Day 18 - 104608492 - Shirisha Perapagu

Heap Sort and Radix Sort

Task 01

What kind of collision resolution strategy is implemented in the below Hash Table ?

import java.util.\*;

class Task01 {

    LinkedList<Entry>[] data = new LinkedList[10];

    public void put(String keyval, int value) {

        int index = Math.abs(keyval.hashCode() % data.length);

        if (data[index] == null) {

            data[index] = new LinkedList<>();

        }

        for (Entry e : data[index]) {

            if (e.keyval.equals(keyval)) {

                e.value = value;

                return;

            }

        }

        data[index].add(new Entry(keyval, value));

    }

    static class Entry {

        String keyval;

        int value;

        Entry(String k, int v) {

            keyval = k;

            value = v;

        }

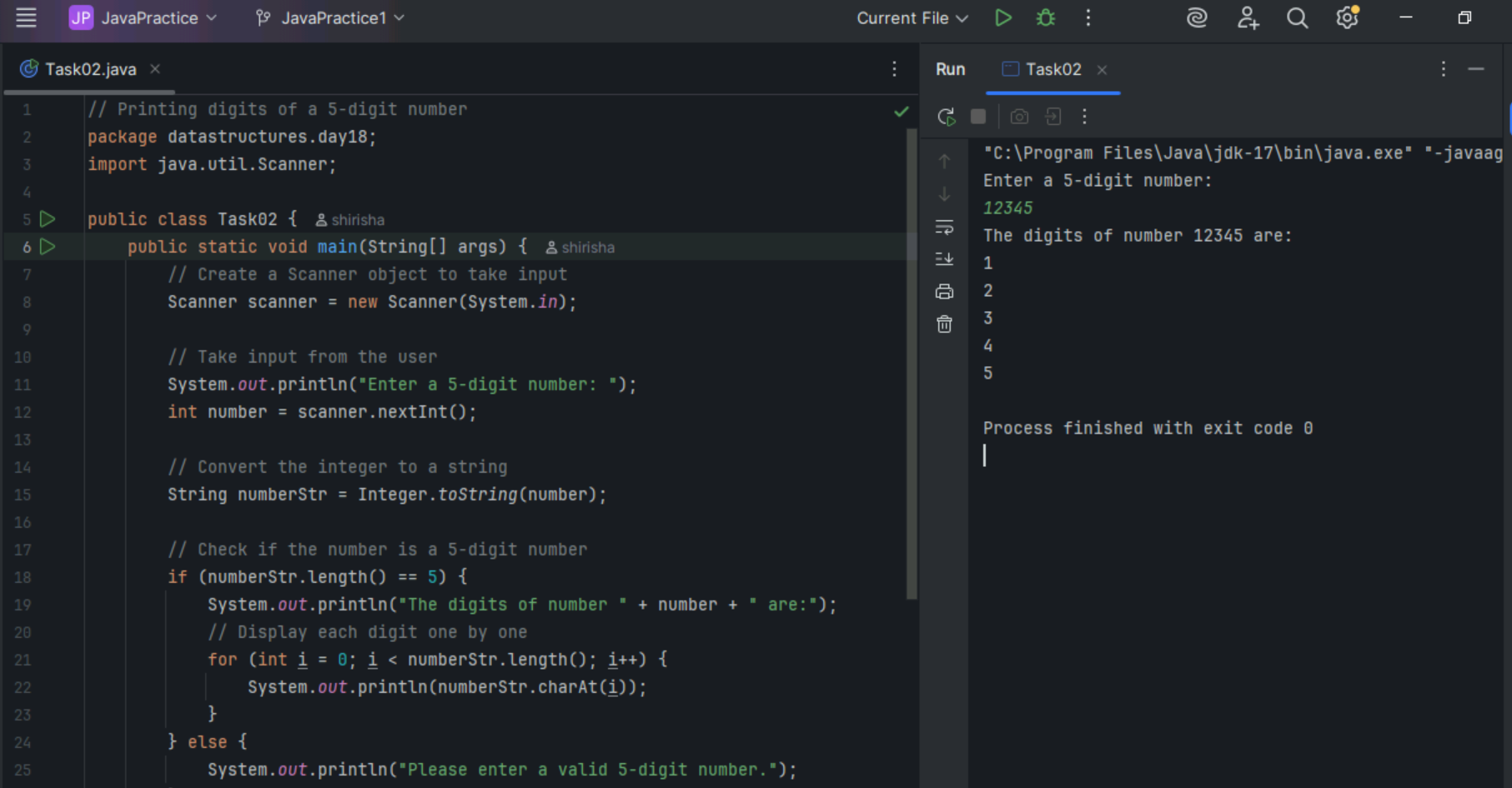
    }

}

1. to fill collisions is it linear probing with backtracking
2. Opening address by placing values at next available bucket
3. at each index chaining using a linked list
4. on each collision resizing hash table

