Page No.
Date

Dr.D.Y.PATIL EDUCATIONAL FEDERATION Varale Campus

Practical No: 5 (B-14)

Theory:

1) What is sorting? Explain concepts of sorting?

→ Sorting is the process of arranging elements in a specific order, typically in ascending or descending order.
In data structure, sorting is an essential operation that organizes data for efficient searching, retrieval and analysis.

2) What are types of sorting? list sorting techniques.

→ Types of sorting:
1. Ascending order - Elements are sorted from smallest to largest.
2. Descending order - Elements are arranged from largest to smallest.

Sorting techniques:
1) Bubble sort
2) Selection sort.
3) Insertion sort.
4) Merge sort.
5) Quick sort.
6) Bucket sort.

Q.3) Explain bubble sort with examples, advantages, disadvantages, time complexity.

→ i) Bubble sort is a simple comparison based sorting algorithm where adjacent elements are compared and swapped if they are in wrong order.

ii) This process repeats until the entire list is sorted.

iii) Algorithm:

1) Start from first element.
2) Compare each pair of adjacent elements.
3) Swap them if they are in wrong order.
4) Repeat the process for each element until no swaps are needed.

iv) Example:

initial array: [5, 1, 4, 2, 8]

Pass 1: [1, 4, 2, 5, 8]

Pass 2: [1, 2, 4, 5, 8]

Sorted after two passes.

v) Advantages:
- Simple & easy to understand.
- Works well with small datasets.
- No additional memory needed.

vi) Disadvantages:
- Inefficient for large datasets.
- High time complexity.

vii) Time Complexity : $(O(n)^2)$

Q. 4) Explain seleetion sort with example, advantages, disadvantages, time complexity.

→ i) Selection sort is a comparison based sorting algorithm where the smallest element is repeatedly seleeted from the unsorted portion of the array and swapped with the first unsorted elements.

ii) Algorithm:
1) Find the smallest element in the unsorted portion of the array.
2) Swap it with the first element of the unsorted portion.
3) Move to the next position and repeat the process for the remaining unsorted part.
4) Continue until entire array is sorted.

iii) Example:
initial array: [29, 10, 14, 37, 13]

Pass 1: [10, 29, 14, 37, 13]
Pass 2: [10, 13, 14, 37, 29]
Pass 3: [10, 13, 14, 37, 29]
Pass 4: [10, 13, 14, 29, 37]

iv) Advantages:
- Simple to understand & implement.
- Does not require additional memory.

v) Disadvantages:
- Inefficient for large datasets.
- Performs unnecessary comparisons even if the array is partially sorted.

vi) Time complexity: $O(n^2)$.

Page No.
Date

Algorithms:

a) To accept percentage from user for N number of students.

Step 1: Create an empty array 'percentages'
Step 2: For i=1 to N do
    i) Input percentage for student i.
    ii) Append the input percentage to array.
Step 3: Return 'percentages'.

b) To perform bubble sort & print sorted elements:

Input: Array 'arr' containing percentage of students.

Output: Sorted array in ascending order.

Step 1: for i=1 to length(arr) - 1 do
Step 2: Set key = arr[i]
Step 3: Set j = i-1
    while j>=0 and arr[j]> key
    do

Step 4: set arr[j+1] = arr[j].

Step 5: Decrease j by 1.

Step 6: Set arr[j+1] = key.

Step 7: Print the sorted array 'arr'.

c) To perform selection sort and print sorted elements.

Step 1: Input arr (array of percentages)
Step 2: for i from 1 to length of arr-1:
set key = arr[i]
set j = i-1
while j >= 0 & arr[i] > key
set arr[j+1] = arr[j]
Step 3: Decrement j.
set arr[j+1] = key
Step 4: Output sorted array.

Dr.D.Y.PATIL EDUCATIONAL FEDERATION Varale Campus

Practical: 6 (B-15)

Theory:

Q.i) Explain insertion sort with examples, advantages, disadvantages, time complexity.

→ i) Insertion sort is a simple sorting algorithm that builds a sorted array one element at a time by repeatedly taking the next unsorted element & inserting it into its correct position.

ii) Algorithm:

1) Start with the second element, treat it as a key.
2) Compare it with elements in the sorted portion.
3) Shift larger elements to the right.
4) Insert the key at the correct position.
5) Repeat for all elements.

iii) Example:

initial array: [5, 2, 9, 1, 5]

Pass 1: [2, 5, 9, 1, 5]
Pass 2: [2, 5, 9, 1, 5]
Pass 3: [1, 2, 5, 9, 5]
Pass 4: [1, 2, 5, 5, 9]

iv) Advantages:
- Efficient for small datasets.
- In place sorting (no memory req.).

v) Disadvantages:
- High time complexity.
- Slower than advanced algorithms like merge sort, etc.

vi) Time complexity: $O(n^2)$

Q.2) Explain shell sort with examples, advantages, disadvantages, time complexity.

