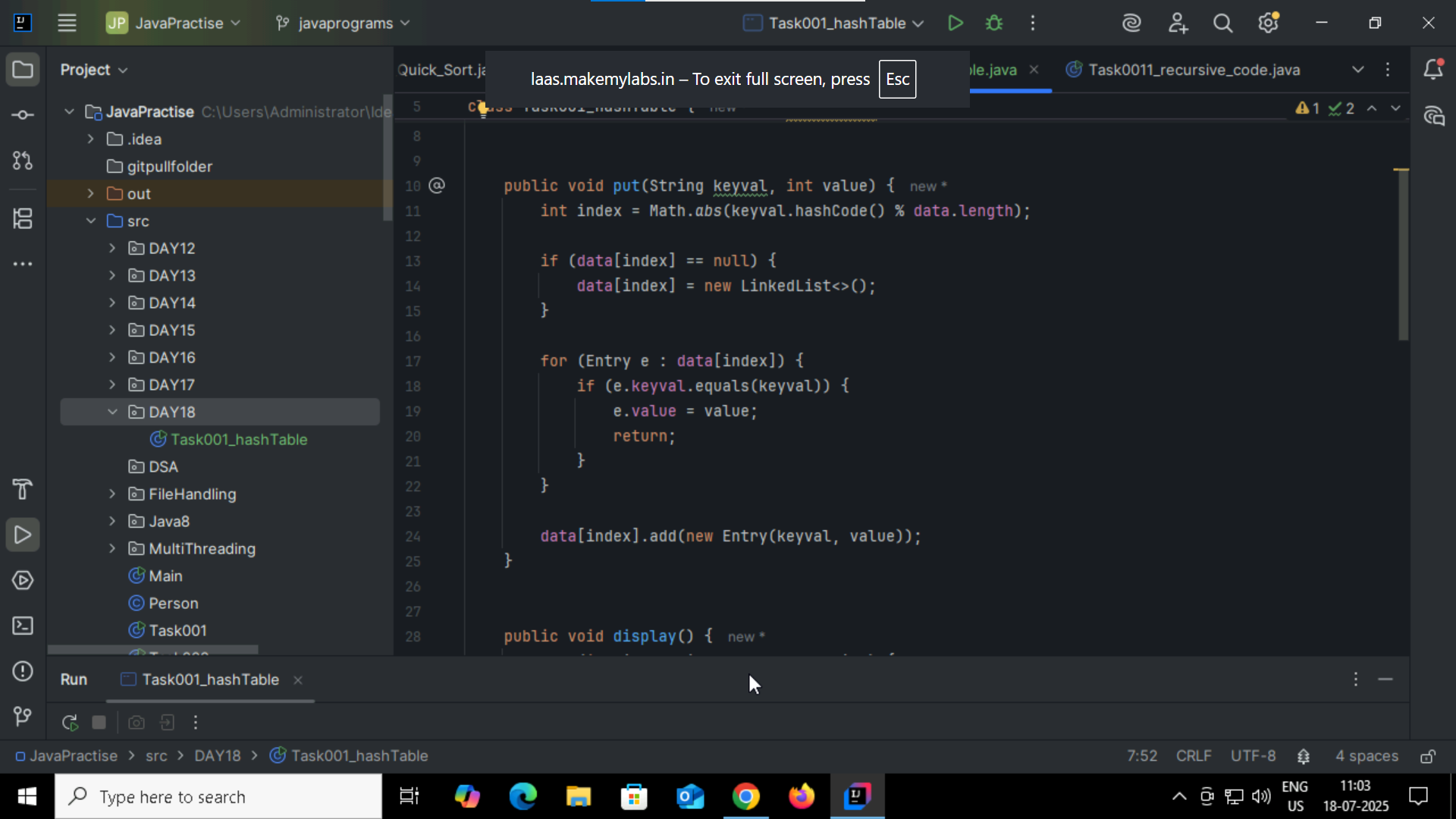
