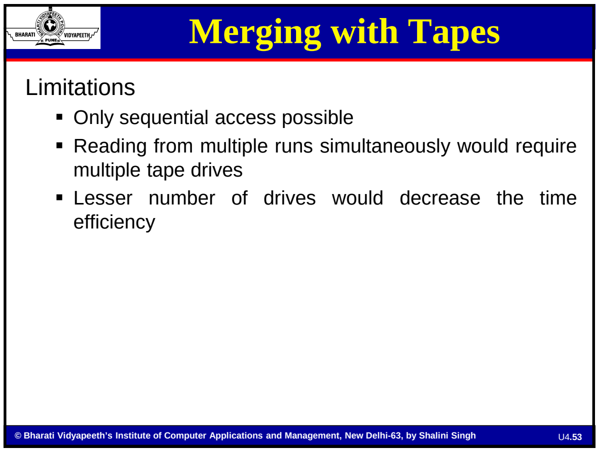
- Merge Ith runs from Input Tape 1 & Input Tape 2
- Place the result on Output Tape 1
- Set I<- I+1
- Merge Ith runs from Input Tape 1 & Input Tape 2
- Place the result on Output Tape 2
- Set I<- I+1

Continue till all the runs are not processed

Result:

- N/2M runs of length 2M each
- Repeat the process after inverting the role of tapes till you don't get a single run of N records

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Shalini Singh U4.52

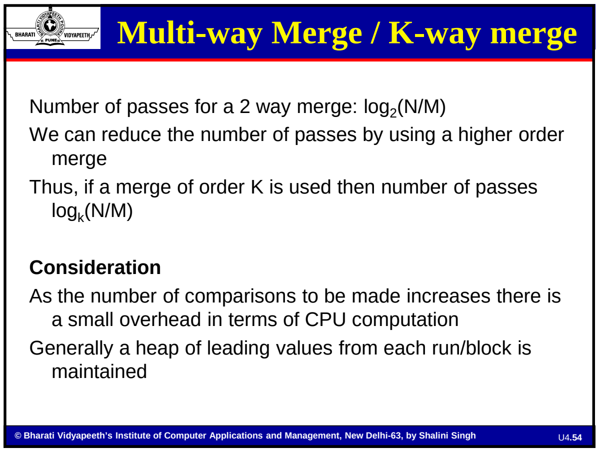


Merging with Tapes

Limitations

- Only sequential access possible
- Reading from multiple runs simultaneously would require multiple tape drives
- Lesser number of drives would decrease the time efficiency

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Shalini Singh U4.53



Multi-way Merge / K-way merge

Number of passes for a 2 way merge: $\log_2(N/M)$

We can reduce the number of passes by using a higher order merge

Thus, if a merge of order K is used then number of passes $\log_k(N/M)$

Consideration

As the number of comparisons to be made increases there is a small overhead in terms of CPU computation

Generally a heap of leading values from each run/block is maintained

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Shalini Singh U4.54

Contd...

Say total number of passes is P

M * (2 * 2 * 2 … P times) = N

M * 2^P = N

Log $_2$ (N/M) = P

K-way Merge Sort using 2*K tapes

Motivation

Need for better performance

Strategy

Increase the number of runs that are merged at a time

Say, K runs are merged at a time

Requirement

According to the above algorithm we require K input tapes and K output tapes => 2*K number of tape drives

