1.List the properties of B tree.

Balanced Structure: B-trees are balanced, meaning that all leaf nodes of the tree are at the same level. This balance is maintained through a series of operations like splitting and merging nodes during insertions and deletions.

Degree: A B-tree of degree 't' is a tree in which each node can have a maximum of 2t - 1 keys and a minimum of 't - 1' keys. The number of children for a node is always one more than the number of keys in the node.

Sorted Data: The keys in a B-tree are stored in a sorted order within each node. This allows for efficient searching using binary search.

2.write the advantages of using heap data structure.

Heap Sort: Heap sort is an in-place sorting algorithm based on binary heaps. It has a time complexity of O(n log n), which is efficient and often outperforms other sorting algorithms, such as bubble sort or insertion sort.

Memory Efficiency: Heaps are often implemented as arrays, which are memory-efficient compared to other data structures with pointers or additional metadata.

Partial Sorting: Heaps can be used for finding the top k elements in a list, where you don't need to sort the entire list. This is advantageous when you need to find the largest or smallest values in a dataset.

3.Differentiate heap tree with B tree.

heap tree	**B tree**
A heap is primarily used to maintain a priority queue, where elements are ordered based on their priority.	B-trees are designed to organise and manage data for efficient insertion, deletion, and retrieval.
It is commonly used in algorithms like heap sort and for tasks that require efficient retrieval of extreme values.	They are often used in database systems and file systems for indexing and data storage.
Heap trees are typically binary trees with specific ordering properties.	B-trees are multiway search trees. Each node in a B-tree can have multiple children, and they are structured to keep the tree balanced.
In a max-heap, each parent node has a value greater than or equal to the values of its children, and in a min-heap, each parent node has a value less than or equal to the values of its children.	B-trees are characterised by a minimum degree 't' and maintain a sorted order of keys within nodes.

4.identify the benefit of B+ tree over B tree.

Sequential Disk Access: B+ trees are designed to facilitate sequential disk access, which is highly beneficial for reading and scanning large datasets. Because the leaf nodes are linked together, traversing the entire dataset sequentially is more efficient in a B+ tree.

Minimises Disk I/O: B+ trees tend to minimise disk I/O operations due to their balanced structure and efficient range queries. This is especially advantageous in scenarios where

disk I/O is a significant performance bottleneck, such as in database systems and file systems.
Non-Leaf Nodes Store Keys: In a B+ tree, non-leaf nodes store keys rather than data, which helps reduce the size of the tree and ensures that the data storage nodes (leaf nodes) are more compact. This can lead to better cache performance.

5.recall Heapify operations in a heap data structure
Down-Heapify: Down-heapify is used when a node violates the heap property. It ensures that the element at the top of the heap (root) is the maximum (in a max-heap) or minimum (in a min-heap) element, and the rest of the elements maintain the heap property. The down-heapify operation is typically performed after removing the root element.
Up-Heapify: Up-heapify is used when a new element is inserted into the heap. It ensures that the newly added element maintains the heap property by comparing it with its parent and potentially swapping elements.

Part-B
3.Construct a B+ Tree of order 2 by inserting the given numbers.
Constructing a B+ tree of order 2 by inserting the given numbers (10, 20, 5, 6, 12, 30, 7, 17, 3, and 19) involves the following steps:

Start with an empty B+ tree of order 2.

Insert the first number, 10. The tree now has only one element.
markdown

```
    10
```

Insert the second number, 20. The tree can still accommodate it without splitting.
markdown

```
    10, 20
```

Insert the third number, 5. Insert it into the first node (since it's less than 10).
markdown

```
    10
    / \
   5   20
```

Insert the fourth number, 6. The first node is full, so split it into two nodes.
markdown

```
    10
    / \
   5   6
```

Insert the fifth number, 12. The first node can still accommodate it.
markdown

```
    10
    / \
   5   6, 12
```

Insert the sixth number, 30. Insert it into the second node (since it's greater than 10).
markdown

```
      10   20
      / \    \
     5   6, 12   30
```

Insert the seventh number, 7. Insert it into the first node.
markdown

```
      10   20
      / \    \
     5   6, 7   12, 30
```

Insert the eighth number, 17. Insert it into the second node.
markdown

```
      10   20
      / \    \
     5   6, 7   12, 17, 30
```

Insert the ninth number, 3. Insert it into the first node.
markdown

```
      10   20
      / \    \
    3, 5   6, 7   12, 17, 30
```

Finally, insert the tenth number, 19. Insert it into the second node.
markdown

```
      10   20
      / \    \
    3, 5   6, 7   12, 17, 19, 30
```

This is the final B+ tree after inserting all the given numbers.
In a B+ tree of order 2, each node can hold a maximum of 2 elements before it is split into two nodes.