Data Structures

1. Array

- Merits:

- Simple to implement

- Constant time access to elements

- Demerits:

- Fixed size

- Insertion and deletion are inefficient

- Operations:

- Access: O(1)

- Insertion: O(n)

- Deletion: O(n)

Example:

```python
# Creating an array in Python
array = [1, 2, 3, 4, 5]

# Accessing elements
print(array[0])  # Output: 1
```

Inserting element

array.append(6)

Deleting element

array.remove(3)

2.Linked List

Merits:

- Dynamic size

- Efficient insertion and deletion

Demerits:

- Sequential access is slow

- Extra memory for pointers

Operations:

- Access: O(n)

- Insertion: O(1)

- Deletion: O(1)

Example:

```python

# Creating a linked list node
```

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

# Creating a linked list
class LinkedList:
    def __init__(self):
        self.head = None

# Inserting at the beginning
def insert(self, data):
    new_node = Node(data)
    new_node.next = self.head
    self.head = new_node

# Deleting a node
def delete(self, key):
    temp = self.head
    if temp is not None:
        if temp.data == key:
            self.head = temp.next
            temp = None
            return
    while temp is not None:
```

```
            if temp.data == key:
                break
            prev = temp
            temp = temp.next

        if temp == None:
            return

        prev.next = temp.next
        temp = None
```

3.Stack

Merits:

- LIFO (Last In First Out) structure
- Simple implementation

Demerits:

- Limited functionality
- Can lead to stack overflow if too many elements are pushed

Operations:

- Push: O(1)
- Pop: O(1)
- Peek: O(1)

Example:

```python
# Creating a stack using Python list
stack = []

# Pushing elements
stack.append(1)
stack.append(2)

# Popping element
top_element = stack.pop()
print(top_element)    # Output: 2
```

4.Queue

Merits:

- FIFO (First In First Out) structure
- Useful in implementing scheduling algorithms

Demerits:

- Can lead to queue overflow if too many elements are enqueued
- Inefficient dequeue operation in some implementations

Operations:

- Enqueue: O(1)

- Dequeue: O(1)

- Peek: O(1)

Example:

```python

# Creating a queue using Python list

queue = []

# Enqueuing elements

queue.append(1)

queue.append(2)

# Dequeuing element

front_element = queue.pop(0)

print(front_element)     # Output: 1
```