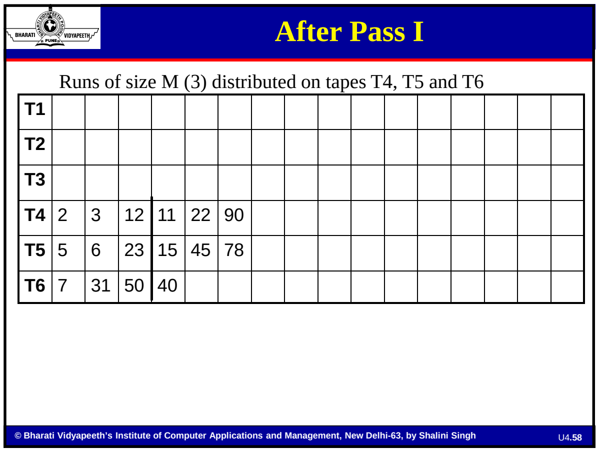
Number of passes

Log $_k$ (N/M) = P

Illustration: 3-Way Merge using 6 tapes

Requirement 2*3 Tapes Say, M=3

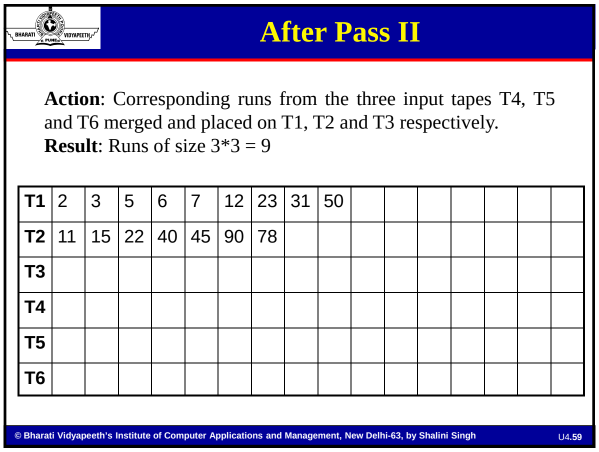
T1	2	12	3	5	23	6	50	7	31	90	22	11	15	78	45	40
T2																
T3																
T4																
T5																
T6																



After Pass I

Runs of size M (3) distributed on tapes T4, T5 and T6

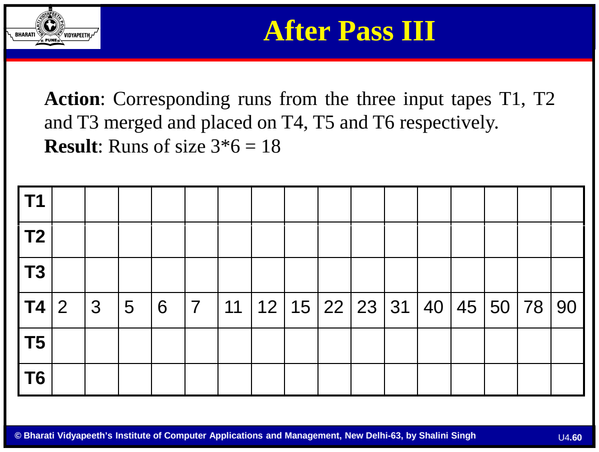
T1																
T2																
T3																
T4	2	3	12	11	22	90										
T5	5	6	23	15	45	78										
T6	7	31	50	40												



After Pass II

Action: Corresponding runs from the three input tapes T4, T5 and T6 merged and placed on T1, T2 and T3 respectively.
Result: Runs of size 3*3 = 9

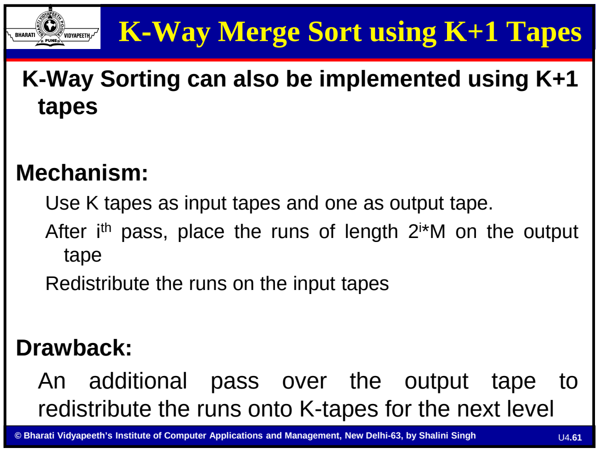
T1	2	3	5	6	7	12	23	31	50							
T2	11	15	22	40	45	90	78									
T3																
T4																
T5																
T6																



After Pass III

Action: Corresponding runs from the three input tapes T1, T2 and T3 merged and placed on T4, T5 and T6 respectively.
Result: Runs of size 3*6 = 18

T1																
T2																
T3																
T4	2	3	5	6	7	11	12	15	22	23	31	40	45	50	78	90
T5																
T6																



K-Way Merge Sort using K+1 Tapes

K-Way Sorting can also be implemented using K+1 tapes

Mechanism:

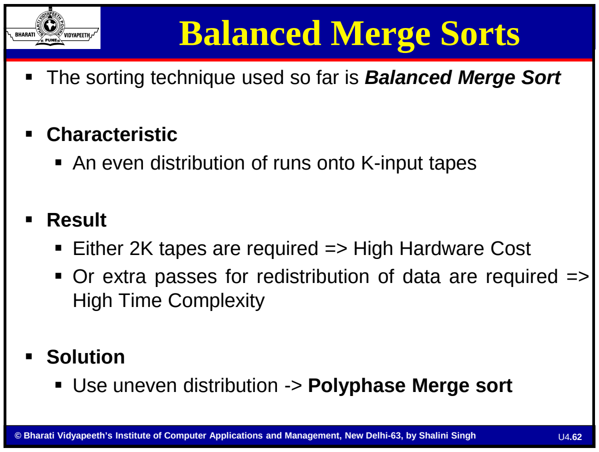
Use K tapes as input tapes and one as output tape.

After i^{th} pass, place the runs of length 2^{i}*M on the output tape

Redistribute the runs on the input tapes

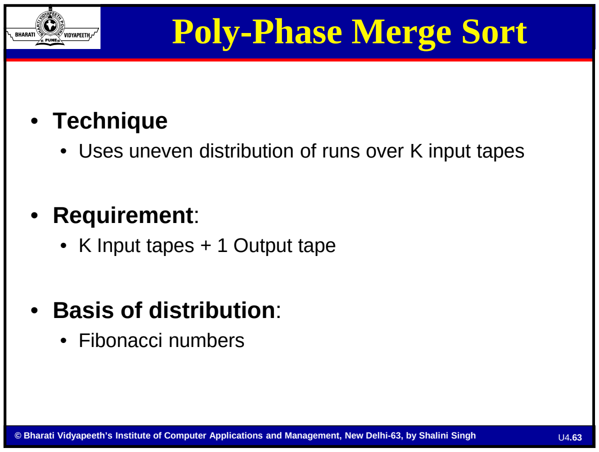
Drawback:

An additional pass over the output tape to redistribute the runs onto K-tapes for the next level



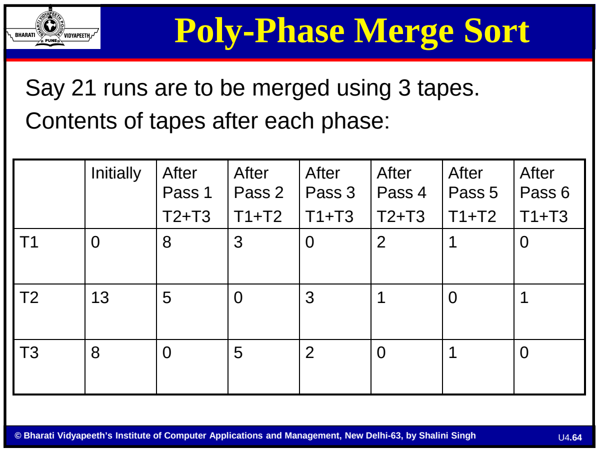
Balanced Merge Sorts

- The sorting technique used so far is ***Balanced Merge Sort***
- **Characteristic**
  - An even distribution of runs onto K-input tapes
- **Result**
  - Either 2K tapes are required => High Hardware Cost
  - Or extra passes for redistribution of data are required => High Time Complexity
- **Solution**
  - Use uneven distribution -> **Polyphase Merge sort**



Poly-Phase Merge Sort

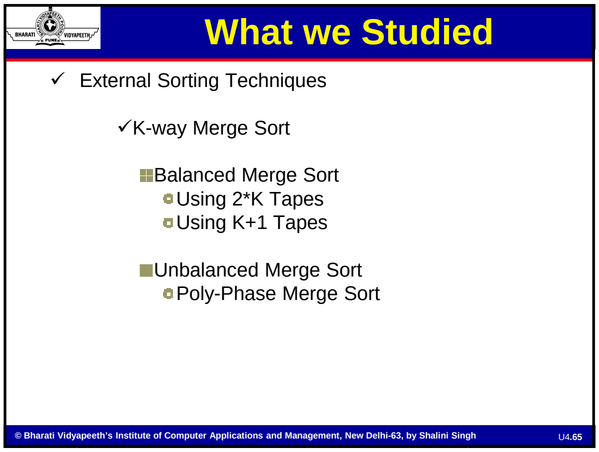
- **Technique**
  - Uses uneven distribution of runs over K input tapes
- **Requirement**:
  - K Input tapes + 1 Output tape
- **Basis of distribution**:
  - Fibonacci numbers



