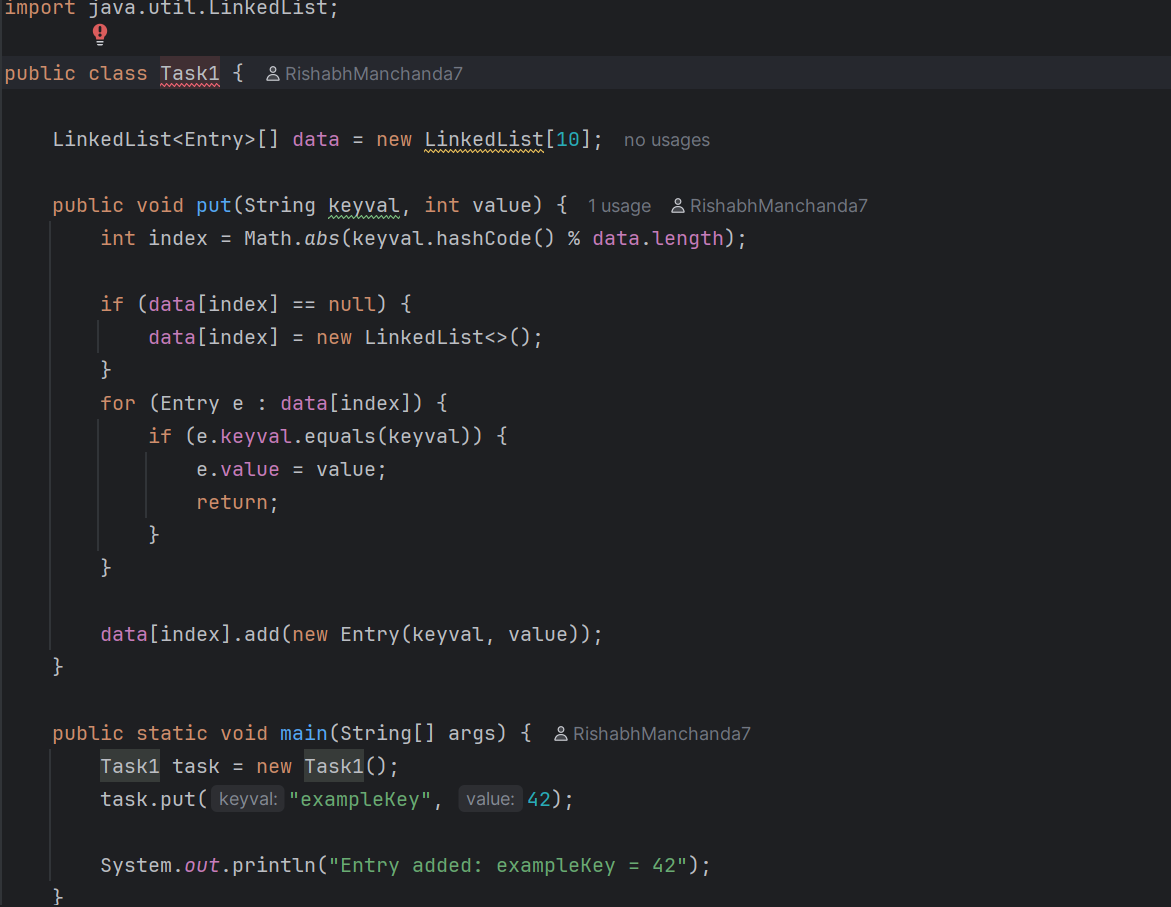
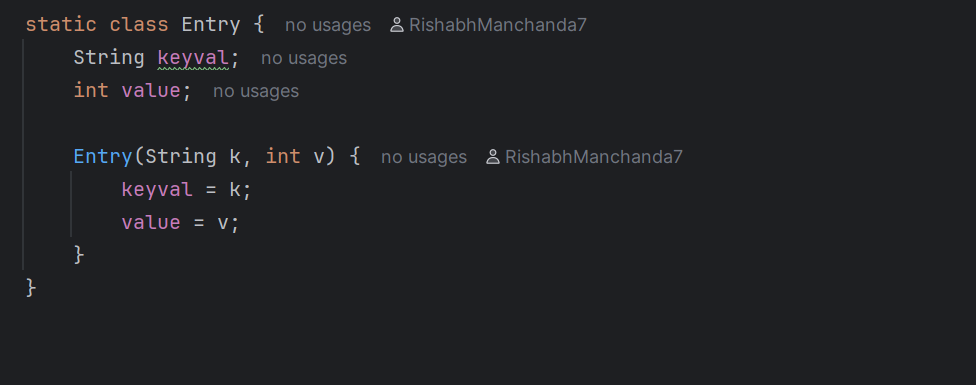
Day-18

