Data Structure and Application (1333203) - Winter 2024 Solution

Milav Dabgar

December 07, 2024

Question 1(a) [3 marks]

Write names of linear data structures.

Solution

Table 1. Linear Data Structures

Linear Data Structures
1. Array
2. Stack
3. Queue
4. Linked List

Mnemonic

"All Students Queue Lazily"

Question 1(b) [4 marks]

Define Time and space complexity.

Solution

Table 2. Complexity Definitions

Complexity Type	Definition	Notation
Time Complexity	Measures how execution time increases as input size grows	O(n), O(1), O(log n)
Space Complexity	Measures how memory usage increases as input size grows	O(n), O(1), O(log n)

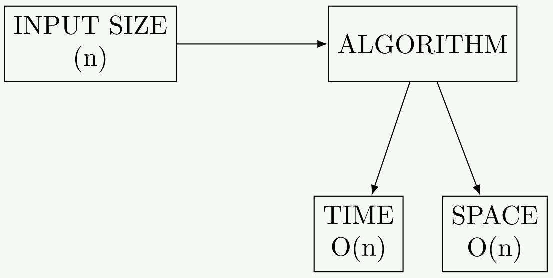


Figure 1. Complexity Analysis

Mnemonic

"Time Steps, Space Stores"

Question 1(c) [7 marks]

Explain concept of class & object with example.

Solution

Class Diagram

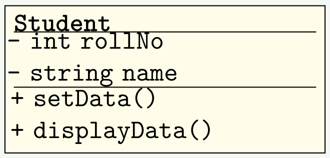


Figure 2. Student Class Structure

Table 3. Class vs Object

Concept	Definition	Example
Class	Blueprint or template for creating objects	Student class with properties (rollNo, name) and methods (setData, displayData)
Object	Instance of a class with specific values	student1 (rollNo=101, name="Raj")

Code Example:

```python
class Student:
    def __init__(self):
        self.rollNo = 0
        self.name = ""

    def setData(self, r, n):
        self.rollNo = r
        self.name = n

    def displayData(self):
        print(self.rollNo, self.name)

# Creating objects
student1 = Student()
student1.setData(101, "Raj")
```

Mnemonic

"Class Creates, Objects Operate"

Question 1(c) OR [7 marks]

Develop a class for managing student records with instance methods for adding and removing students from a class.

Solution

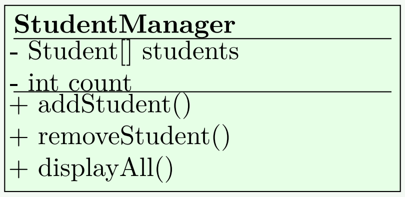


Figure 3. StudentManager Class

Code:

```python
class StudentManager:
    def __init__(self):
        self.students = []

    def addStudent(self, roll, name):
        student = Student()
        student.setData(roll, name)
        self.students.append(student)

    def removeStudent(self, roll):
        for i in range(len(self.students)):
            if self.students[i].rollNo == roll:
                self.students.pop(i)
                break

    def displayAll(self):
        for student in self.students:
            student.displayData()
```

Mnemonic

"Add Accumulates, Remove Reduces"

Question 2(a) [3 marks]

Explain the importance of constructor in class.

Solution

Table 4. Constructor Importance

Constructor Importance
1. Initializes object's data members
2. Automatically called when object is created
3. Can have different versions (default, parameterized, copy)

Mnemonic

"Initialization Always Creates"

Question 2(b) [4 marks]

Explain different operations on stack.