→ i) Shell sort is advanced version of insertion sort. It improves the efficiency by breaking the original list into smaller sublists.

ii) Algorithm:
1) Start with a large gap.
2) Divide the list into sublists, each consisting of elements that are a gap distance apart.
3) Perform insertion sort on each sublist.
4) Reduce the gap and repeat the process.
5) Continue until the gap becomes 1, then do the final insertion.

iii) Advantages:
- faster than insertion sort for larger datasets.
- can handle medium sized data more efficiently.

iv) Disadvantages:
- Performance depends on the gap sequence chosen.
- slightly complex to understand.

v) Time complexity:
O(n^1.5) to O(n log n).

3) Difference betn insertion & shell sorting.

Insertion sort	Shell sort
i) Sorts elements one at a time by inserting them into their correct position in the sorted part.	i) An advanced version of insertion sort that first sorts elements far apart then reduces the gap.

Page No.
Date

	Insertion sort	Shell sort.
ii)	Compares and sorts adjacent elements	ii) Starts with a larger gap and reduces the gap.
iii)	Inefficient for larger datasets.	iii) More efficient for larger datasets.
iv)	Stable sorting algorithm	iv) Not stable by default.
v)	Time complexity $O(n^2)$	iv) Time complexity $O(n \log n)$.

Page No.
Date

Practical No: 07 (B-16)

Theory:

Q.1) Explain quick sort with examples, advantages time complexity.

→ i) Quick sort is a divide & conquer algorithm that sorts by selecting a pivot element, partitioning the array into two sub arrays and recursively sorting them.

ii) Example:

Arrays: [8, 3, 1, 7, 0, 10, 2]

1) Select pivot - 7.

2) Partition [3, 1, 0, 2] , [10, 8]

3) Recursively sort the sub arrays.

- [0, 1, 2, 3, 7, 8, 10]

iii) Advantages:

- Efficient for large datasets.
- Inplace sorting.

iv) Disadvantages:

- Not stable (order of equal elements may change)

v) Time complexity:

Best case: $O(n \log n)$

Worst case: $O(n^2)$

Q.2) Explain algorithm to perform quick sort.

→ i) Choose a Pivot: Select an element as pivote. This can be any element (Commonly middle element).

ii) Partitioning: Rearrange the array so that - All elements smaller than the pivot are on the left.
- All elements greater than the pivot are on the right now.
- Pivot is at the correct position.

iii) Recursion:
Recursively apply the same process to the right and left subarrays. (Excluding the pivot).

Q.3) Solve the examples of quick sort:

1) Array [25, 82, 17, 23, 38, 7, 64, 86, 21]

→

1) Pivot: 21

∴ [17, 7, 21, 23, 38, 64, 86, 82, 25]

2) Left subarray: [17, 7]
Pivot: 7.

[7, 17]

Array sorted.

3) Right sub-array: [23, 38, 64, 86, 82, 25]

Pivot: 25

[23, 25, 64, 86, 82, 38]

4) Right sub-array:

[64, 86, 82, 38]

Pivot: 38

[38, 64, 86, 82]

5) Right subarray:

[64, 86, 82]

Pivote: 82

[64, 82, 86]

Final sorted array: [7, 17, 21, 23, 25, 38, 64, 82, 86]

2) Array: [39, 9, 81, 45, 90, 27, 72, 18]

-> 1) Choose pivot 18.

[9, 18, 81, 45, 90, 27, 72, 18]

2) Pivote: 39

[27, 39, 45, 90, 72, 81]

3) Rigth subarray: [45, 90, 72, 81]

Pivot : 81

[45, 72, 81, 90]

Final sorted : [09, 18, 27, 39, 45, 72, 81, 90]

array

Page No.
Date

Dr.D.Y.PATIL EDUCATIONAL FEDERATION Varale Campus

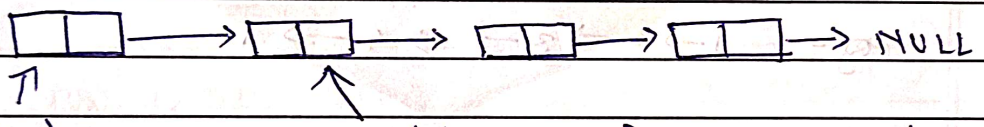
Practical No: 08 (C-19)

Theory:

Q.1) What is linked list? Representation, Advantages, disadvantages, applications.