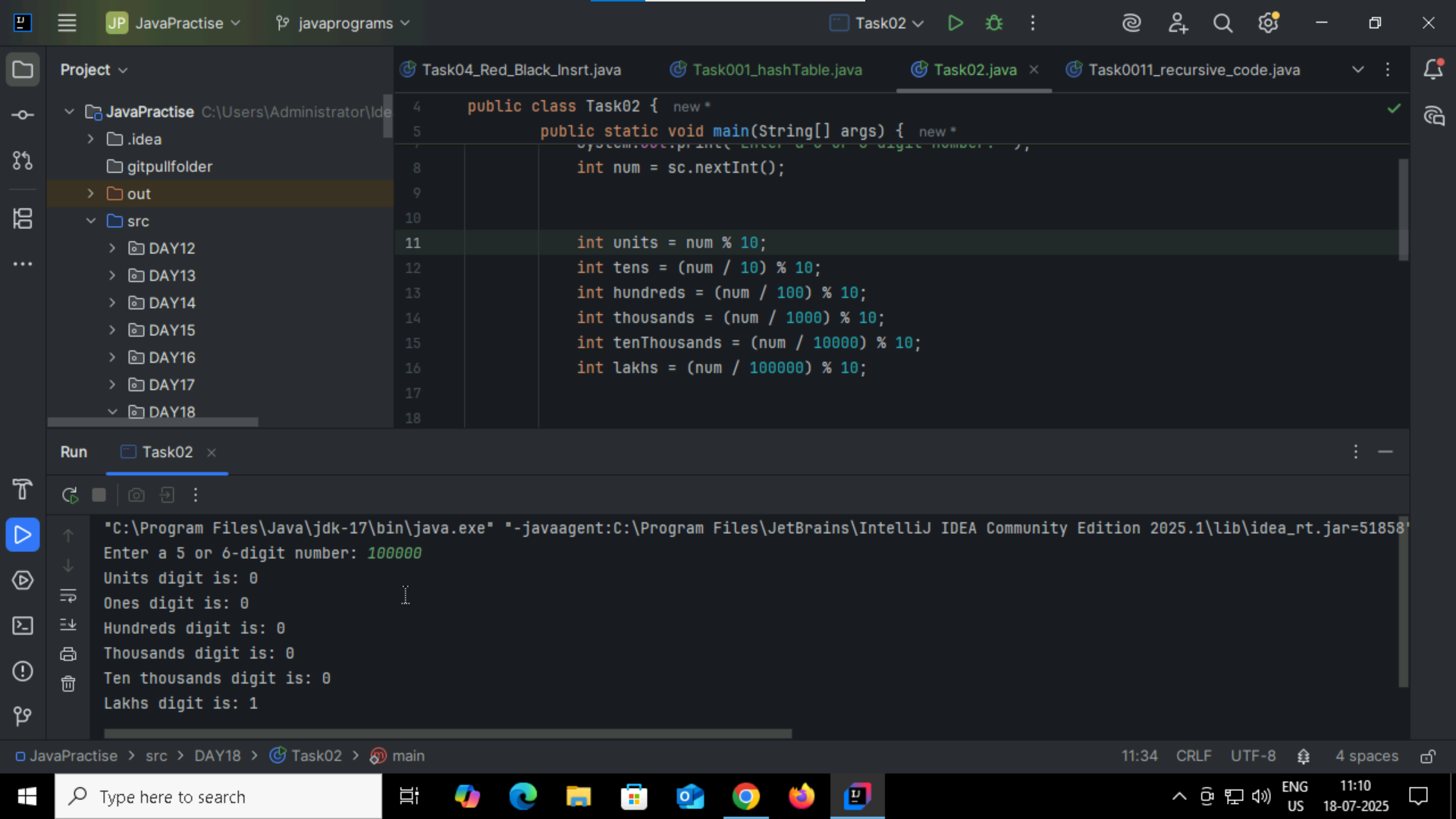
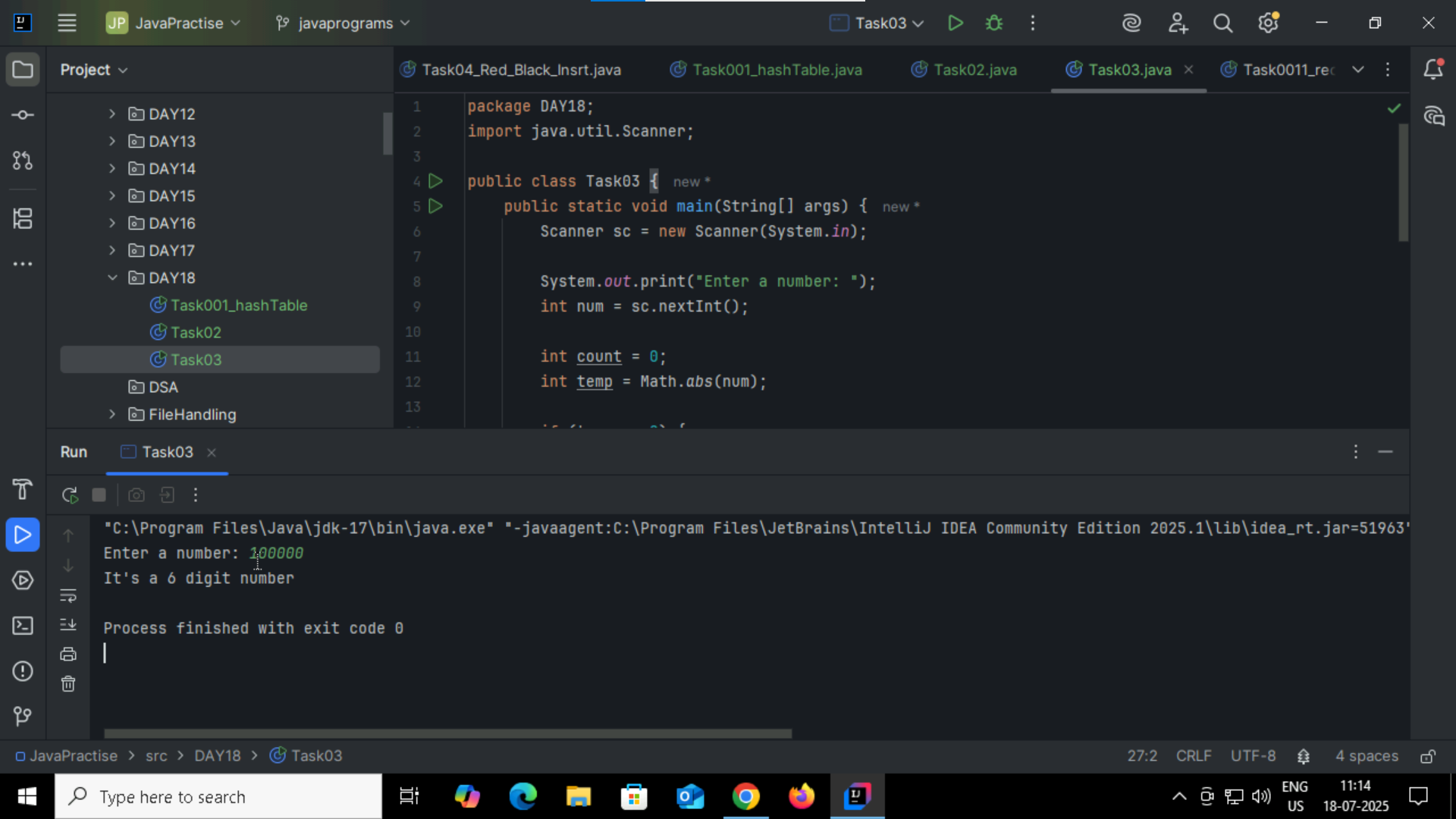
Task01: at each index chaining using a linked list



