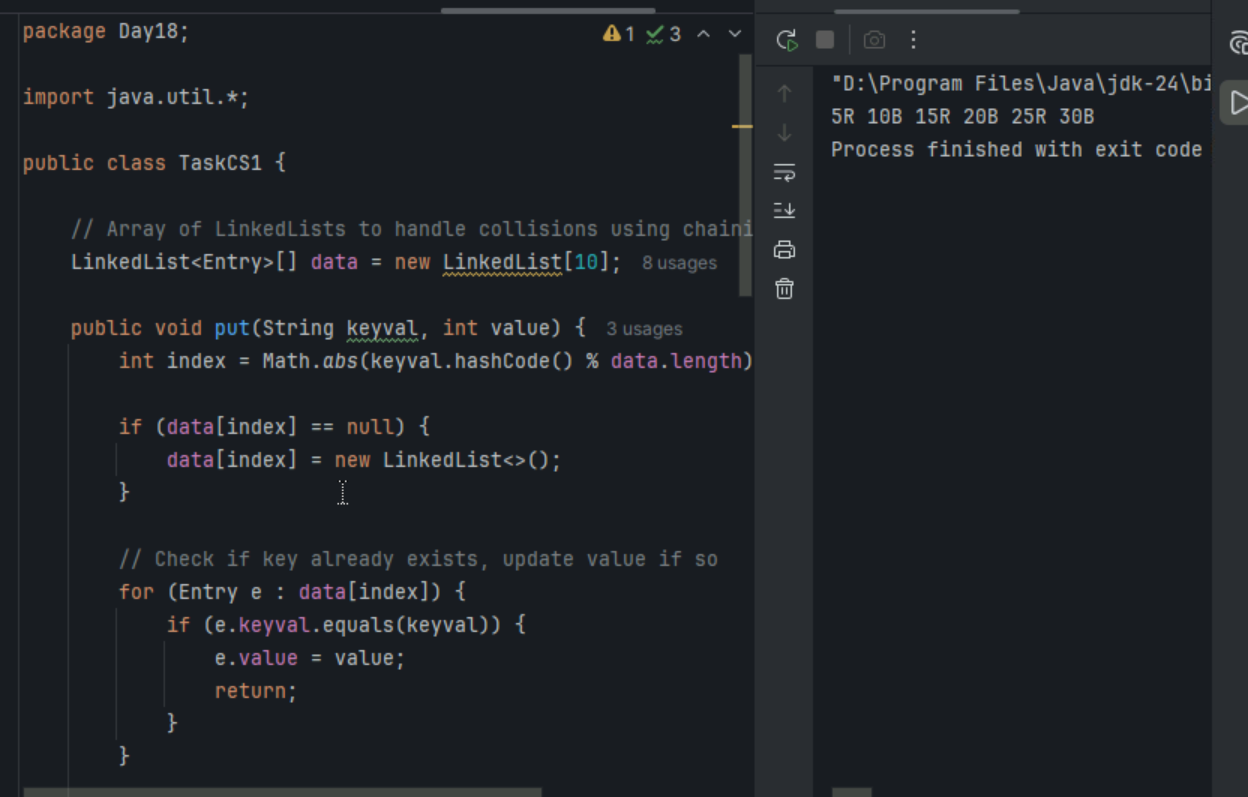
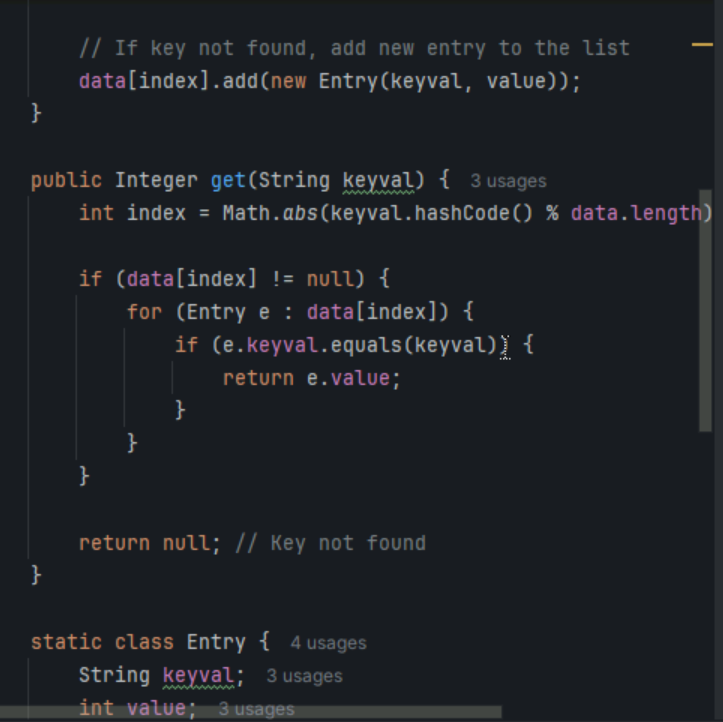
