Day 18 - 16th july 2025

Name:Aravind Kasanagottu

ID: mvsnarav

Task01

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;

        }

    }

}

is it using

to fill collisions is it linear probing with backtracking

or

Opening address by placing values at next available bucket

or

at each index chaining using a linked list

or

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

Hint:

If input is  456897

Output:

units digit is 7

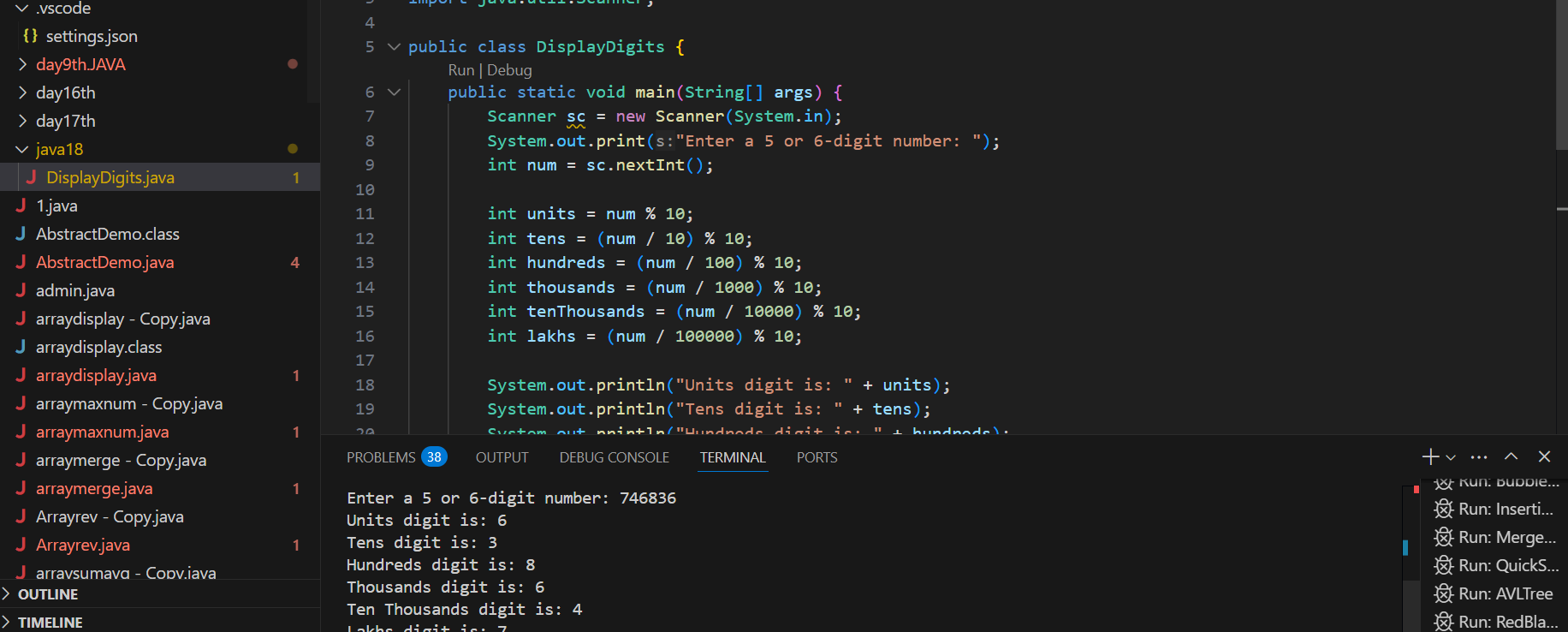
Ones digit is 9

Hundreds digit is 8

Thousands digit is 6

10 thousands digit is 5

Lakhs digit is 4



Task 03:

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

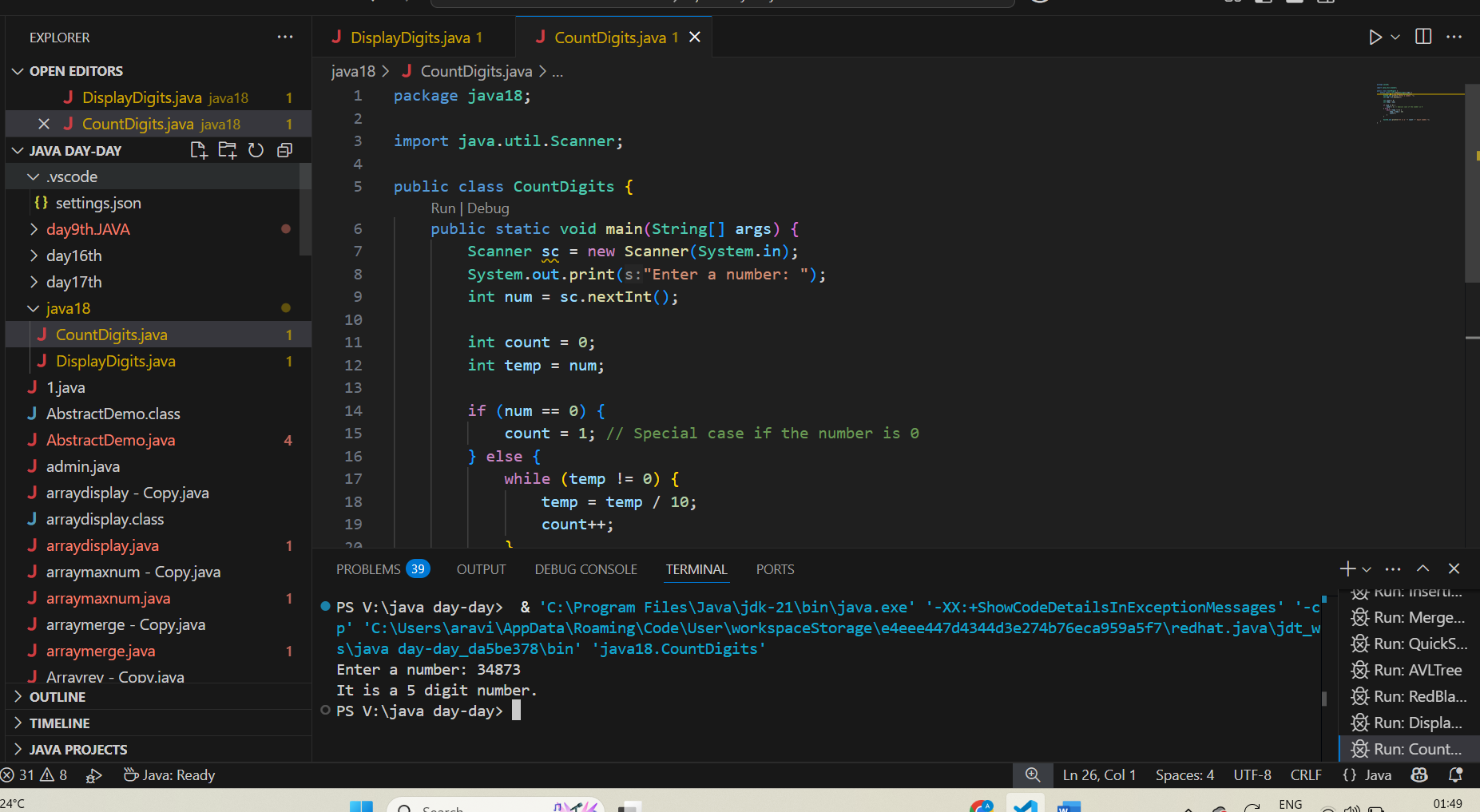
HInt:

If input is:

10,000

Output:

Its a 5 digit number



Task 04:

Wap to display the groups of digits depending upon the unit digits

Hint:

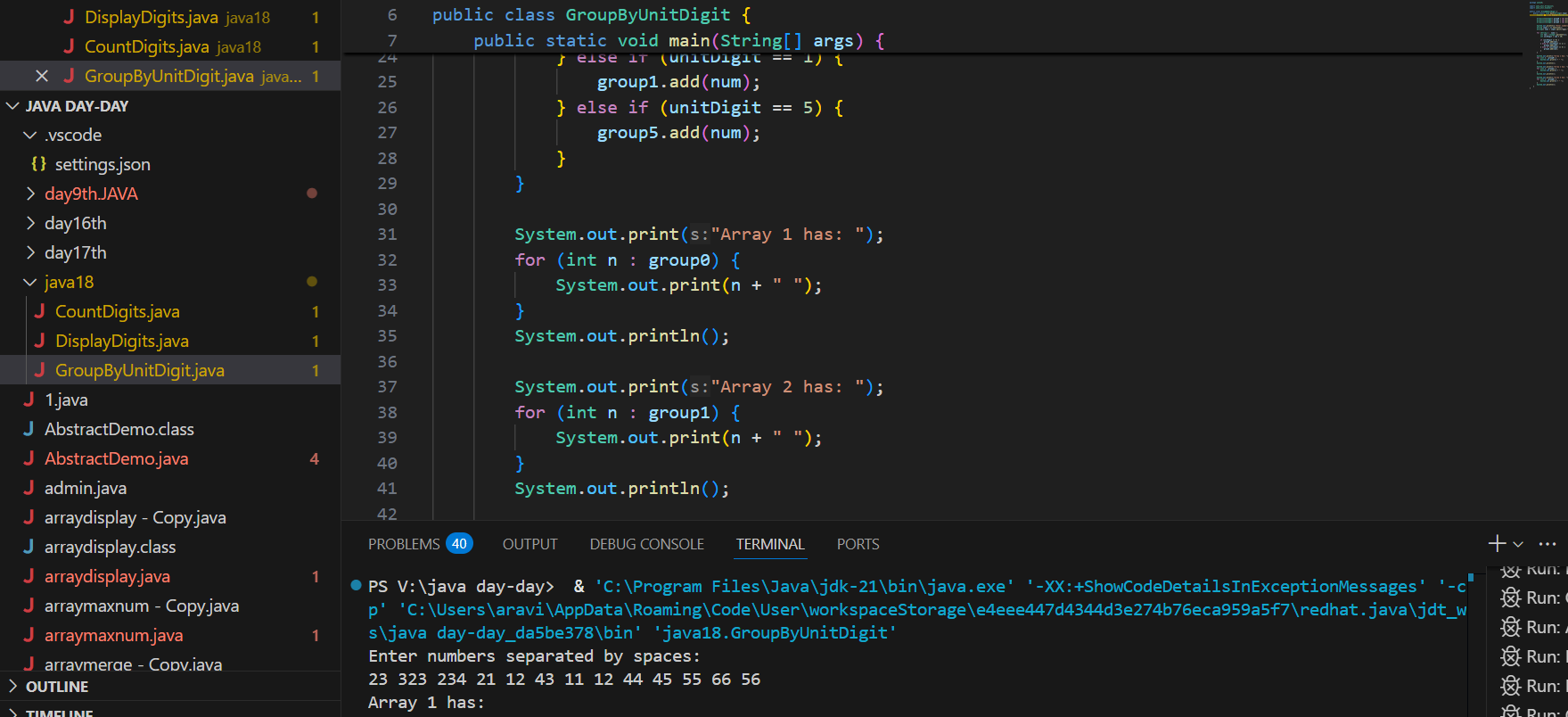
If input is 45,81, 85,100,20. 95,60,10,21

Output:

Array 1 has : 100,20,60,10

Array 2 has : 81, 21

Array 3 has : 45 , 85 ,95



Task 05:

Write algo for radix sort

1️ Check if all the numbers have the **same number of digits.**  
If not, find the **maximum digit length** in the list and add **leading zeroes** to shorter numbers.

2️ Pick the **units place (least significant digit)** from each number.

3️ Using **counting sort**, sort the numbers based on these unit digits, rearranging the original list accordingly.  
(For decimal numbers, possible digit values are 0-9 for indexing.)

4️ Now, move to the **next digit place (tens place) and repeat counting sort** based on this digit, rearranging the list again.

5️ Repeat the above step for **each digit place until the highest place digit is processed.**

6️ After all digit places are sorted, the **final list will be sorted in ascending order.**

Task 06:

Write pseudo code for radix sort

RadixSort(arr[], n):

max = arr[0]

for i = 1 to n - 1:

if arr[i] > max:

max = arr[i]

