SESSION 2 REPORT

Possible implementations

1. Binary Heap

- **Description**:

A binary heap is a tree-based data structure where each parent node is greater than (max-heap) its children. It is commonly used to implement priority queues.

- **Advantages :**
  - ☐ Time complexity for insertions and deletions is O(log n).
  - ☐ Efficient insertion and deletion of elements based on priority.
  - ☐ Can dynamically handle varying numbers of posts.
  - ☐ Space-efficient and works well for large datasets.
- **Disadvantages :**
  - ☐ Complex to implement compared to simpler structures.

2. Array

- **Description :**

Array is a linear data structure that is a collection of data elements of the same types. Arrays are stored in contiguous memory locations. It is a static data structure with a fixed size.

- **Advantages :**
  - ☐ **Efficient and Fast Access**: Arrays allow direct and efficient access to any element in the collection with constant access time, as the data is stored in contiguous memory locations.
  - ☐ **Memory Efficiency**: Arrays store elements in contiguous memory, allowing efficient allocation in a single block and reducing memory fragmentation.
  - ☐ **Compatibility with hardware**: The array data structure is compatible with most hardware architectures.
  - ☐ **Simple to implement.**
  - ☐ **Retrieval of the top priority element is O(1).**
- **Disadvantages :**

- ☐ **Fixed Size**: Arrays have a fixed size set at creation. Expanding an array requires creating a new one and copying elements, which is time-consuming and memory-intensive.
- ☐ **Memory Allocation Issues**: Allocating large arrays can cause memory exhaustion, leading to crashes, especially on systems with limited resources.
- ☐ **Insertion and Deletion**: Adding or removing elements requires shifting subsequent elements, making these operations inefficient.
- ☐ **Limited Data Type Support**: Arrays support only elements of the same type, limiting their use with complex data types.
- ☐ **Lack of Flexibility**: Fixed size and limited type support make arrays less adaptable than structures like linked lists or trees.

3. Linked list

- **Description:**

A Linked list is a dynamic arrangement that contains a “link” to the structure containing the subsequent items. It’s a set of structures ordered not by their physical placement in memory (like an array) but by logical links that are stored as a part of the information within the structure itself.

- **Advantages:**
  - ☐ Easy to implement.
  - ☐ Efficient insertion and deletion at any position
  - ☐ Good for dynamic memory allocation.
  - ☐ Flexible.
  - ☐ Efficient for large data.
  - ☐ Scalability.
- **Disadvantages :**
  - ☐ **Memory usage**: More memory is required in the linked list as compared to an array.
  - ☐ **Random Access**: Random access is not possible in a linked list due to its dynamic memory allocation.

- ☐ **Lower efficiency at times**: For certain operations, such as searching for an element or iterating through the list, can be slower in a linked list.
- ☐ **Complex implementation**: The linked list implementation is more complex when compared to array. It requires a complex programming understanding.
- ☐ **Difficult to share data**: This is because it is not possible to directly access the memory address of an element in a linked list.
- ☐ **Not suited for small dataset**: Cannot provide any significant benefits on small dataset compared to that of an array.

4. Hash table

A hash table is a data structure that you can use to store data in key-value format with direct access to its items in constant time.

- **Advantages:**
  - ☐ Flexibility.
  - ☐ Fast retrieval, insertion, and deletion.
  - ☐ Good for large datasets.
- **Disadvantages**
  - ☐ Hash tables have a limited capacity and will eventually fill up.
  - ☐ Hash tables can be complex to implement.
  - ☐ Hash tables do not maintain the order of elements, which makes it difficult to retrieve elements in a specific order.

Complexity Analysis

We decided to base the complexity analysis on the parameter ***input size (n)***, because it is the most critical factor influencing the performance of the algorithm. Input size represents the number of elements the algorithm processes, and it directly affects the time and space complexity.