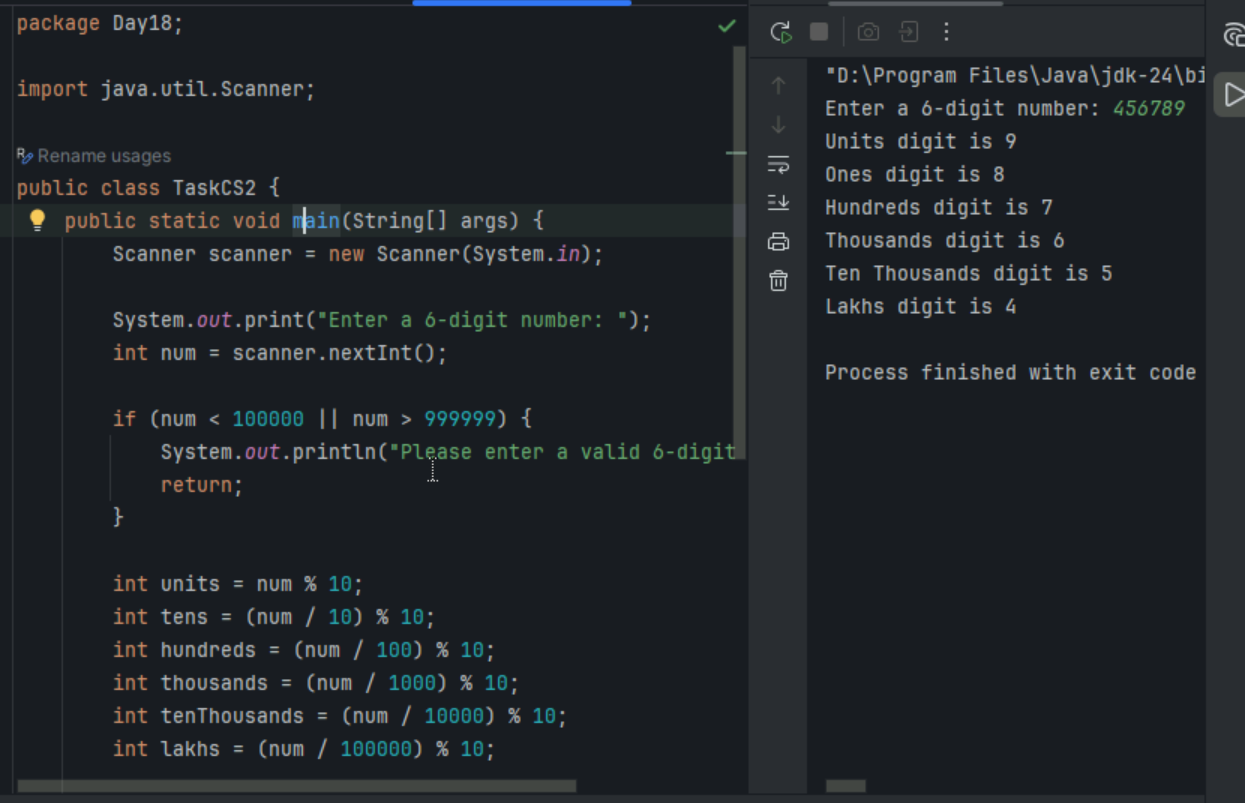
Task1