pos = 1

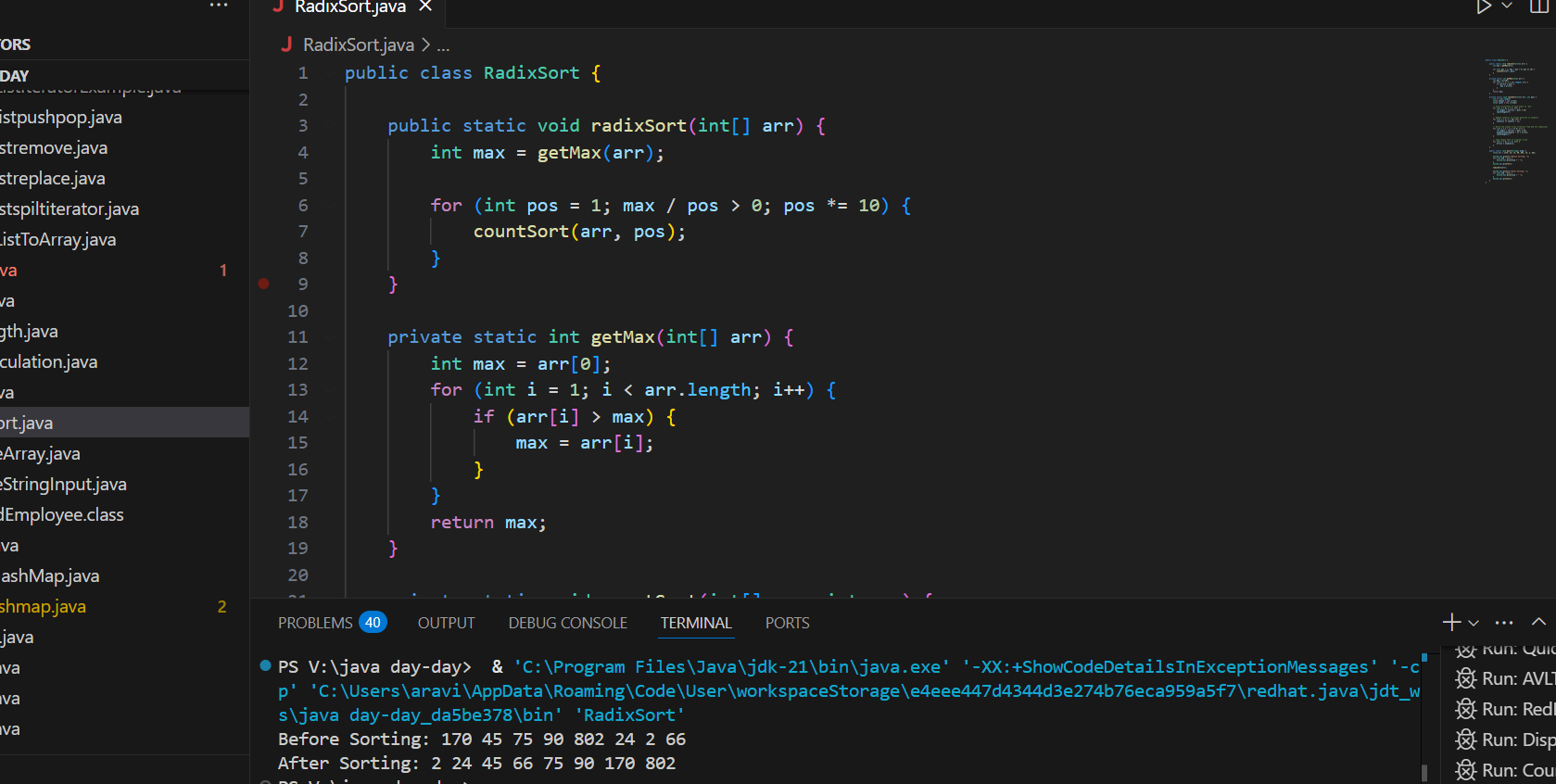
while max / pos > 0:

CountSort(arr, n, pos)

pos = pos \* 10

Task 07:

Write code for radix sort



Task 08:

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?

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

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

or

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

Task 09:

What is memoization?

Write in ur  own words with examples.

**Memoization** is a technique where we **store the results of expensive function calls in a table (or array/map) so we can reuse them instead of recalculating.**  
 It helps to **reduce time complexity** in problems with **overlapping subproblems**.

Instead of **recalculating the same result again and again**, we **remember it for future use**.

Task 10:

What do you understand by Dynamic Programming

Write in your own words with examples.