→ i) Linked list is a linear data structure where elements, called nodes are linked using pointers. Each node contains data and a reference to the next node in the sequence.

ii) Representation of Linked list:



iii) Advantages:
- Dynamic size.
- Efficient insertion/deletion, nos shift required.

iv) Disadvantages:
- More memory required for pointers.
- Sequential access.

v) Applications:
- Dynamic memory allocation.
- Implementing stacks, queues & graphs.

or

Q.2) List types of linked list with example.

→ 1) singly linked list: Each node points to next node.

eg - 1 → 2 → 3 → 4 → NULL

2) Doubly linked list:
Each node points to both the next and the previous node.

eg - NULL ← 1 ↔ 2 ↔ 3 → NULL

3) Circular linked list:
The last node points back to first node.

eg - 1 → 2 → 3 → 4 → 1

Q.3) Explain singly linked list as an ADT,

→ Operations on singly lists include:

1) Insert: Add a node at the beginning end or specific position.
2) Delete: Remove a node from the beginning, end or specific position.
3) Search: Find a node with specific node.
4) Traverse: visit all nodes in sequence.

Page No.
Date

Dr. D. Y. PATIL EDUCATIONAL FEDERATION Varale Campus

Q.4) Difference betn array & linked list.

	Array	linked list.
i)	Fixed size	i) Dynamic size
ii)	Random access	ii) sequential access
iii)	Contiguous memory allocation	iii) Non-contiguous memory location.
iv)	Insertion/deletion is difficult.	iv) Efficient insertion/deletion.
v)	Requires shifting of elements.	v) No need for shifting.

Page No.
Date

Dr.D.Y.PATIL EDUCATIONAL FEDERATION Varale Campus

Practical No : 09 (C-20)

Theory :

Q.1) What is doubly circual linked list? Representation, advantages, applications?

→ i) A doubly circual linked list where each node has two pointers : one pointing to the next node and one to previous node.

ii) Additionally, the last node points to the first node forming a circular structure in both directions.

iii) Representation of double circular linked list :

[Prev | Data | Next] ⟷ [Prev | Data | Next]

(back to first address) ⟷ [Next | Data | Prev]

iv) Advantages :
- Bidirectional traversal.
- Circual nature allows continuous traversal from any node.

v) Disadvantages :
- Complex to implement.
- More memory required.

vi) Applications:
- Implementation of music/video playlists
- Circular queues, buffer management.

Q.2) Explain doubly linked list as an ADT.
→ Operation include:
i) Insert: Add a node at the beginning, end, or a specific position.
ii) Delete: Remove a node from the beginning, end, or specific position.
iii) Traverse forward: Visit all nodes from the first to the last in sequence.
iv) Traverse backwards: Visit all nodes from the last to the first in reverse.

Q.3) Difference betn singly & doubly circular linked lists:

Singly circular linked list	Doubly circular linked list
i) Each node has one pointer	i) Each node has two pointers.
ii) Only forward traversal is possible	ii) Both forward & backward traversal.
iii) Less memory overhead	iii) More memory req.
iv) Simple to implement	iv) More complex to implement.

Page No.
Date

Dr.D.Y.PATIL EDUCATIONAL FEDERATION Varale Campus

Practical No : 10 (C-21)

Theory :

Q.1) Write short note on linked list.

→ A linked list is a dynamic data structure consisting of nodes. Each node holds data and a reference to the next node in sequence. Linked list can dynamically grow or shrink, making them suitable for ~~suit~~ situations where the number of ~~em~~ elements may change frequently. Common types include single linked list, Doubly linked list & circular linked list.

Q.2) Explain logic to book and cancel appointment.

→ 1) Create a new node to store appointment details.
2) If the list is empty, insert the new node as the first node.
3) If the list is not empty, traverse the list to find the correct position (based on time/date) and insert the node at the appropriate spot.

- canceling an appointment :
1) Search for the appointment in the list using the provided appointment details.
2) If found, adjust the pointers of neighboring nodes to bypass the node containing the appointment.
3) Delete the node to free memory.

Q. 3) Explain logic for sorting appointment linked list.

→ Sorting logic - To sort appointments you can use insertion sort or merge sort.