Poly-Phase Merge Sort

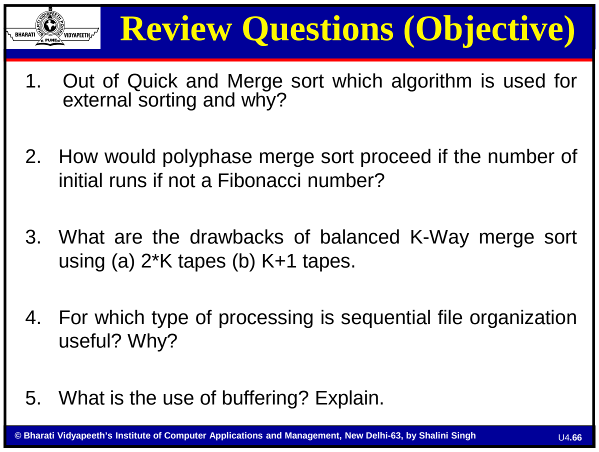
Say 21 runs are to be merged using 3 tapes.
Contents of tapes after each phase:

	Initially	After Pass 1 T2+T3	After Pass 2 T1+T2	After Pass 3 T1+T3	After Pass 4 T2+T3	After Pass 5 T1+T2	After Pass 6 T1+T3
T1	0	8	3	0	2	1	0
T2	13	5	0	3	1	0	1
T3	8	0	5	2	0	1	0



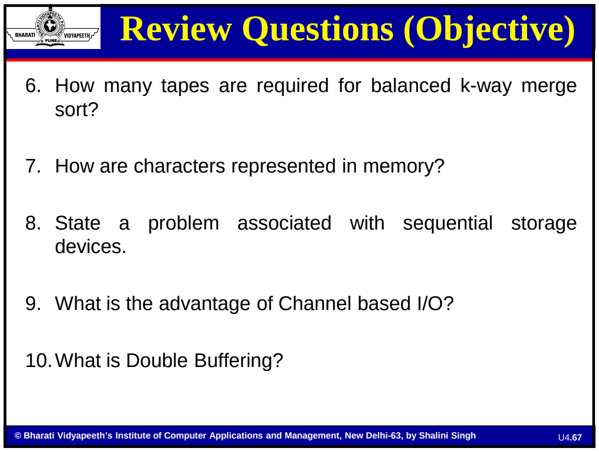
What we Studied

- ✓ External Sorting Techniques
  - ✓K-way Merge Sort
    - Balanced Merge Sort
      - Using 2*K Tapes
      - Using K+1 Tapes
    - Unbalanced Merge Sort
      - Poly-Phase Merge Sort



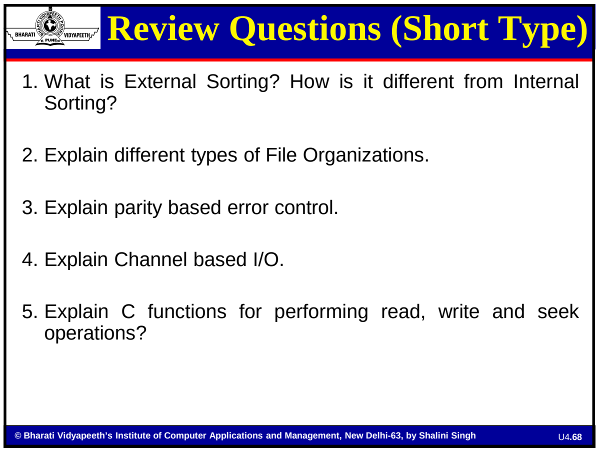
Review Questions (Objective)

1. Out of Quick and Merge sort which algorithm is used for external sorting and why?
2. How would polyphase merge sort proceed if the number of initial runs if not a Fibonacci number?
3. What are the drawbacks of balanced K-Way merge sort using (a) 2*K tapes (b) K+1 tapes.
4. For which type of processing is sequential file organization useful? Why?
5. What is the use of buffering? Explain.