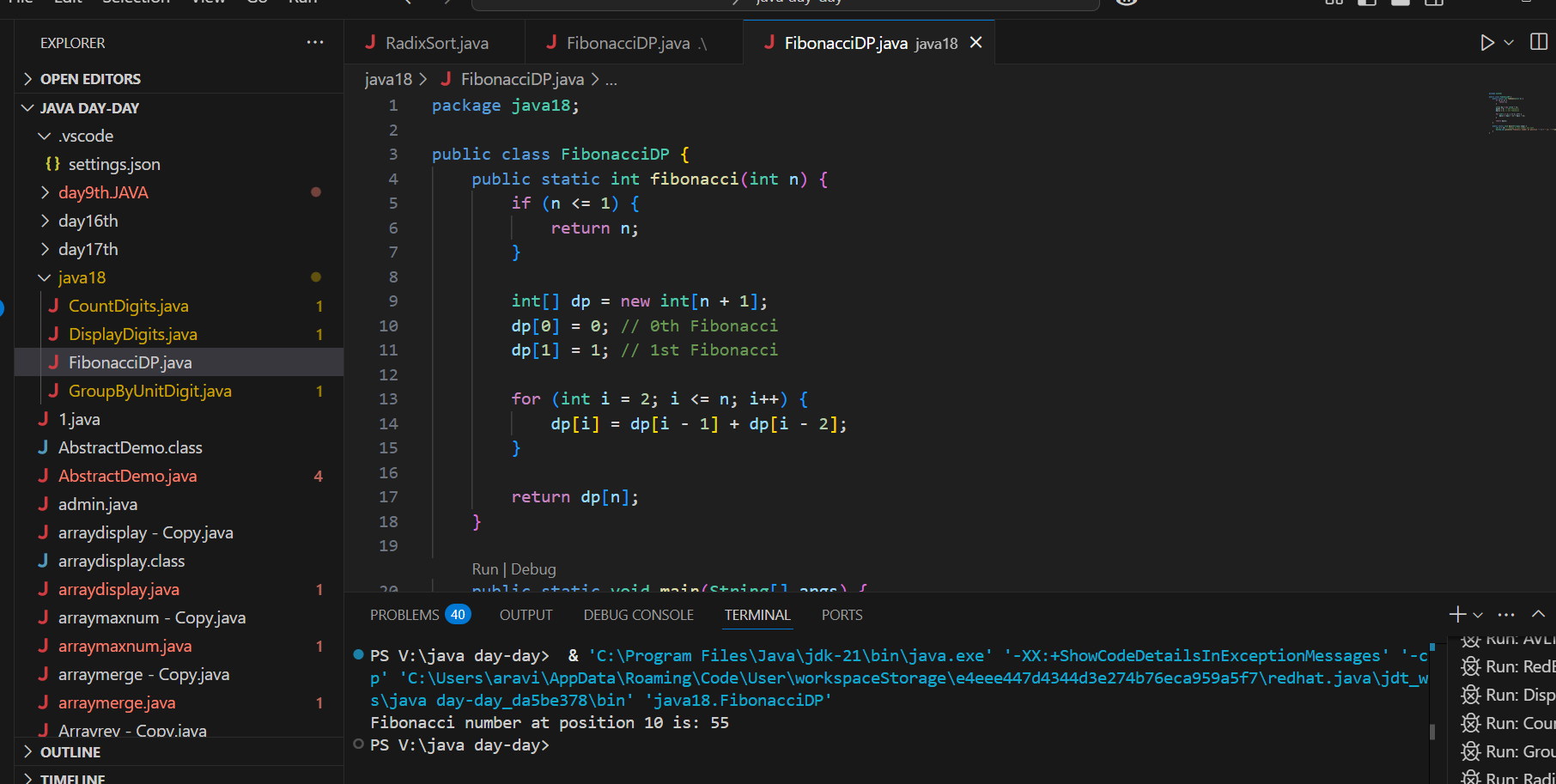
**Dynamic Programming (DP) is a method to solve complex problems by breaking them into smaller subproblems, solving each subproblem only once, and storing their results for reuse**.

* Break the problem into smaller parts.
* Solve each small part.
* Store the answer.
* Use the stored answer whenever needed instead of recalculating.

12.21 to 12.24

Task 11:

Can you write fibonacci series using dynamic programming?



Task 12:

How does heap sort work ? explain the technique

**Heap Sort** is a **comparison-based sorting technique** that uses a **binary heap data structure (max heap or min heap) to sort elements efficiently.**

**Build a Max Heap:**

* Arrange the array elements into a **max heap** so that the **largest element is at the root** of the heap.

**Swap Root with Last Element:**

* Swap the **root (maximum) element with the last element** of the heap.
* Now, the last element is **correctly placed in its sorted position**.

**Heapify the Reduced Heap:**

* Reduce the heap size by 1.
* Apply **heapify** to the root to maintain the max heap property on the reduced heap.

**Repeat:**

* Repeat **step 2 and step 3** until the heap size becomes 1.

### Why is it called Heap Sort?

Because it **uses the heap data structure to repeatedly extract the maximum element and place it in the correct position** in the array.

TSK 13:

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 14:

Iterative implementations use less memory as they do not require stack frames for each call. True

with regard to the difference in memory usage between recursive and iterative implementations of the same algorithm?

the above statement is true or false

TRUE

Task 15:

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

with regard to stack overflow error in recursive functions.. do you think above statement is true

Task 16:

what happens when inserting keys with the same hash in this custom hash map

public class HashCollision {

     static class Entry {

        String key;

        int value;

        Entry(String key, int value) {

            this.key = key;

            this.value = value;

        }

    }

     List<Entry>[] table = new AL[10];

     public void put(String key, int val) {

        int index = Math.abs(key.hashCode() % table.length);

        if (table[index] == null) {

            table[index] = new AL<>();

        }

        table[index].add(new Entry(key, val));