Solution

Table 5. Stack Operations

Operation	Description	Example
Push	Adds element to top	push(5)
Pop	Removes element from top	x = pop()
Peek/Top	Views top element without removing	x = peek()
isEmpty	Checks if stack is empty	if(isEmpty())

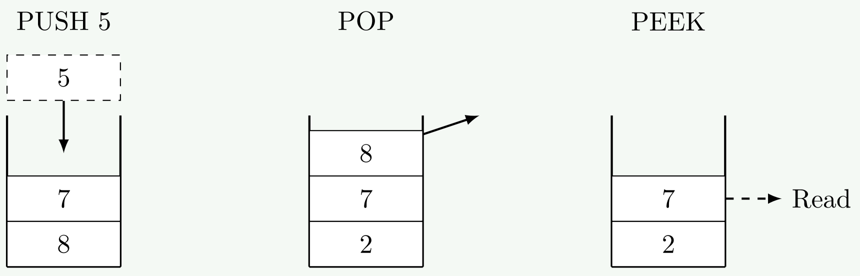


Figure 4. Stack Operations

Mnemonic

"Push Pop Peek Properly"

Question 2(c) [7 marks]

Describe evaluation algorithm of postfix expression A B C + * D /

Solution

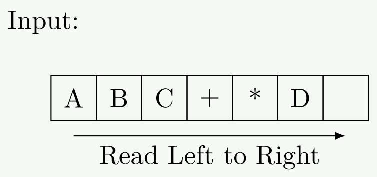


Figure 5. Postfix Evaluation Process

Table 6. Detailed Step-by-Step Trace

Step	Symbol	Action	Stack
1	A	Push onto stack	A
2	B	Push onto stack	A, B
3	C	Push onto stack	A, B, C
4	+	Pop B, C; Push B+C	A, (B+C)
5	*	Pop A, (B+C); Push A*(B+C)	A*(B+C)
6	D	Push onto stack	A*(B+C), D
7	/	Pop A*(B+C), D; Push result	(A*(B+C))/D

Mnemonic

"Read, Push, Pop, Calculate"

Question 2(a) OR [3 marks]

Write difference between stack and queue.

Solution

Table 7. Stack vs Queue

Feature	**Stack**	**Queue**
Principle	LIFO (Last In First Out)	FIFO (First In First Out)
Operations	Push/Pop	Enqueue/Dequeue
Access Points	Single end (top)	Two ends (front, rear)

Mnemonic

"Stack LIFO, Queue FIFO"

Question 2(b) OR [4 marks]

Explain concept of circular queue.

Solution

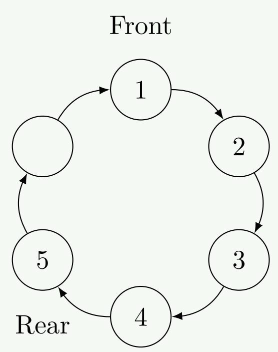


Figure 6. Circular Queue Concept

Table 8. Circular Queue Features

Feature	**Description**
Structure	Linear data structure with connected ends
Advantage	Efficiently uses memory by reusing empty spaces
Operations	Enqueue, Dequeue with modulo arithmetic

Mnemonic

"Circular Connects Front to Rear"

Question 2(c) OR [7 marks]

Describe the procedure for inserting a new node after and before a given node in a singly linked list.

Solution

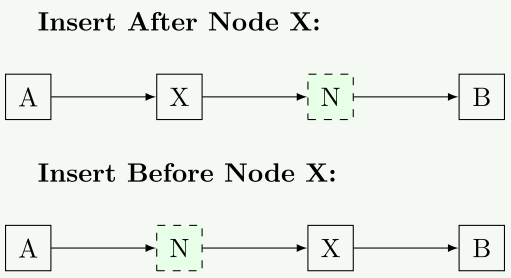


Figure 7. Insertion in Singly Linked List

Table 9. Insertion Procedure

Insertion	**Steps**
After Node X	1. Create new node N 2. Set N's next to X's next 3. Set X's next to N
Before Node X	1. Create new node N 2. Find node A pointing to X 3. Set N's next to X 4. Set A's next to N

Question 3(a) [3 marks]

Explain traversing a linked list.