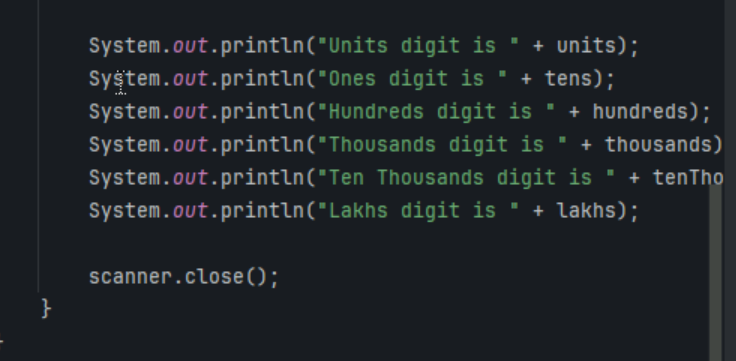




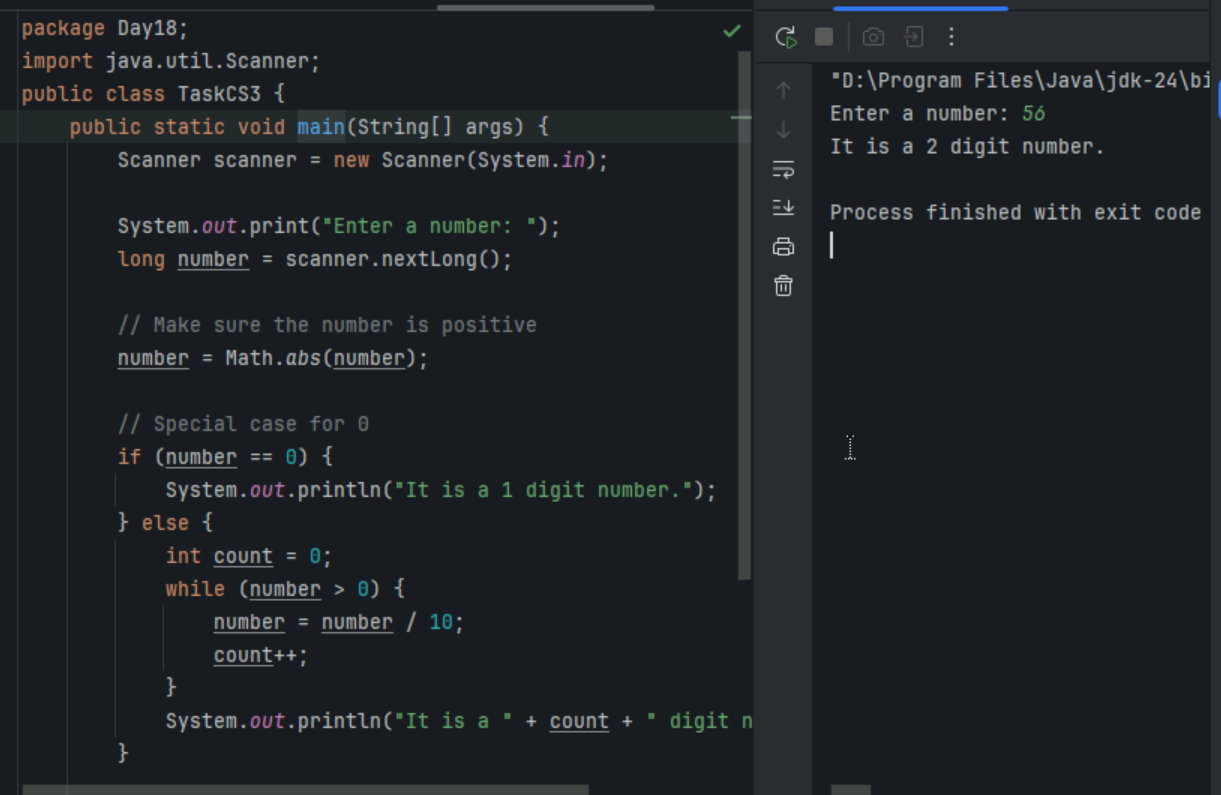


Task2





Task3



Task 4 - What are the applications of heap sort?