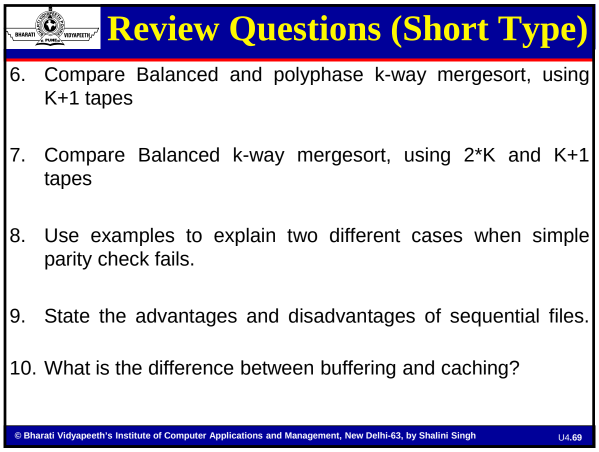
Review Questions (Objective)

6. How many tapes are required for balanced k-way merge sort?
7. How are characters represented in memory?
8. State a problem associated with sequential storage devices.
9. What is the advantage of Channel based I/O?
10. What is Double Buffering?



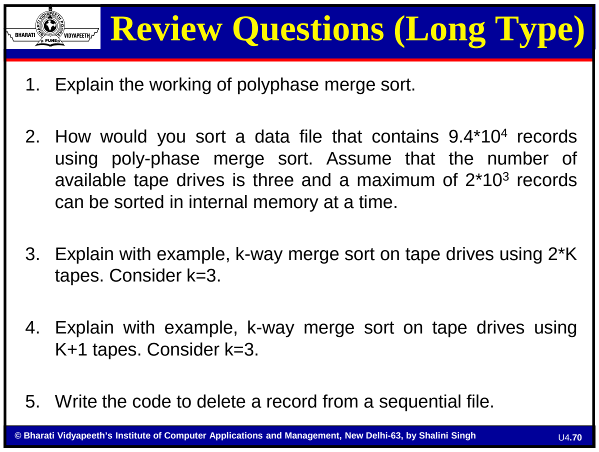
Review Questions (Short Type)

1. What is External Sorting? How is it different from Internal Sorting?
2. Explain different types of File Organizations.
3. Explain parity based error control.
4. Explain Channel based I/O.
5. Explain C functions for performing read, write and seek operations?



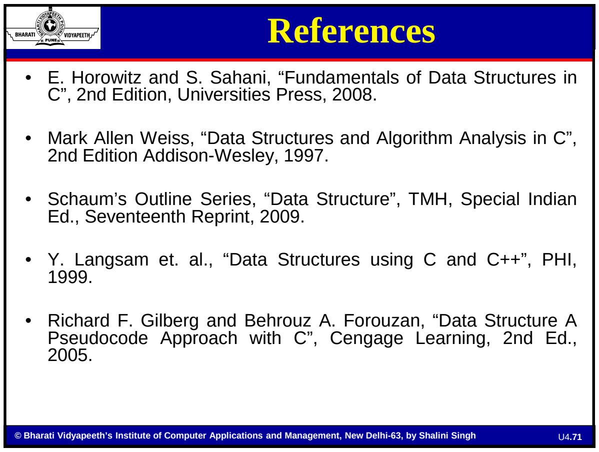
Review Questions (Short Type)

6. Compare Balanced and polyphase k-way mergesort, using K+1 tapes
7. Compare Balanced k-way mergesort, using 2*K and K+1 tapes
8. Use examples to explain two different cases when simple parity check fails.
9. State the advantages and disadvantages of sequential files.
10. What is the difference between buffering and caching?



Review Questions (Long Type)

1. Explain the working of polyphase merge sort.
2. How would you sort a data file that contains $9.4*10^4$ records using poly-phase merge sort. Assume that the number of available tape drives is three and a maximum of $2*10^3$ records can be sorted in internal memory at a time.
3. Explain with example, k-way merge sort on tape drives using 2*K tapes. Consider k=3.
4. Explain with example, k-way merge sort on tape drives using K+1 tapes. Consider k=3.
5. Write the code to delete a record from a sequential file.



References

- E. Horowitz and S. Sahani, "Fundamentals of Data Structures in C", 2nd Edition, Universities Press, 2008.
- Mark Allen Weiss, "Data Structures and Algorithm Analysis in C", 2nd Edition Addison-Wesley, 1997.
- Schaum's Outline Series, "Data Structure", TMH, Special Indian Ed., Seventeenth Reprint, 2009.
- Y. Langsam et. al., "Data Structures using C and C++", PHI, 1999.
- Richard F. Gilberg and Behrouz A. Forouzan, "Data Structure A Pseudocode Approach with C", Cengage Learning, 2nd Ed., 2005.