Solution

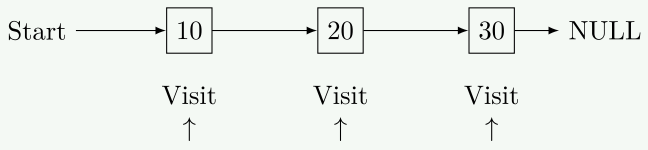


Figure 8. Linked List Traversal

Table 10. Traversal Steps

Step	**Action**
1	Initialize pointer to head
2	Access data at current node
3	Move pointer to next node
4	Repeat until NULL

Mnemonic

"Start, Access, Move, Repeat"

Question 3(b) [4 marks]

Explain expression conversion from infix to postfix.

Solution

Example Conversion

Infix: $A + B * C$
Postfix: $A\ B\ C\ *\ +$

Table 11. Conversion Algorithm Trace

Step	Action	Stack	Output
1	Scan from left to right		
2	If operand, add to output		A
3	If operator, push if higher precedence	+	A
4	Pop lower precedence operators	+	A B
5	Push current operator	*	A B
6	Continue until expression ends	*	A B C
7	Pop remaining operators		A B C * +

Mnemonic

"Operators Push Pop, Operands Output Directly"

Question 3(c) [7 marks]

Write a program to delete a node at the beginning and end of singly linked list.

Solution

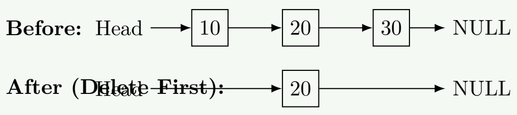


Figure 9. Deletion Visualization

Code:

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def deleteFirst(self):
        if self.head is None:
            return
        self.head = self.head.next

    def deleteLast(self):
        if self.head is None:
            return

        # If only one node
        if self.head.next is None:
            self.head = None
            return

        temp = self.head
        while temp.next.next:
            temp = temp.next

        temp.next = None
```

Mnemonic

"Delete First: Shift Head, Delete Last: Find Second-Last"

Question 3(a) OR [3 marks]

Explain searching an element in linked list.

Solution

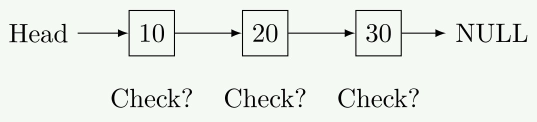


Figure 10. Linear Search in Linked List

Table 12. Search Steps

Step	Description
1	Start from head node
2	Compare current node's data with key
3	If match found, return true
4	Else, move to next node and repeat

Mnemonic

"Start, Compare, Move, Repeat"

Question 3(b) OR [4 marks]

Explain concepts of circular linked lists.

Solution

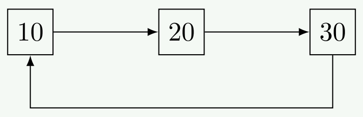


Figure 11. Circular Linked List

Table 13. Circular LL Features

Feature	Description
Structure	Last node points to first node
Advantage	No NULL pointers, efficient for circular operations
Traversal	Need extra condition to prevent infinite loop

Mnemonic

"Last Links to First"

Question 3(c) OR [7 marks]

Explain algorithm to search a particular element from list using Binary Search.

Solution

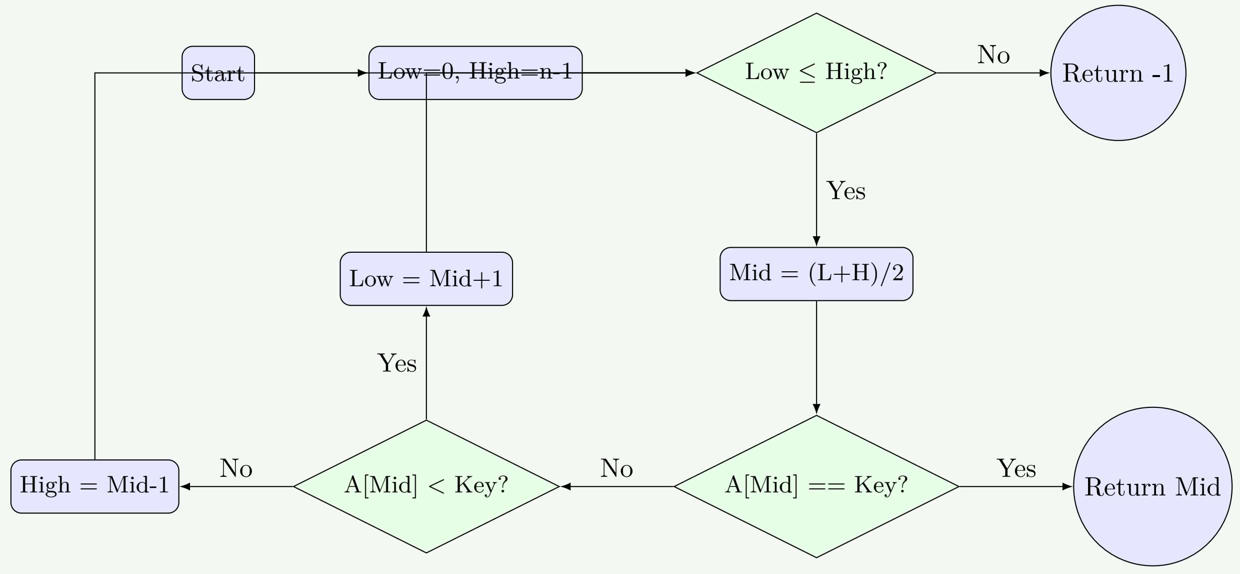


Figure 12. Binary Search Flowchart

Code:

```python
def binarySearch(arr, key):
    low = 0
    high = len(arr) - 1

    while low <= high:
        mid = (low + high) // 2

        if arr[mid] == key:
            return mid
        elif arr[mid] < key:
            low = mid + 1
        else:
            high = mid - 1

    return -1
```

Mnemonic

"Middle, Compare, Eliminate Half"

Question 4(a) [3 marks]

Write applications of linked list.

Solution

Table 14. Applications of Linked List

Applications
1. Implementation of stacks and queues
2. Dynamic memory allocation
3. Image viewer (next/previous images)

Mnemonic

"Store Data Dynamically"

Question 4(b) [4 marks]

Differentiate between singly linked list and doubly linked list.

Solution

Table 15. Singly vs Doubly Linked List

Feature	**Singly Linked List**	**Doubly Linked List**
Node Structure	One pointer (next)	Two pointers (next, prev)
Traversal	Forward only	Both directions
Memory	Less memory	More memory
Operations	Simple, less code	Complex, more flexible

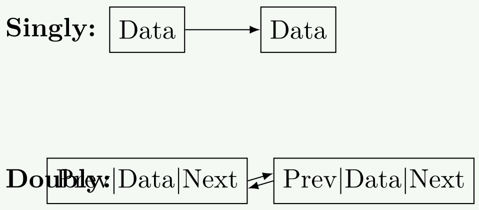


Figure 13. Node Structures

Mnemonic

"Single Direction, Double Direction"

Question 4(c) [7 marks]

Write a program to sort numbers in ascending order using selection sort algorithm.

Solution

Initial:	5	3	8	1	2
Pass 1 (Swap 5,1):	1	3	8	5	2
Pass 2 (Swap 3,2):	1	2	8	5	3
Pass 3 (Swap 8,3):	1	2	3	5	8

Figure 14. Selection Sort Calculation

Code:

```python
def selectionSort(arr):
    n = len(arr)