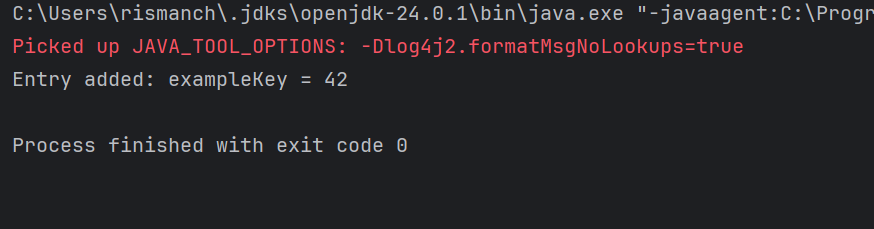
Name- Rishabh Manchanda  
  
Employee ID:

109883998

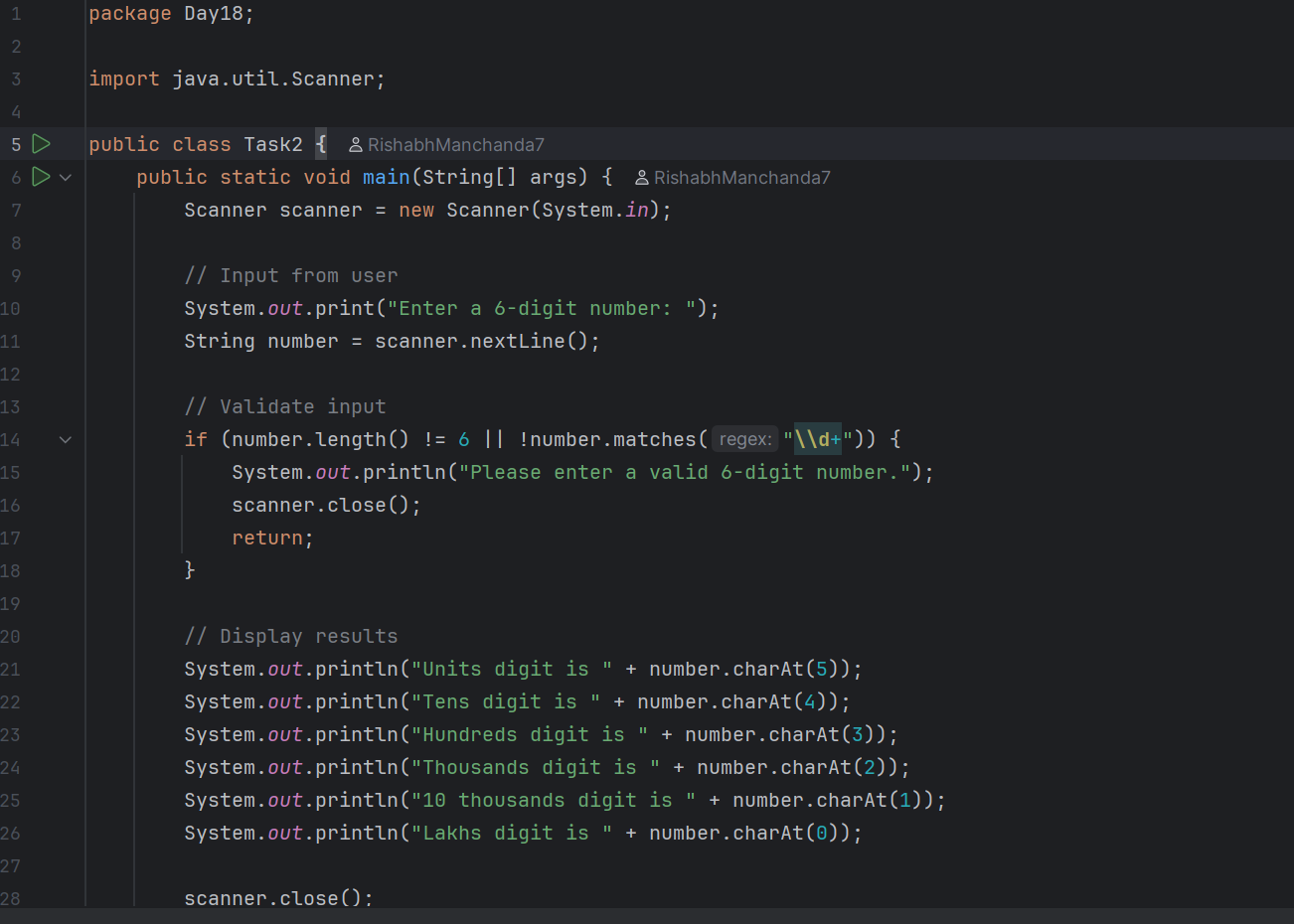
TASK- 1

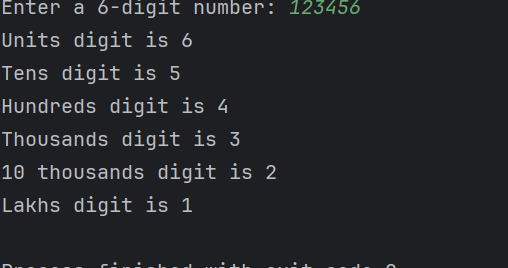






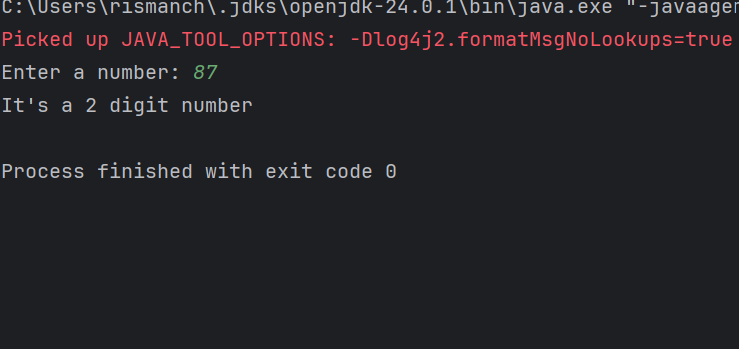
TASK-2





TASK-3



   
  
  
  
Task-4

What are the applications of heap sort?

Here are the main applications of Heap Sort:

1. Priority Queue Operations

* Efficiently managing priority queues
* Task scheduling in operating systems
* Process scheduling
* Event-driven simulation
* Dijkstra's shortest path algorithm

1. Memory Management

* Garbage collection
* Memory allocation/deallocation
* Managing system resources

1. Finding K-th Elements

* Finding k largest/smallest elements
* Median finding
* Statistical analysis

1. External Sorting

* Sorting large datasets that don't fit in memory
* Database operations
* File system organization

1. Graph Algorithms

* Prim's minimum spanning tree
* Network flow algorithms
* Best-first search

1. Gaming Applications

* AI decision making
* Game state evaluation
* Path finding algorithms

1. System Applications

* Job scheduling in operating systems
* Real-time systems
* Network packet prioritization

