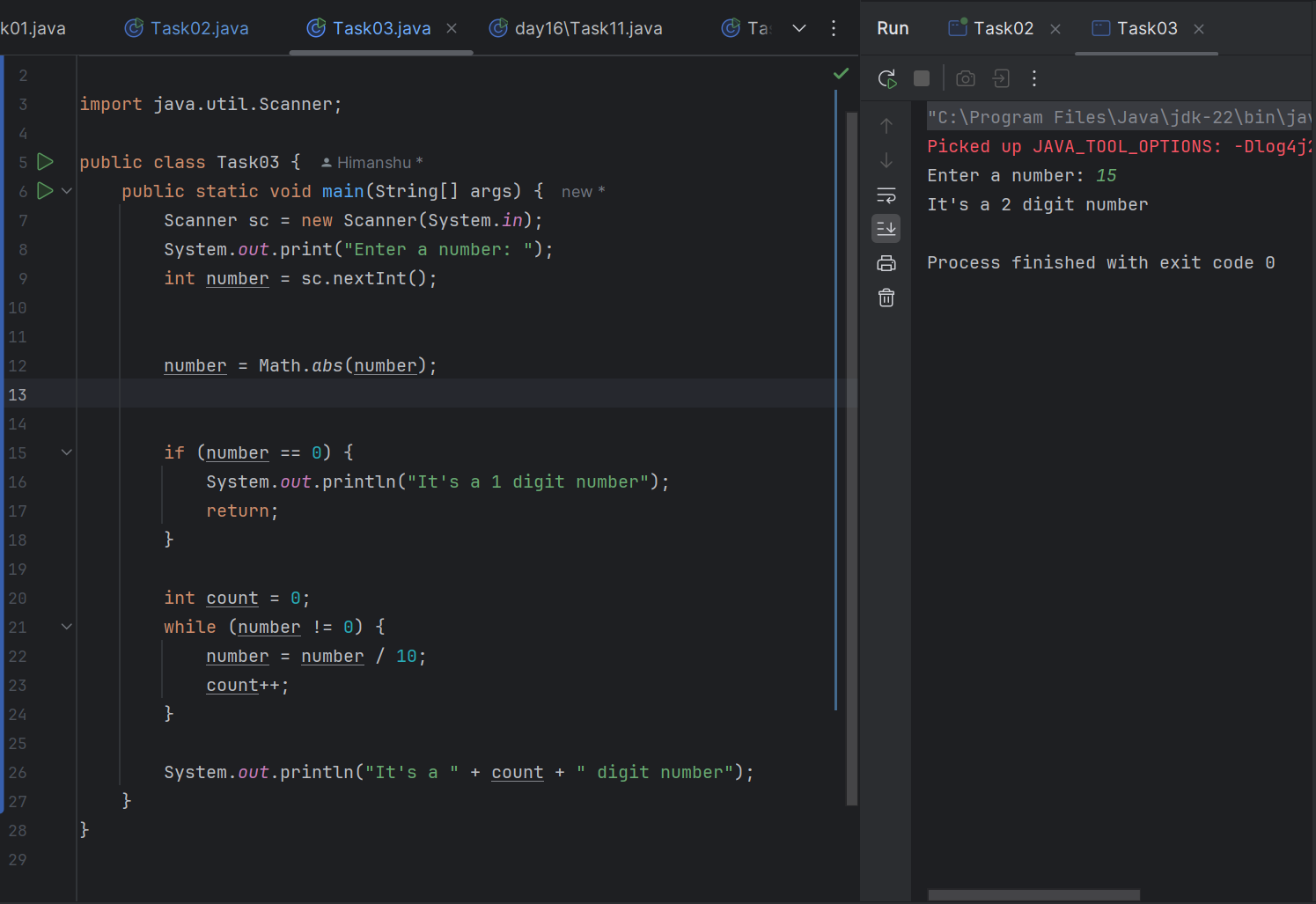
Task 03



Task 04

What are the applications of heap sort?

1. Job scheduling in operating systems
2. CPU task prioritization
3. Network packet routing
4. Embedded systems where memory is limited.
5. Sorting large files where stable sort isn’t necessary.

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?
2. Will it simplify implementation by using queues implicitly within recursive function calls?
3. will it achieve the same result but emphasizes on recursive style using the same level-order logic with explicit queue management?
4. will it process nodes in post-order sequence to avoid memory allocation?

Task 06

1. Build a Max Heap

2. Swap Root with Last Element

3. Reduce Heap Size

4. Heapify the Root Again

5. Repeat Until SortedTask 07

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?