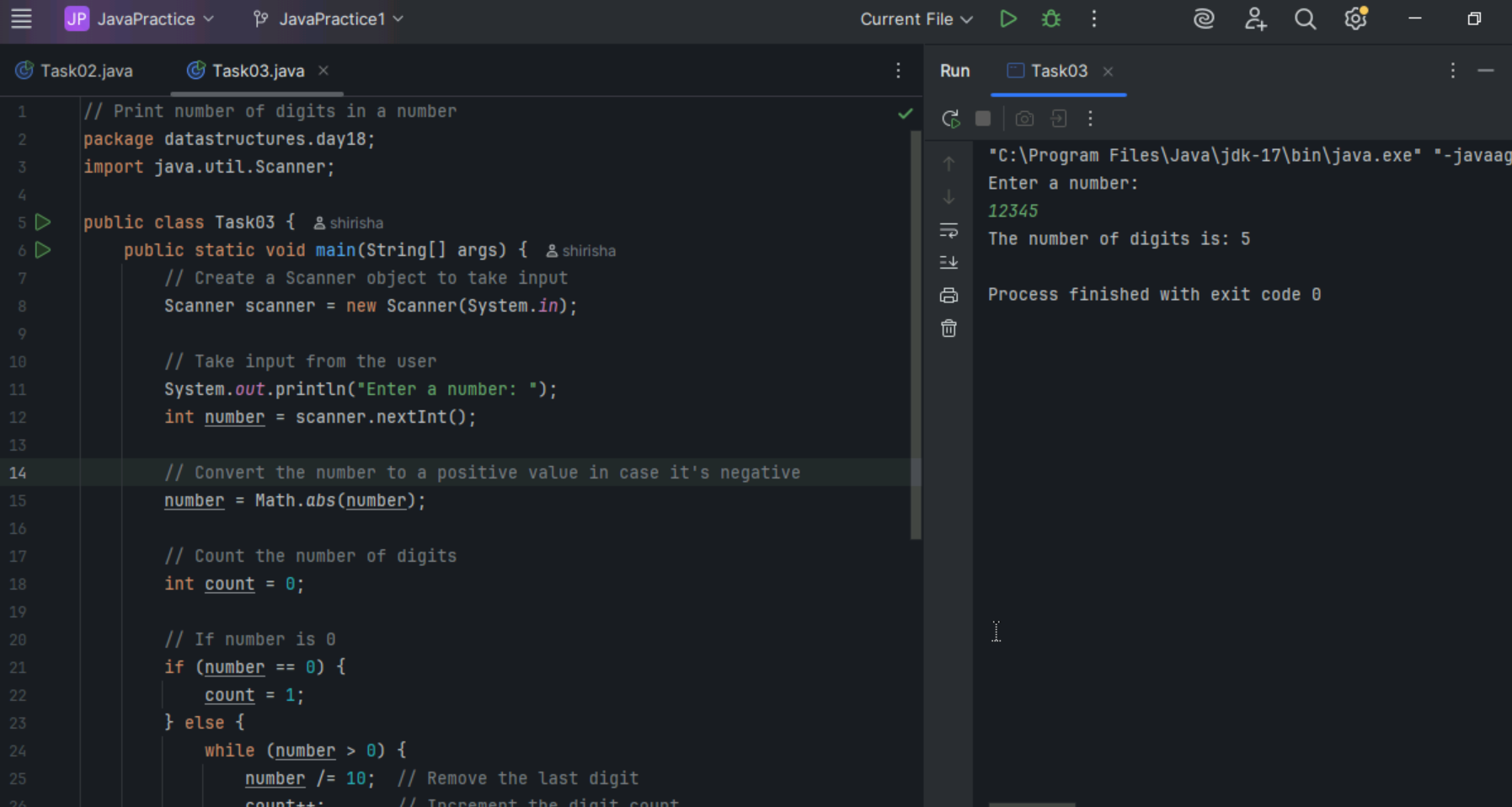
Task 02

Wap to take input from the user a 5 digit no and display digit by digit in the output



Task 03

Wap to take number from the user and display the no. of digit it has



Task 04

What are the applications of heap sort?

1. To implement priority queues.
2. Graph algorithms like Dijkstra's shortest path and Prim's MST.
3. Find the top k largest or smallest elements in a dataset.
4. External sorting where data is too large to fit in memory.
5. Search engines - To rank web pages based on relevance.