Heap sort is a versatile sorting algorithm with several practical applications. Here are some key areas where heap sort and the heap data structure are particularly useful:

1. **Priority Queues**: Heaps are commonly used to implement priority queues, where elements with higher priority are processed first. This is useful in scheduling tasks, handling interruptions, and processing events
2. **Graph Algorithms**: Heaps are used in algorithms like Prim's and Dijkstra's for finding the shortest path in graphs
3. **Data Compression**: Heaps are used in Huffman coding, a lossless data compression algorithm
4. **Order Statistics**: Heaps can efficiently find the kth smallest or largest element in an array
5. **Job Scheduling**: In job scheduling algorithms, heaps help manage tasks based on priority or deadlines
6. **Load Balancing**: Heaps are used to distribute tasks or requests to servers by processing elements with the lowest load first
7. **Resource Allocation**: Heaps can efficiently allocate resources like memory blocks or CPU time by assigning a priority to each resource
8. **Sorting Large Datasets**: Heap sort is efficient for sorting large datasets due to its consistent time complexity of (O(n \log n))

Task 5 - Do you find any significance change between the breadthFirstSearchRecursive() approach compared to the standard BFS?

1. Will it need for queues entirely by using a stack-based recursion?
2. Will it simplifies implementation by using queues implicitly within recursive function calls?
3. will it achieve same result but emphasizes on recursive style using the same level-order logic with explicit queue management?

Task 6 - How does heap sort work ? explain the technique in 5 .. algorithm

Task - 7 how can you say recursive functions maintain the state of each call during execution?

1. Each recursive call creates a new thread, and context switching maintains state.

2. Recursive functions store state in global variables accessible across calls.

3. The system call stack tracks local variables and return addresses for each recursive invocation.

4. Recursive functions replicate the heap structure to keep values between calls.

Task 8 - Which property of a priority queue differentiates it most from a regular queue implementation?

1. It allows insertion and removal only from one end, similar to a stack.

2. Elements are removed based on their order of insertion rather than priority.

3. Elements are dequeued based on their priority, not their insertion order, often implemented using a binary heap.

4. It maintains a strict hierarchical structure using a self-balancing BST to enforce priority.

Task 9 -What is the main purpose of using a binary heap in the implementation of a priority queue?