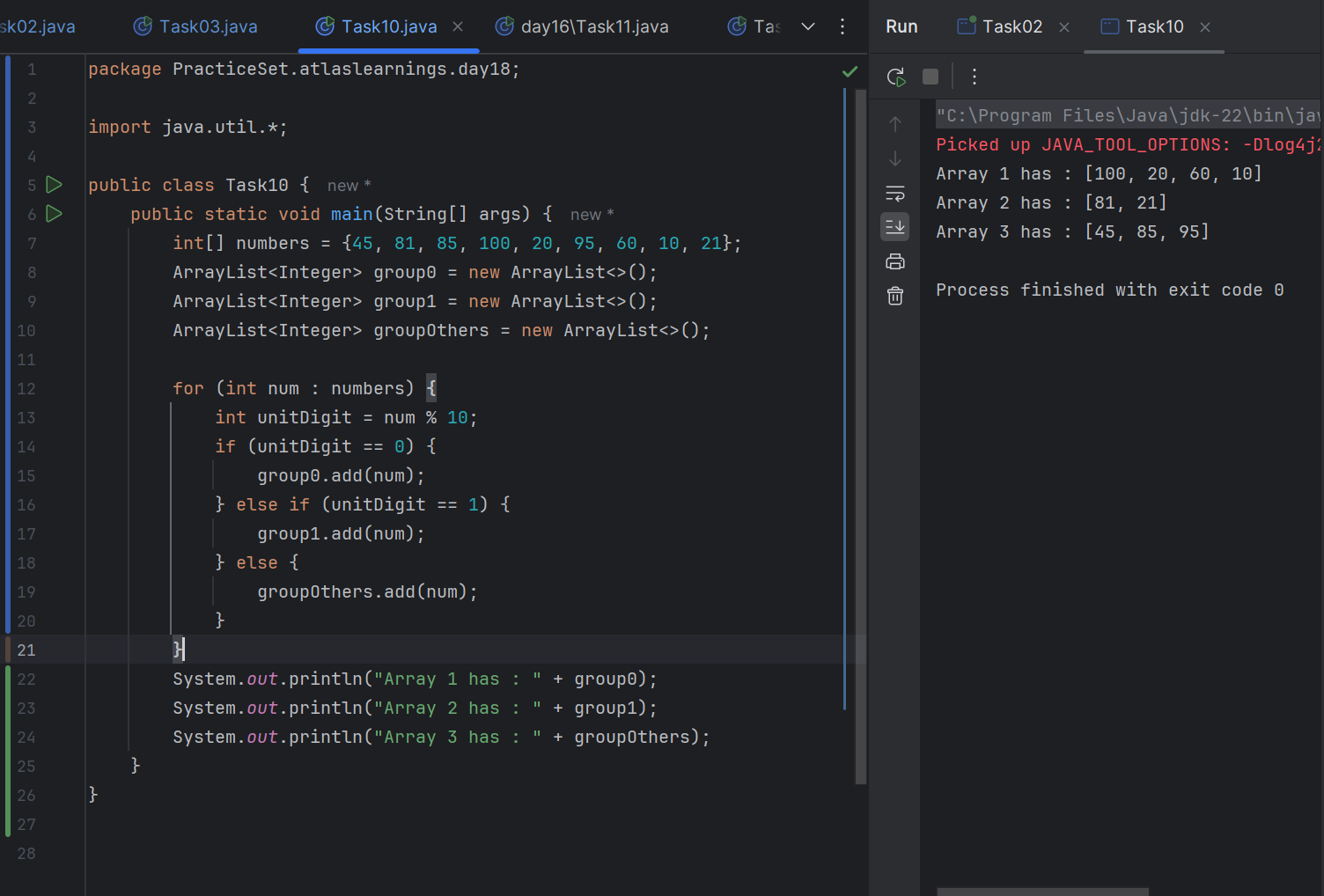
1. To maintain keys in alphabetical order for efficient string processing.