    for i in range(n):
        min_idx = i

        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j

        # Swap the found minimum element with the first element
        arr[i], arr[min_idx] = arr[min_idx], arr[i]

    return arr

# Example usage
arr = [5, 3, 8, 1, 2]
sorted_arr = selectionSort(arr)
print(sorted_arr)  # Output: [1, 2, 3, 5, 8]
```

Mnemonic

"Find Minimum, Swap Position"

Question 4(a) OR [3 marks]

Explain bubble sort algorithm.

Solution

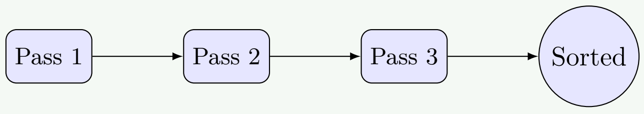


Figure 15. Bubble Sort Flow

Table 16. Key Points

Key Points
Compare adjacent elements
Swap if they are in wrong order
Largest element bubbles to end in each pass

Mnemonic

"Bubble Bigger Elements Upward"

Question 4(b) OR [4 marks]

Differentiate Linear & Binary search.

Solution

Table 17. Linear vs Binary Search

Feature	**Linear Search**	**Binary Search**
Working Principle	Sequential checking	Divide and conquer
Time Complexity	O(n)	O(log n)
Data Arrangement	Unsorted or sorted	Must be sorted
Best For	Small datasets	Large datasets

Mnemonic

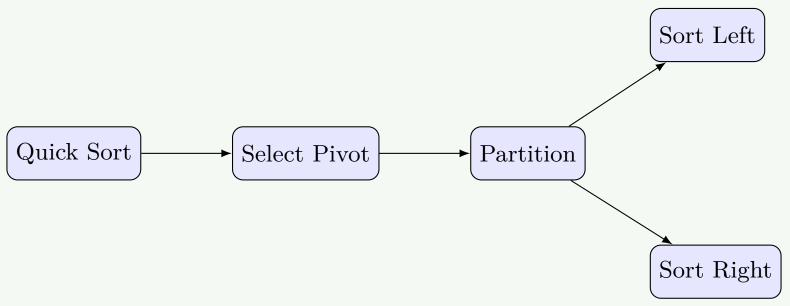
"Linear Looks at All, Binary Breaks in Half"

Question 4(c) OR [7 marks]

Explain Quick sort & Merge sort algorithm.

Solution

Quick Sort:



Merge Sort:

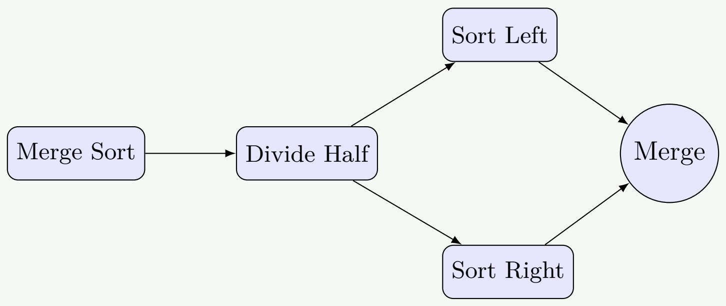


Table 18. Complexity Comparison

Algorithm	Principle	Avg Time	Space
Quick Sort	Partitioning around pivot	O(n log n)	O(log n)
Merge Sort	Divide, conquer, combine	O(n log n)	O(n)

Question 5(a) [3 marks]

Define a complete binary tree.

Solution

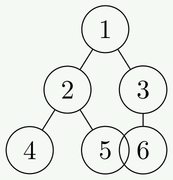


Figure 16. Complete Binary Tree

Table 19. Properties

Property	Description
All levels filled	Except possibly the last level
Last level filled from left	Nodes added from left to right

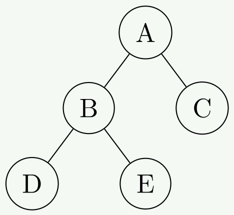
Mnemonic

"Fill Left to Right, Level by Level"

Question 5(b) [4 marks]

Explain inorder traversal of a binary tree.

Solution



Inorder: $D \rightarrow B \rightarrow E \rightarrow A \rightarrow C$

Figure 17. Inorder Traversal

Table 20. Algorithm Steps

Step	Action
1	Traverse left subtree
2	Visit root node
3	Traverse right subtree

Code:

```python
def inorderTraversal(root):
    if root:
        inorderTraversal(root.left)
        print(root.data, end=" -> ")
        inorderTraversal(root.right)
```

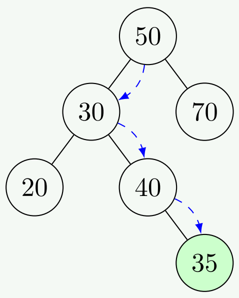
Mnemonic

"Left, Root, Right"

Question 5(c) [7 marks]

Write a program to inserting a node into a binary search tree.

Solution



Insert 35:
1. 35 < 50 (Left)
2. 35 > 30 (Right)
3. 35 < 40 (Left)

Figure 18. Insertion Process

Code:

```python
class Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

def insert(root, key):
    if root is None:
        return Node(key)

    if key < root.key:
        root.left = insert(root.left, key)
    else:
        root.right = insert(root.right, key)

    return root
```

Mnemonic

"Compare, Move, Insert"

Question 5(a) OR [3 marks]

State the fundamental characteristic of a binary search tree.

Solution

Table 21. BST Characteristics

Characteristics
1. Left child nodes < Parent node
2. Right child nodes > Parent node
3. No duplicate values allowed

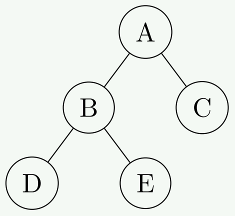
Mnemonic

"Left Less, Right More"

Question 5(b) OR [4 marks]

Explain postorder traversal of a binary tree.

Solution



Postorder: $D \rightarrow E \rightarrow B \rightarrow C \rightarrow A$

Figure 19. Postorder Traversal

Table 22. Step-by-Step

Step	**Action**
1	Traverse left subtree
2	Traverse right subtree
3	Visit root node

Code:

```python
def postorderTraversal(root):
    if root:
        postorderTraversal(root.left)
        postorderTraversal(root.right)
        print(root.data, end=" -> ")
```

Mnemonic

"Left, Right, Root"

Question 5(c) OR [7 marks]

Write a program to delete a node from a binary search tree.

Solution

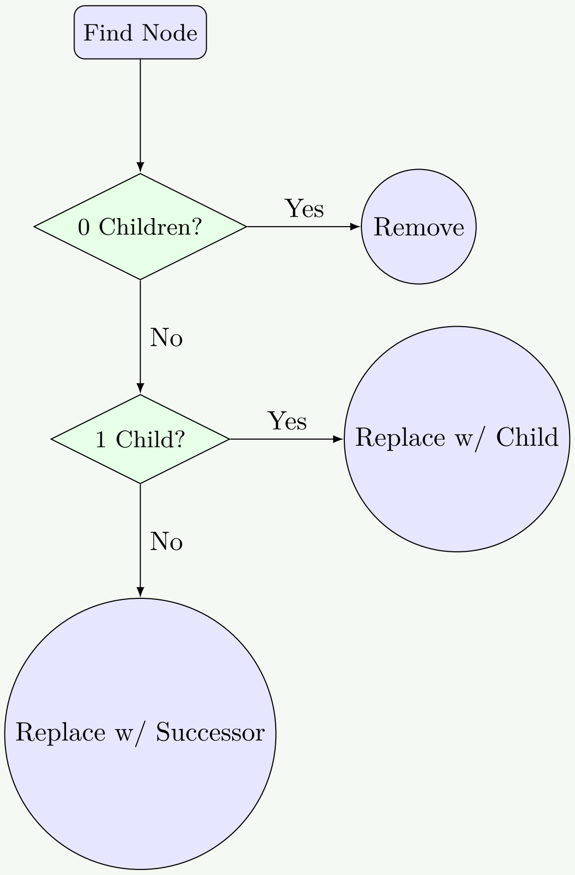


Figure 20. Deletion Logic

Code:

```python
def minValueNode(node):
    current = node
    while current.left is not None:
        current = current.left
    return current

def deleteNode(root, key):
    if root is None: return root

    if key < root.key:
        root.left = deleteNode(root.left, key)
    elif key > root.key:
        root.right = deleteNode(root.right, key)
    else:
        # Node with only one child or no child
        if root.left is None:
            return root.right
        elif root.right is None:
            return root.left

        # Node with two children
        temp = minValueNode(root.right)
```

```
            root.key = temp.key
            root.right = deleteNode(root.right, temp.key)

    return root
```

Mnemonic

"Zero: Remove, One: Replace, Two: Successor"