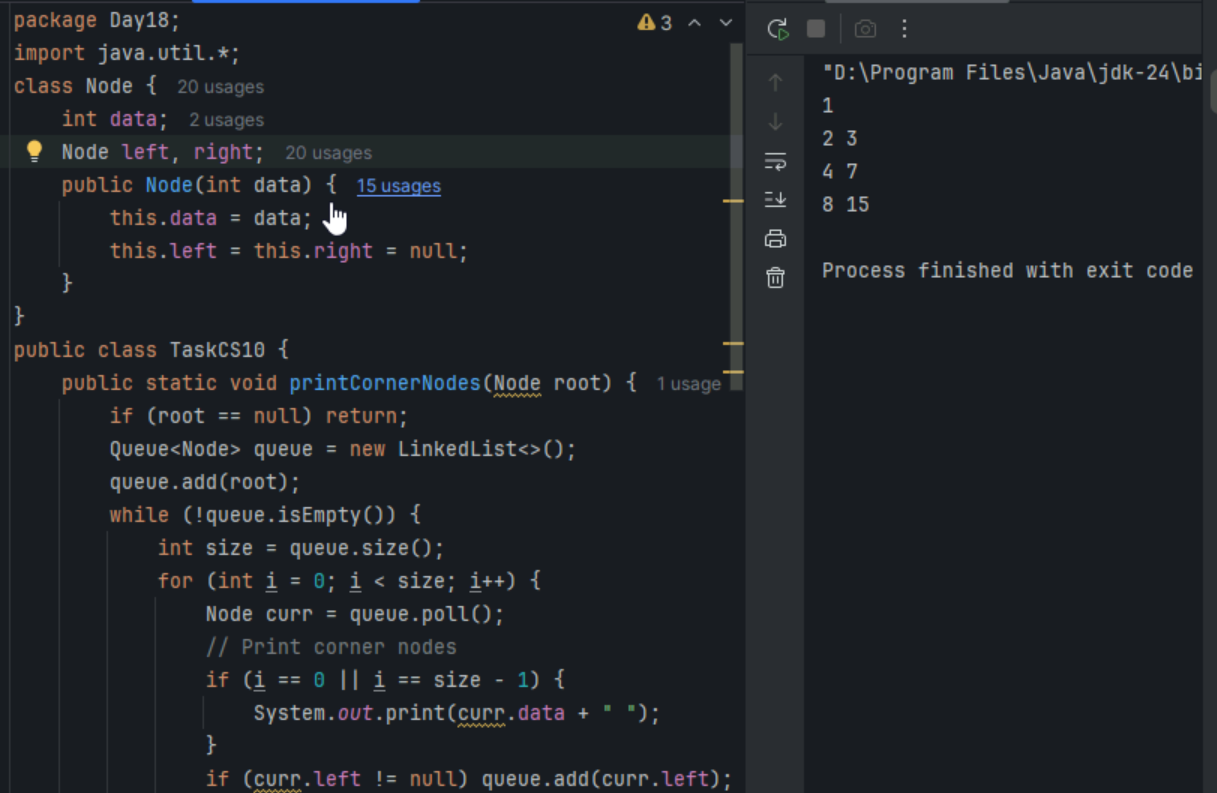
1. To maintain keys in alphabetical order for efficient string processing.

2. To ensure that the highest-priority element always bubbles to the root efficiently.

3. To guarantee constant-time insertion and logarithmic-time deletion.

4. To reduce memory consumption by flattening the tree into a linear array.

Task10





Task12 - How does this binary search function behave on unsorted arrays?

public class BinarySearch {

public int search(int[] arr, int target) {

int left = 0, right = arr.length - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == target) {

return mid;

} else if (arr[mid] < target) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return -1;

}

}

1. It works regardless of sorting

2. It throws exception if unsorted

3. It may return incorrect index

4. It sorts before searching

Task 13 - What is the result of performing DFS traversal in this graph implementation?

import java.util.\*;

public class DFSGraph {

Map<Integer, List<Integer>> adj = new HashMap<>();

Set<Integer> visited = new HashSet<>();

public void addEdge(int u, int v) {

adj.computeIfAbsent(u, x -> new ArrayList<>()).add(v);

}

public void dfs(int node) {

if (visited.contains(node)) {

return;

}

visited.add(node);

System.out.print(node + " ");

for (int neighbor : adj.getOrDefault(node, new ArrayList<>())) {

dfs(neighbor);

}

}

}

1. DFS uses a queue to ensure order

2. DFS will return shortest path like BFS

3. DFS traverses all nodes depth-first recursively

4. DFS skips connected nodes due to reentrancy issue

Task14 - Why is BFS generally preferred over DFS in shortest path algorithms for unweighted graphs?

1. BFS uses random access to edges, ensuring constant-time traversal.

2. BFS explores one path to maximum depth before switching, reducing memory usage.

3. BFS ignores revisiting nodes, reducing processing time in cyclic graphs.

4. BFS explores nodes in increasing distance order from the source, ensuring shortest paths are found first.

Task 15 - Write Algo for radix sort?

1. Find the maximum number in the array to determine the number of digits (d).
2. Loop from the least significant digit to the most significant digit (from units to the highest place):  
   * Use a stable sort (usually Counting Sort) to sort elements based on the current digit.
3. Repeat the sorting process for each digit position.

Task 16 - Write pseudo code for radix sort

function radixSort(array):

maxNum = find maximum number in array

place = 1

while maxNum / place > 0:

countingSortByDigit(array, place)

place \*= 10

Task17