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

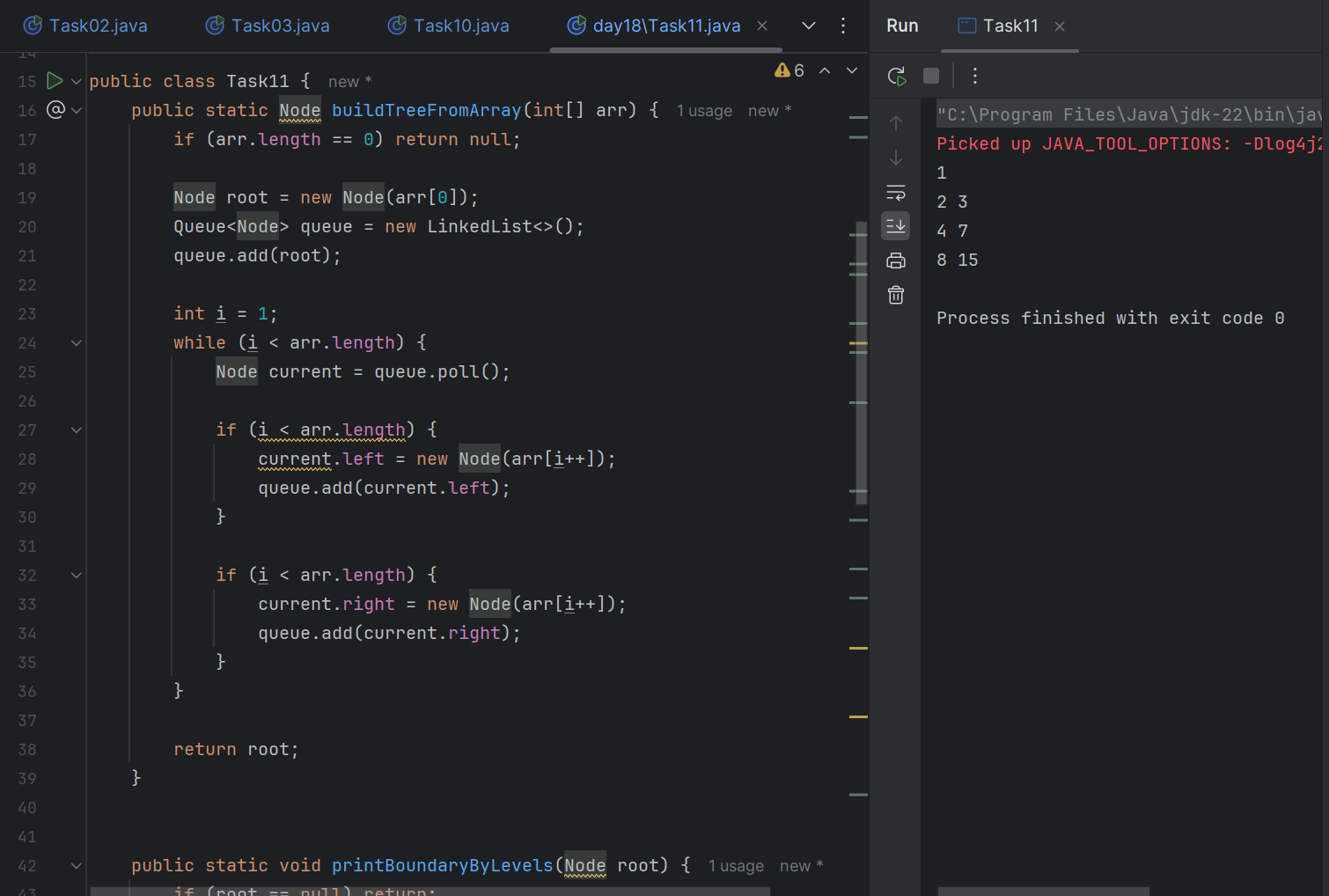
3. To guarantee constant-time insertion and logarithmic-time deletion.

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

Task 10



Task 11



**Task 15**

**Initialize 10 queues** (for digits 0 through 9).  
 These queues will temporarily store numbers grouped by digit values.

**Set power = 1**.  
 This represents the digit position:

**Repeat d times** (for each digit place from least significant to most significant):  
  
 **Distribute Phase:**

For each number in the array:

Extract the digit at the current power place:  
 (number / power) % 10

Add the number to the queue corresponding to that digit.

**B. Collect Phase:**

Traverse queues from 0 to 9:  
While the queue is not empty:  
Remove the front number and place it back into the array in order.

**C. Update Power:**Multiply power by 10 to move to the next digit position.

**After all digit places are processed**, the array is sorted.

**Task 16**

function radixSort(array, d):

create a list of 10 queues (digitQueues) for digits 0 to 9

power ← 1

for i from 0 to d - 1 do:

// Distribute phase

for each number in array:

digit ← (number / power) % 10

digitQueues[digit].enqueue(number)

// Collect phase

index ← 0

for digit from 0 to 9 do:

while digitQueues[digit] is not empty:

array[index] ← digitQueues[digit].dequeue()

index ← index + 1

power ← power \* 10

Task 17