Task 05

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?
4. will it processes nodes in post-order sequence to avoid memory allocation?

Task 06

How does heap sort work? Explain the algorithm.

1. Build a Max Heap from array. The largest number is at top.
2. Swap the top number with the last number in array.
3. Shrink the heap by ignoring the last number as its sorted.
4. Fix the heap i.e. heapify so the new largest number goes to top.
5. Repeat the same until the whole array is sorted.

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.

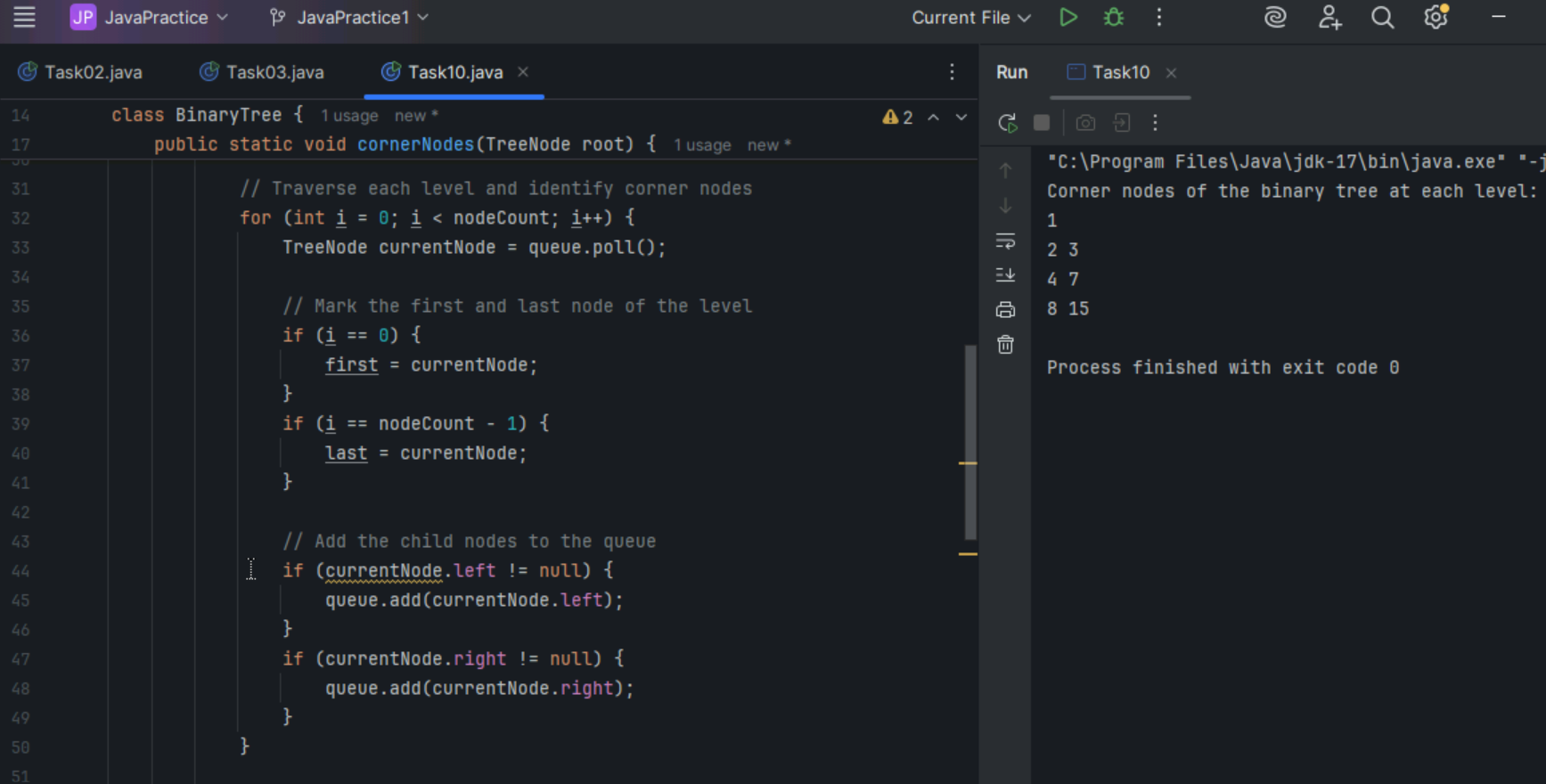
2. To ensure that the highest-priority element always bubbles to the root efficiently.

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

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

Task10

Program to print corner nodes at each level of Binary tree.



Task 11

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.

