Date: 11/11/2024

**DSA Practice Problems**

1. 0-1 knapsack problem

public class Knapsack {

    static int calculateprofit(int W, int weight[],int profit[],int n){

        int dp[]=new int[W+1];

        for(int i=0;i<n;i++){

            for(int w=W;w>=weight[i];w--){

                dp[w]=Math.max(dp[w],dp[w-weight[i]]+profit[i]);

            }

        }

        return dp[W];

    }

    public static void main(String[] args){

        int capacity=50;

        int weight[]={ 10,20,30};

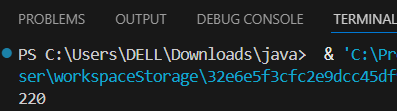
        int profit[]={ 60,100,120};

        int n=profit.length;

        System.out.println(calculateprofit(capacity, weight, profit, n));

    }

}



Time Complexity: O(n\*W)

1. Floor in sorted array

public class FloorSortedArray {

    static int floorSearch(int arr[], int low, int high, int x) {

        if (low > high)

            return -1;

        if (x >= arr[high])

            return high;

        int mid = (low + high) / 2;

        if (arr[mid] == x)

            return mid;

        if (mid > 0 && arr[mid - 1] <= x && x < arr[mid])

            return mid - 1;

        if (x < arr[mid])

            return floorSearch(arr, low, mid - 1, x);

        return floorSearch(arr, mid + 1, high, x);

    }

    public static void main(String[] args) {

        int arr[] = {1, 2, 4, 6, 10, 12, 14};

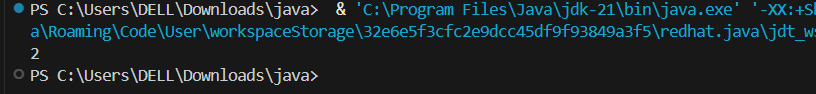
        int n = arr.length;

        int x = 5;

        System.out.println(floorSearch(arr, 0, n - 1, x));

    }

}



Time Complexity: O(log n)

1. Check equal arrays

import java.util.Arrays;

class EqualArray {

    public static boolean CheckEqual(int arr1[], int arr2[]) {

        int n = arr1.length;

        int m = arr2.length;

        if (n!=m)

            return false;

        Arrays.sort(arr1);

        Arrays.sort(arr2);

        for (int i = 0; i < n; i++)

            if (arr1[i] != arr2[i])

                return false;

        return true;

    }

    public static void main(String[] args) {

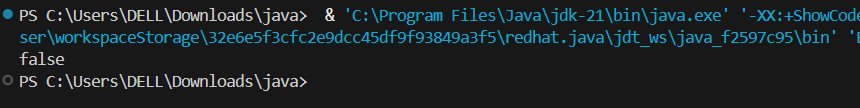
        int arr1[] = {3, 5, 2, 5, 2};

        int arr2[] = {2, 3, 6, 5, 2};

        System.out.println(CheckEqual(arr1,arr2));

    }

}



Time Complexity: O(n log n)

1. Palindrome linked list

public class PalindromeLinkedList {

static class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

static boolean CheckPalindrome(Node head) {

if (head == null || head.next == null) {

return true;

}

int length = 0;

Node temp = head;

while (temp != null) {

length++;

temp = temp.next;

}

Node start = head;

Node end = head;

for (int i = 0; i < length - 1; i++) {

end = end.next;

}

for (int i = 0; i < length / 2; i++) {

if (start.data != end.data) {

return false;

}

start = start.next;

Node tempend = head;

for (int j = 0; j < length - i - 2; j++) {

tempend = tempend.next;

}

end = tempend;

}

return true;

}

public static void main(String[] args) {

Node head = new Node(1);

head.next = new Node(2);

head.next.next = new Node(1);

head.next.next.next = new Node(1);

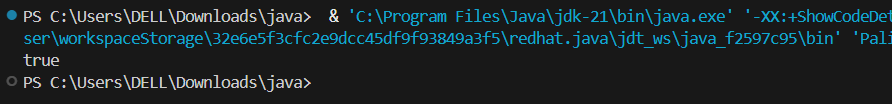
head.next.next.next.next = new Node(2);

head.next.next.next.next.next = new Node(1);

System.out.println(CheckPalindrome(head));

}

}



Time Complexity: O(N)

1. Balanced tree check

class Node {

int data;

Node left, right;

Node(int data) {

this.data = data;

left = right = null;

}

}

class BalancedTreeCheck {

static boolean CheckBalanced(Node root) {

return height(root) != -1;

}

static int height(Node node) {

if (node == null) {

return 0;

}

int leftheight = height(node.left);

int rightheight = height(node.right);

if (leftheight == -1 || rightheight == -1 || Math.abs(leftheight - rightheight) > 1) {

return -1;

}

return Math.max(leftheight, rightheight) + 1;

}

public static void main(String[] args) {

Node root = new Node(10);

root.left = new Node(20);

root.right = new Node(30);

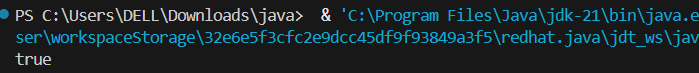
root.left.left = new Node(40);

root.left.right = new Node(60);

System.out.println(CheckBalanced(root));

}

}



Time Complexity: O(N)

1. Triplet sum in array

import java.util.Arrays;

public class ThreeSumArray {

static void CalculateSum(int[] arr, int sum) {

int n = arr.length;

Arrays.sort(arr);

int count = 0;

for (int i = 0; i < n - 2; i++) {

int left = i + 1;

int right = n - 1;

while (left < right) {

int calculate\_sum = arr[i] + arr[left] + arr[right];

if (calculate\_sum == sum) {

count++;

left++;

right--;

} else if (calculate\_sum < sum) {

left++;

} else {

right--;

}

}

}

if (count == 0) {

System.out.println(0);

} else {

System.out.println(count);

}

}

public static void main(String[] args) {

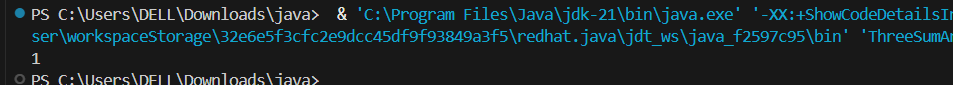
int[] arr = { 1, 4, 45, 6, 10, 8 };

int sum = 22;

CalculateSum(arr, sum);

}

}



Time Complexity: O(n^2)