Logic :
Merge sort for linked list.
Step 1 : divide the list into two halves
Step 2 : Recursively solve each half
Step 3 : Merge the two sorted halves

Practical No : 11 (D-26)

Theory :

Q.i) What is stack? Explain working of stack.

→ i) A stack is a linear data structure that follows LIFO principle. This means the last element added to the stack will be first one to be removed.

ii) A stack uses two primary operations. push and pop.

iii) Working of stack :
- Push operation : Adds an element to the top of the stack.
- Pop operation : Removes the topmost element from the stack.
- Peek / Top : Retrieves the top element without removing it.
- isEmpty : Checks if the stack is empty.

iv) Advantages :
- Simple to implement.
- Ideal for problems requiring LIFO order.

v) Disadvantages :
- Limited access to elements.
- Size may be limited.

vi) Applications:
- Expression evaluation.
- Function call management.
- Undo mechanisms in text editors.
- Depth-first search in graph algorithms.

Q.2) Explain stack as an ADT?
→ Operations on a stack include:
- Push(x): Insert element (x) into the stack.
- Pop(): Remove the top element from the stack.
- Peek(): Return the top element without removing it.
- isEmpty(): Check if the stack is empty.

Q.3) Explain check for balanced bracket expression using stack.
→ i) To check if an expression with brackets is balanced, use a stack.
1) Traverse the expression character by character.
2) Push opening brackets ({, [, ( ) into the stack.
3) Pop the the stack when encountering a closing bracket (},],) and check if it matches the top of the stack.

Page No.
Date

4) If all brackets match correctly and the stack is empty at the end, the expression is balanced.

Algorithm:

```
isBalanced(expression):
    stack = create empty stack.

    for char in expression:
        if char is '(', '{', '[':
            push(stack, char)
        else if char is ')', '}', ']':
            if stack is empty:
                return false
            top = pop(stack)
            if char does not match top:
                return false
    return stack is empty.
```

Page No.
Date

Q.4) Difference betn stack & Queue.

	Stack	Queue.
i)	A linear data structure that follows the LIFO principle.	i) A linear data structure that follows the FIFO.
ii)	Operation like push & pop are performed.	ii) operations like enqueue & Dequeue can be performed.
iii)	It has single end.	iii) It has two ends.
iv)	Time complexity O(1) for both push and pop operations.	iv) Time complexity for enqueue is O(1).

Page No.
Date

Dr.D.Y.PATIL EDUCATIONAL FEDERATION Varale Campus

Practical No : 12 (D-27)

Theory :

Q.1) What is polish notation? List different notations with example. Mention operator precedence in tabular form.

→ i) Polish or prefix notation is a mathematical notation in which operators precede their ~~opran~~ operands.

ii) It eliminates the need of parentheses to define operation precedence as the order of operations is fully determined by the placement of the operator.

1) Infix notation : The operator is placed between operands.
eg : A + B.

2) Prefix Notation : The operator comes befor the operands.
eg : + A B.

3) Postfix notation : The operator comes after the operands.
eg : A B +.

Operator	Description	Precedence
( )	Paranthesis	Highest
^	Exponentiation	2
*, /	multiplication	3
+, -	Add, subtract	4 (Low)

Q.2) Explain infix expression, Postfix expression with example.

→

i) Infix expression:
In an infix expression the operator is placed between operands.
Example: (A+B)*C

ii) Postfix expression:
In a postfix expression, the operator is placed after the operands, eliminating the need for parentheses.
Example: A B + C*

Q.3) Explain algorithms to convert infix to postfix with example.

→ Algorithm to convert infix to Postfix expression:

1) Initialize an empty stack for operators and an empty list for the output.
2) Scan each character of the infix expression from left to right:
a) Operand: Add it directly to the output list.
b) Operator: Pop operators from the stack to output the list if they have higher or equal precedence to

the current operater, then push the current operator into the stack.
c) Left Parenthesis (: Push it into the stack.
d) Right Parenthesis) : Pop from the stack until a left parenthesis is encountered; discard the left parenthesis.
3) Pop remaining operators in the stack to the output list.

Q. 4) Write algorithms for postfix expression evaluation with example.

→

1) Initialize an empty ~~array~~ stack.
2) Scan each character of the postfix expression from left to right.
- Operand: Push it into the stack.
- Operator: Pop the two elements from the stack, apply the operator and push the result back into the stack.

3) After scanning the expression the stack will contain the final result.