Task 12

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

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

Radix Sort Algorithm

1. Find the largest number in the list to know how many digits we need to process.
2. Sort all numbers by the one’s place using counting sort.
3. Then sort all numbers by ten’s place.
4. Then 100s place and so on..
5. After going through all digits places, the list is sorted.

Task 16

Radix Sort Pseudocode

procedure radixSort(array)

max = getMax(array)

exp =1

while max/ep > 0

countingSortByDigit(array,exp)

exp = exp \*10

end while

end procedure

procedure countingSortByDigit(array, exp)

output = new array of same length

count = array of size 10 (for digits 0-9)

for i=0 to array.length -1

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

count[index]++

end for

for i= 1 to 9

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

end for

for i = array.length -1 to 0

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

output[count[index] – 1] = array[i]

count[index]--

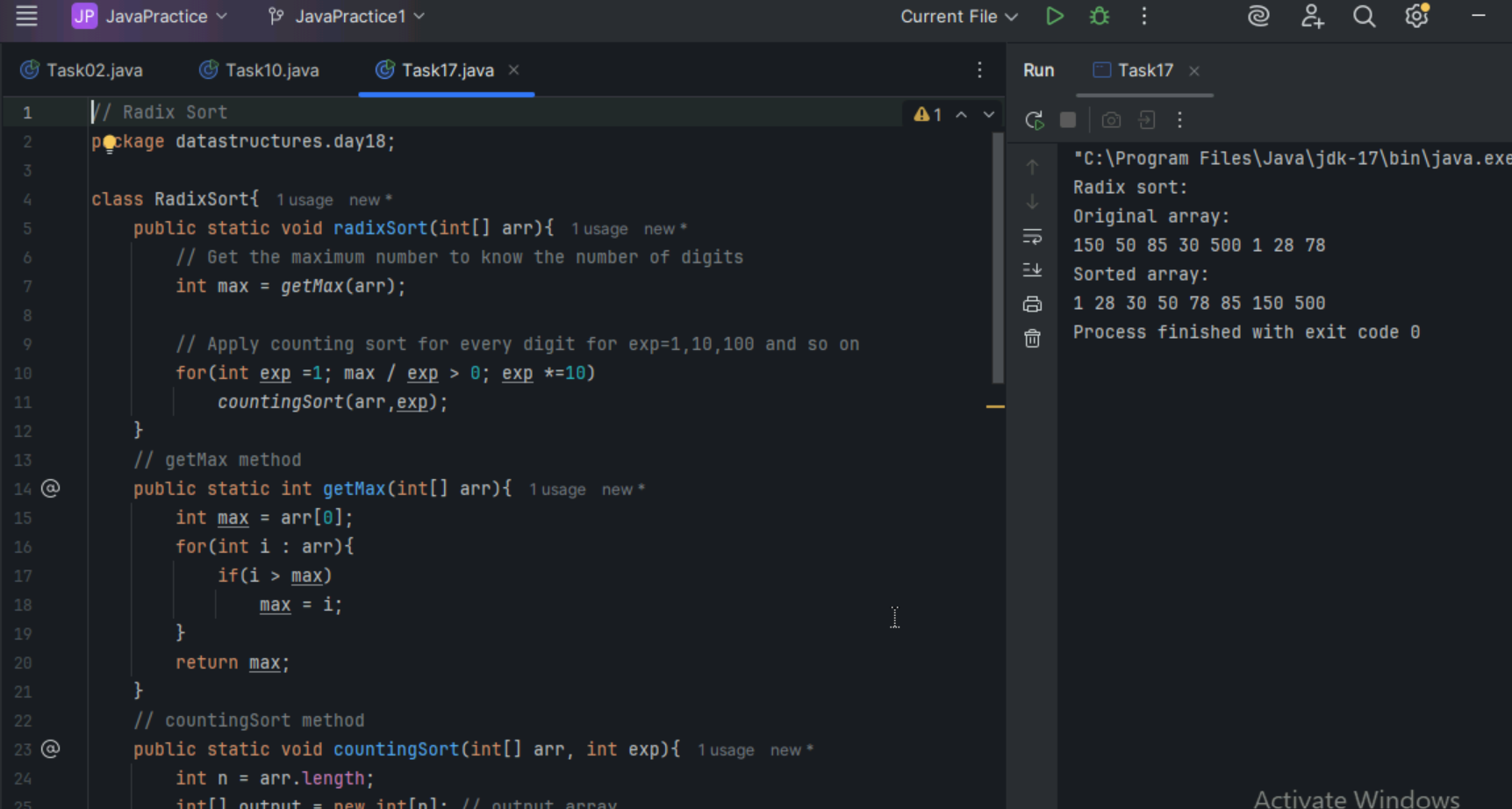
end for

copy output to array

end procedure

Task 17

Radix sort code



Task 18

What causes a stack overflow error in recursive functions?