Task-6

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

1 BUILD MAX-HEAP

2 HEAPIFY

3 COMPLETE IMPLEMENTATION

4 COMPLETE IMPLEMENTATION

5 EXAMPLE VISUALIZATION

Key Characteristics:

* Time Complexity: O(n log n)
* Space Complexity: O(1)
* Not Stable
* In-place sorting

Task 15:

Write algo for radix sort

The key idea behind Radix Sort is to exploit the concept of place value. It assumes that sorting numbers digit by digit will eventually result in a fully sorted list. Radix Sort can be performed using different variations, such as Least Significant Digit (LSD) Radix Sort or Most Significant Digit (MSD) Radix Sort.

***Step 1:*** *Find the largest element in the array, which is 802. It has three digits, so we will iterate three times, once for each significant place.*

***Step 2:*** *Sort the elements based on the unit place digits (X=0). We use a stable sorting technique, such as counting sort, to sort the digits at each significant place. It's important to understand that the default implementation of counting sort is unstable i.e. same keys can be in a different order than the input array. To solve this problem, We can iterate the input array in reverse order to build the output array. This strategy helps us to keep the same keys in the same order as they appear in the input array.*

***Step 3:*** *Sort the elements based on the tens place digits.*

***Step 4:*** *Sort the elements based on the hundreds place digits and so on*

Task 16:

Write pseudo code for radix sort

function RadixSort(array):

# Find the maximum number to know number of digits

max = getMaxValue(array)

# Do counting sort for every digit

# exp is 10^i where i is current digit number

exp = 1

while max/exp > 0:

CountingSortByDigit(array, exp)

exp \*= 10

function CountingSortByDigit(array, exp):

n = length(array)

output = new array[n]

count = new array[10] initialized to 0

# Store count of occurrences in count[]

for i from 0 to n-1:

digit = (array[i]/exp) % 10

count[digit]++

# Change count[i] so that count[i] contains

# actual position of this digit in output[]

for i from 1 to 9:

count[i] += count[i-1]

# Build the output array

for i from n-1 down to 0:

digit = (array[i]/exp) % 10

output[count[digit]-1] = array[i]

count[digit]--

# Copy output array to input array

for i from 0 to n-1:

array[i] = output[i]

function getMaxValue(array):

max = array[0]

for i from 1 to length(array)-1:

if array[i] > max:

max = array[i]

return max

Task 05:

Do you find any significant change between the breadthFirstSearchRecursive() approach compared to the standard BFS?

1. Will it need queues entirely by using a stack-based recursion?

1. Will it simplify implementation by using queues implicitly within recursive function calls?

1. will it achieve the same result but emphasizes on recursive style using the same level-order logic with explicit queue management?

1. will it process nodes in post-order sequence to avoid memory allocation?

Task 07

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 08:

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 09

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

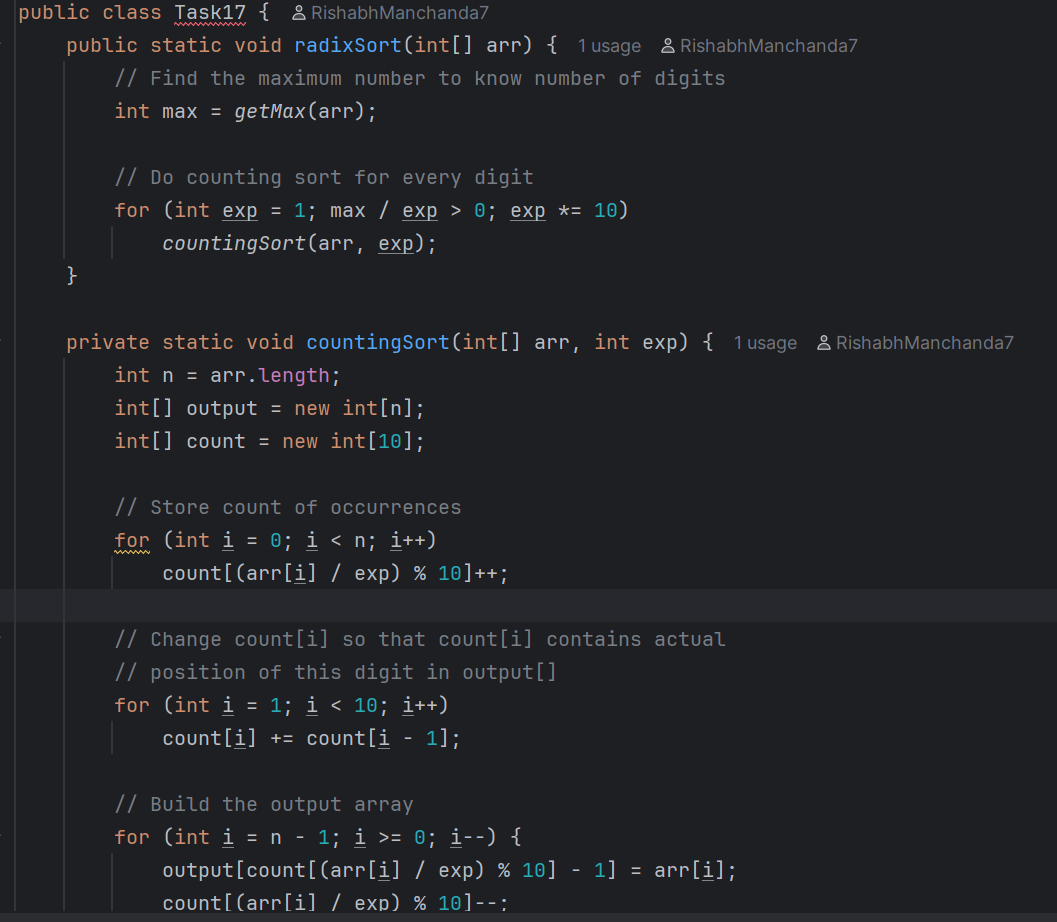
1. To maintain keys in alphabetical order for efficient string processing.