Task002:



Task03:



Task04:

What are the applications of heap sort?  
Priority Queue - Efficient access to highest priority  
Scheduling - Task/job execution based on priority  
Graph Algorithms   
Data Stream Processing   
Embedded Systems - Memory-efficient sorting   
Simulation Systems - Event scheduling and ordering

Task005:  
  
Do you find any significance change between the breadthFirstSearchRecursive() approach compared to the standard BFS?  
1.Will it the 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?  
4.will it processes nodes in post-order sequence to avoid memory allocation?

Task06:  
  
How does heap sort work ? explain the technique in 5

step1:Construct Max Heap  
Build a max heap from the input array so the largest element is at the root (index 0).  
step2:Swap Root with Last Element  
Swap the root element (maximum) with the last element in the heap (end of the array).  
step3:Shrink Heap Size  
Reduce the heap size by 1 to exclude the last (now sorted) element from the heap.  
step4:Heapify the Root  
Call heapify() on the root to restore the max-heap property in the reduced heap.  
Step5:Repeat Steps 2–4  
Keep repeating until the heap size becomes 1. The array is now sorted in ascending order.  
  
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.  
  
Task08:

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.

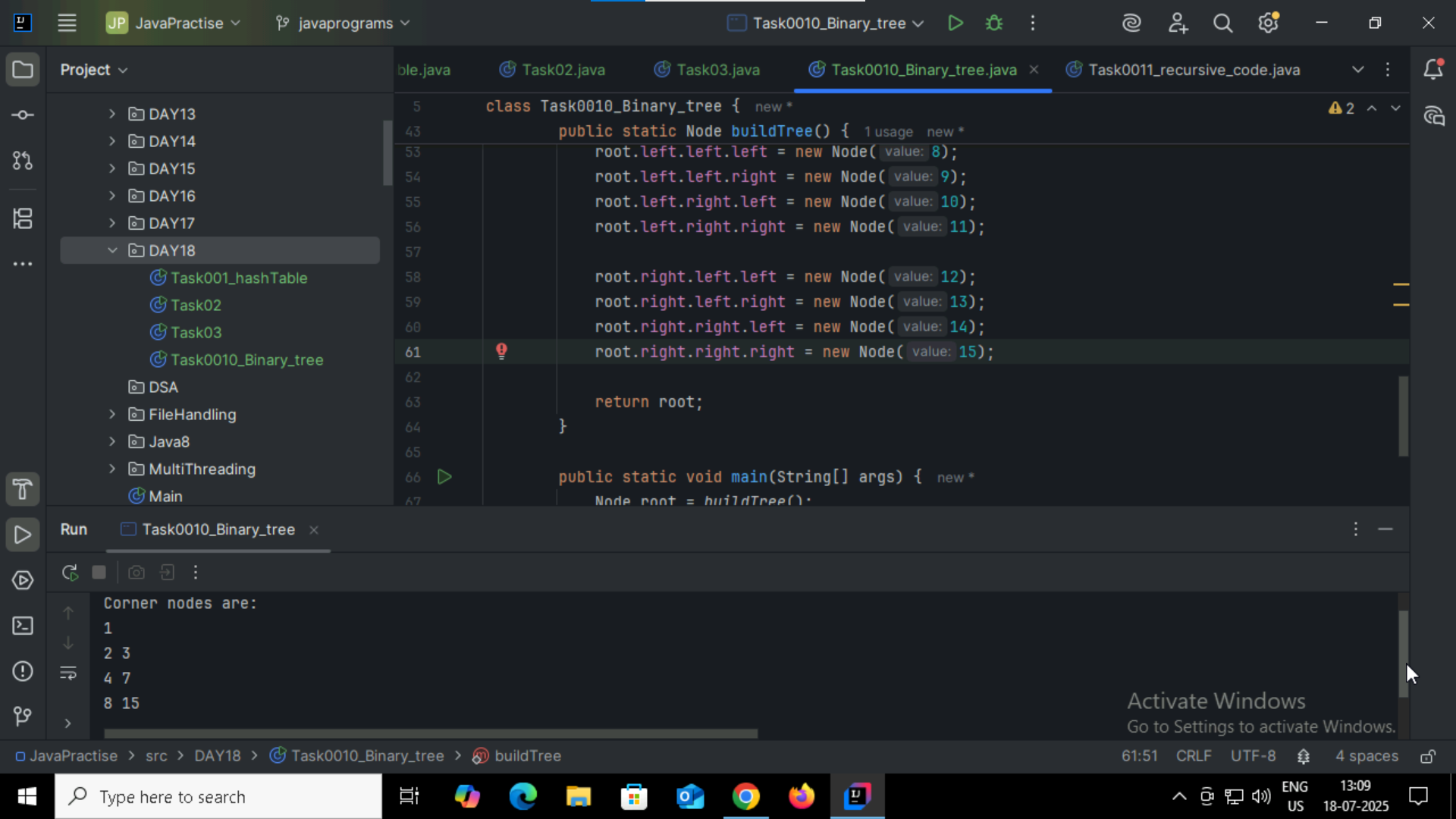
Task09:

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.

Task010:

Which concept explains how 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.

Task010:



Task 12:

How does this binary search function behave on unsorted arrays?

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?

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

Task 14:  
  
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

Step 1: Find the maximum number in the array to determine the number of digits (i.e., the number of passes needed).  
Step 2: Starting from the least significant digit (unit place), perform a stable sort (typically Counting Sort) based on that digit.  
Step 3: Move to the next significant digit (tens, hundreds, etc.) and sort again.  
Step 4: Repeat Step 2 and Step 3 until the most significant digit is processed.  
Step 5: After all passes, the array will be sorted.

Task 16:

Write pseudo code for radix sort

Radix\_Sort(Array, p) // p is the number of passes

for j = 1 to p do

int count\_array[10] = {0};

for i = 0 to n do

count\_array[key of(Array[i]) in pass j]++ // count array stores the count of key

for k = 1 to 10 do

count\_array[k] = count\_array[k] + count\_array[k-1]

for i = n-1 downto 0 do

result\_array[ count\_array[key of(Array[i])] ] = Array[j]

count\_array[key of(Array[i])]--

for i=0 to n do

Array[i] = result\_array[i]

the end for(j)

end function

Task017:  