1. Excessive memory allocation in the heap due to global variables.

2. Infinite iteration loops that do not update the loop variable.

3. Recursion that lacks a proper base case or makes too many nested calls, exhausting the call stack.

4. Function calls that return too quickly without completing the recursion tree.

========================================================================

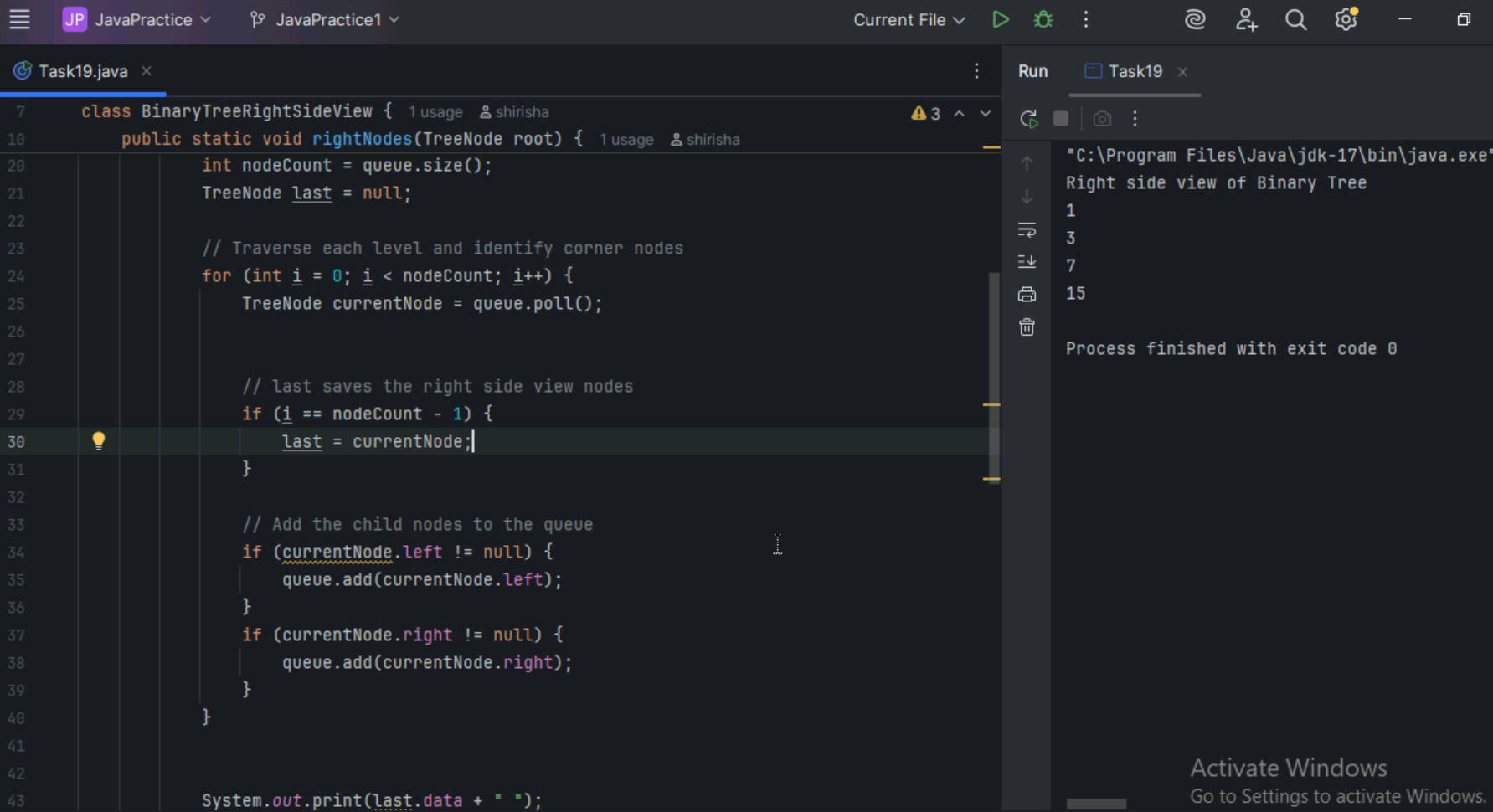
Home Task:

========================================================================

Task 19

<https://leetcode.com/problems/binary-tree-right-side-view/description/>

Right Side View of Binary Tree



Task 20

Reverse Alternate Levels Nodes of Binart Tree