    }

}

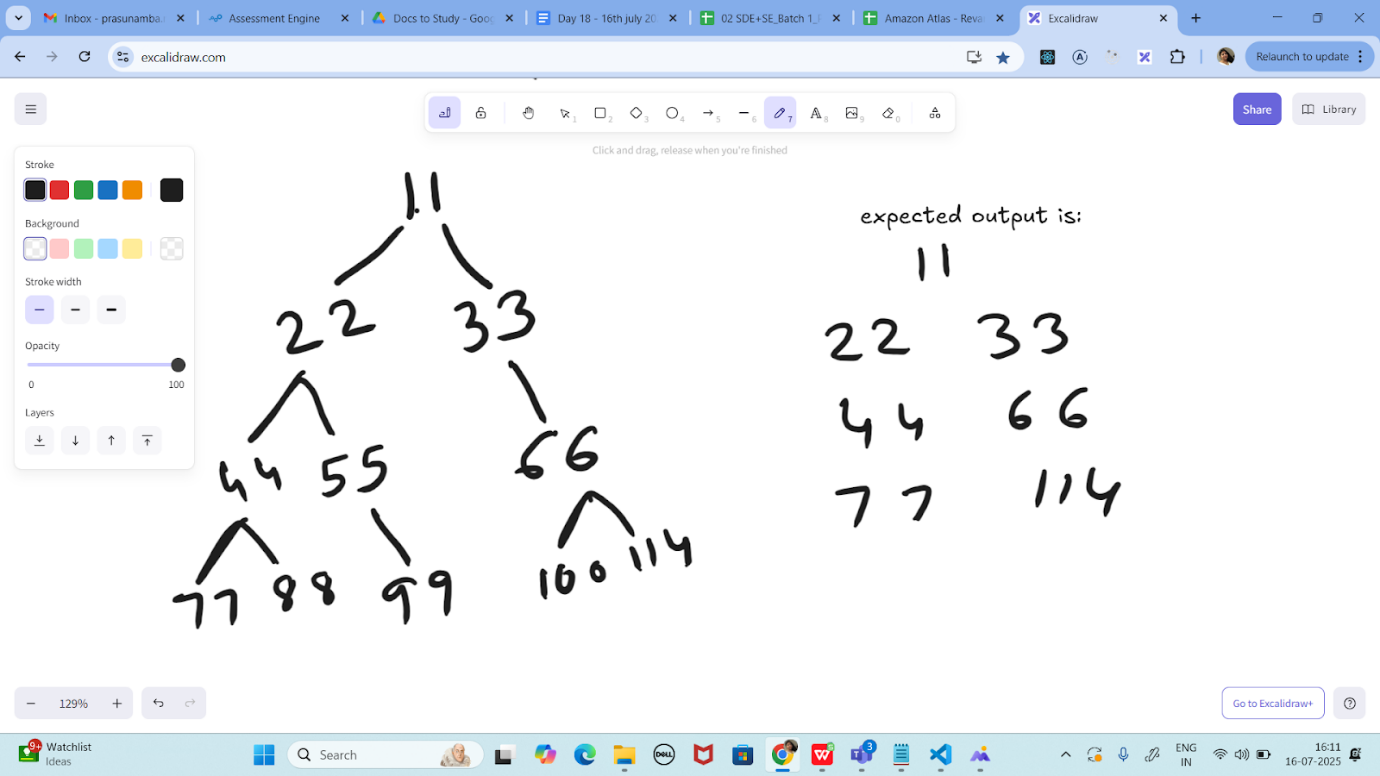
1. Insertion will fail due to duplicate key exception