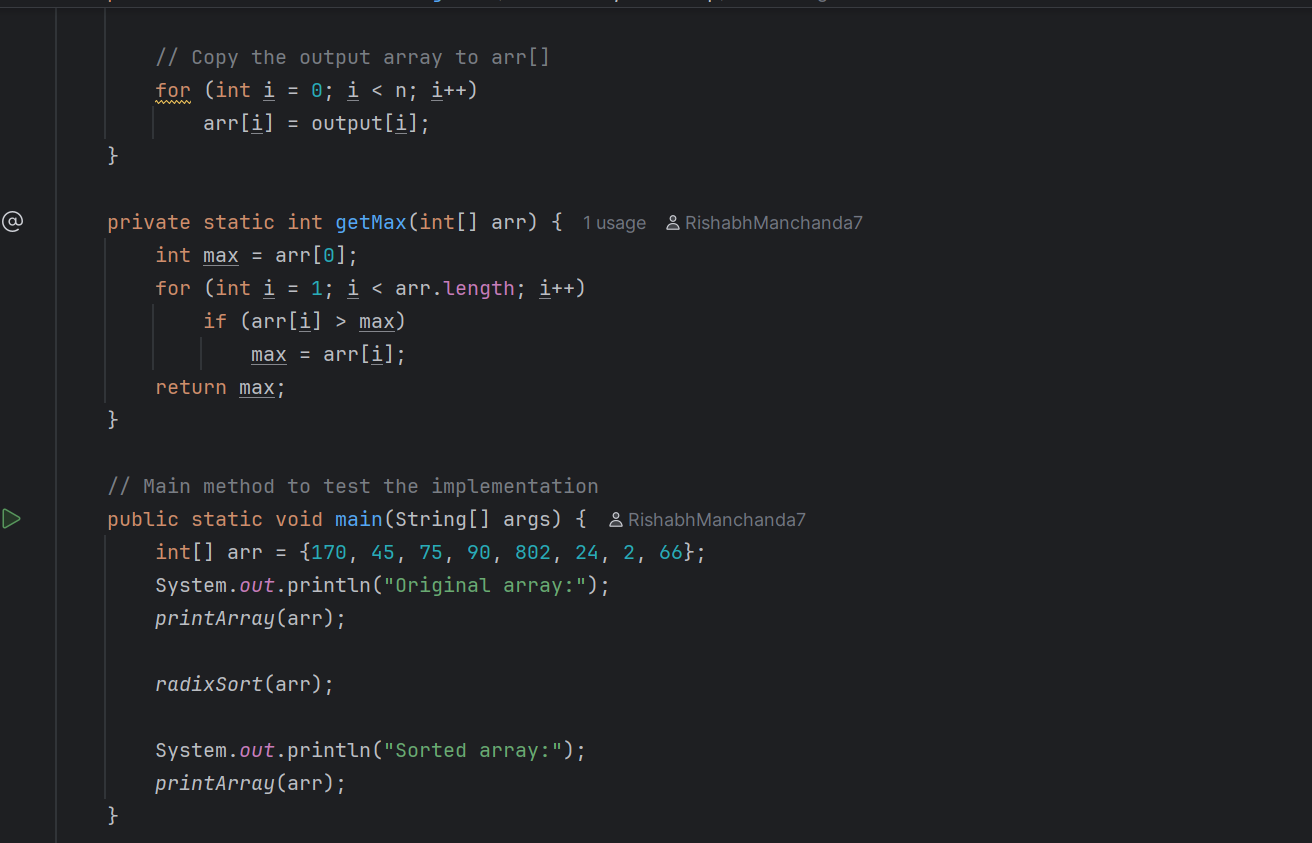
2. To ensure that the highest-priority element always bubbles to the root efficiently.

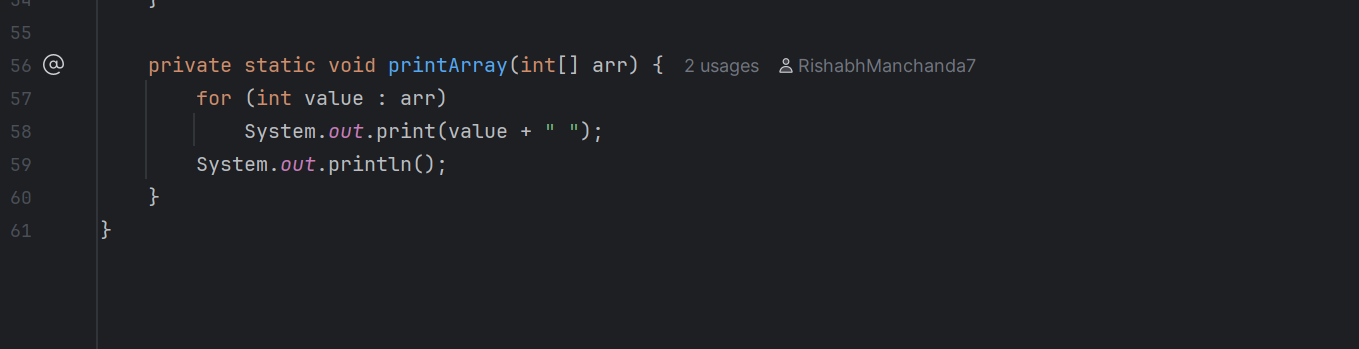
3. To guarantee constant-time insertion and logarithmic-time deletion.

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

TASK-17







Task-18