2. Values are distributed across different buckets using linear probing

3. Only one key-value pair will be stored due to overwriting

4. Multiple values are stored in same bucket via chaining

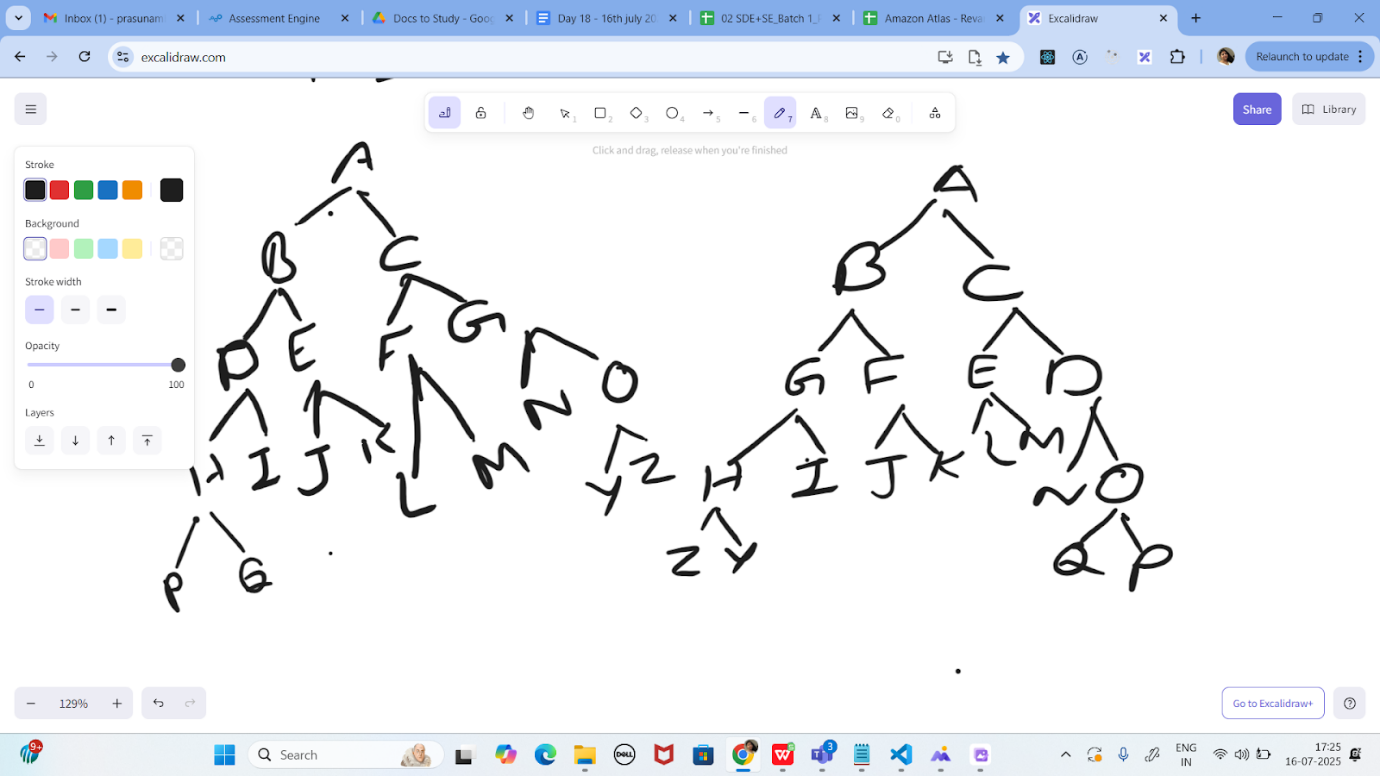
Task 17:



Write a code for binary search tree for expected output

16.10 to 16.18

Task 18:



Print reverse order for alternative levels..

17.26 to 17.36

Task 19:

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

Plz solve this Problem statement

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

Solutions:

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

Tsk 05:

Radix Sort Algo:

1.Check if all the input elements have same number of digits.

If not, check numbers that have maximum number of digits in the list and add leading zeroes to the ones that do not.

2. Take the least significant digit/units digit of each element.

3. Sort these digits using counting sort logic and try to change the order of elements depending on the output achieved.

Sample: if input elements are decimal numbers, possible values each digit can take would be 0-9, so index the digits based on these values.

4. Repeat step 2 for next least significant digits until all digits in given elements are sorted.

5. The final list of elements achieved after kth loop is the sorted output.

Task 06:

Radix Sort Pseudo

RadixSort(arr[], n):

max = arr[0]

   for (i=1 to n-1):

      if (arr[i]>max):

         max=arr[i]

For (pos=1 to max/pos>0):

      countSort(arr, n, pos)

      pos=pos\*10

Task 07:

import java.util.\*;

class RadixSorting {

    static int getMax(int arr[], int n)   {

        int  max = arr[0];

        for (int i = 1; i < n; i++)

            if (arr[i] >  max)

                 max = arr[i];

        return  max;

    }

    static void countSort(int arr[], int n, int exp) {

        int output[] = new int[n];

        int i;

        int count[] = new int[10];

        Arrays.fill(count, 0);

        for (i = 0; i < n; i++)

            count[(arr[i] / exp) % 10]++;

        for (i = 1; i < 10; i++)

            count[i] += count[i - 1];

        for (i = n - 1; i >= 0; i--) {

            output[count[(arr[i] / exp) % 10] - 1] = arr[i];

            count[(arr[i] / exp) % 10]--;

        }

        for (i = 0; i < n; i++)

            arr[i] = output[i];

    }

    static void radixsort(int arr[], int n) {

        int m = getMax(arr, n);

        for (int exp = 1; m / exp > 0; exp \*= 10)

            countSort(arr, n, exp);

    }

    static void print(int arr[], int n) {

        for (int i = 0; i < n; i++)

            System.out.print(arr[i] + " ");

    }

    public static void main(String[] args) {

        int arr[] = { 142,458,70, 45, 75, 90, 802, 24, 2, 66 };

        int n = arr.length;

        radixsort(arr, n);

        print(arr, n);

    }

}

Tsk 11:

class Fibinocci\_DynamicProgramming {

    static int fib(int n) {

        int[] fibArray = new int[n + 2];

        fibArray[0] = 0;

        fibArray[1] = 1;

        for (int i = 2; i <= n; i++) {

            fibArray[i] = fibArray[i - 1] + fibArray[i - 2];

        }

        return fibArray[n];

    }

    public static void main(String[] args) {

        int N = 5;

       for (int i = 0; i < N; i++) {

            System.out.print(fib(i) + " ");

        }

    }

}

Task 17:

import java.util.\*;

class Node {

    int key;

    Node left, right;

    public Node(int key)

    {

        this.key = key;

        left = right = null;

    }

}

class BinaryTreeCornerNodes {

    Node root;

    void printCorner(Node root) {

        Queue<Node> q = new LinkedList<Node>();

        q.add(root);

        // level order traversal

        while (!q.isEmpty()) {

            int n = q.size();

            for(int i = 0 ; i < n ; i++){

             Node temp = q.peek();

             q.poll();// retrieve and remove the node

            if(i==0 || i==n-1)

                System.out.print(temp.key + "  ");

            if (temp.left != null)

                q.add(temp.left);

            if (temp.right != null)

                q.add(temp.right);

        }

        }

    }

    public static void main(String[] args){

        BinaryTree tree = new BinaryTree();

        tree.root = new Node(11);           ====> 11, 22, 33, 44, no 55, no 66, 77

        tree.root.left = new Node(22);

        tree.root.right = new Node(33);

        tree.root.left.left = new Node(44);

        tree.root.left.right = new Node(55);

        tree.root.right.left = new Node(66);

        tree.root.right.right = new Node(77);

        tree.printCorner(tree.root);

    }

}

Task 18:

import java.util.\*;

class Node {

    int data;

    Node left, right;

    Node(int data) {

        this.data = data;

        this.left = this.right = null;

    }

}

class  ReverseAlternate{

    static void storeAlternate(Node root, List<Integer> arr, int lvl) {

        if (root == null) return;

        storeAlternate(root.left, arr, lvl + 1);

        if (lvl % 2 != 0)

            arr.add(root.data);

        storeAlternate(root.right, arr, lvl + 1);

    }

    static void modifyTree(Node root, List<Integer> arr, int lvl) {

        if (root == null) return;

        modifyTree(root.left, arr, lvl + 1);

        if (lvl % 2 != 0) {

            root.data = arr.remove(arr.size() - 1);

        }

        modifyTree(root.right, arr, lvl + 1);

    }

    static void reverseAlternate(Node root) {

        List<Integer> arr = new ArrayList<>();

        storeAlternate(root, arr, 0);

        modifyTree(root, arr, 0);

    }

    static void printInorder(Node root) {

        if (root == null) return;

        printInorder(root.left);

        System.out.print(root.data + " ");

        printInorder(root.right);

    }

    public static void main(String[] args) {

        Node root = new Node(1);

        root.left = new Node(2);

        root.right = new Node(3);

        root.left.left = new Node(4);

        root.left.right = new Node(5);

        root.right.left = new Node(6);

        root.right.right = new Node(7);

        System.out.println("Inorder Traversal of given tree");

        printInorder(root);

        reverseAlternate(root);

        System.out.println("\nInorder Traversal of modified tree");

        printInorder(root);

    }

}

Task 19:

Method 1 using BFS

class Solution {

  public List<Integer> rightSideViewBFS(TreeNode root) {

    if (root == null)

      return new ArrayList<>();

    List<Integer> ans = new ArrayList<>();

    Queue<TreeNode> q = new ArrayDeque<>(List.of(root));

    while (!q.isEmpty()) {

      final int size = q.size();

      for (int i = 0; i < size; ++i) {

        TreeNode node = q.poll();

        if (i == size - 1)

          ans.add(node.val);

        if (node.left != null)

          q.offer(node.left);

        if (node.right != null)

          q.offer(node.right);

      }

    }

    return ans;

  }

}

Or

Method 2 using DFS

class Solution {

  public List<Integer> rightSideViewDFS(TreeNode root) {

    List<Integer> ans = new ArrayList<>();

    dfs(root, 0, ans);

    return ans;

  }

  private void dfs(TreeNode root, int depth, List<Integer> ans) {

    if (root == null)

      return;

    if (depth == ans.size())

      ans.add(root.val);

    dfs(root.right, depth + 1, ans);

    dfs(root.left, depth + 1, ans);

  }

}

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