DSA Practice-2 SRIKANTH M

22AD136

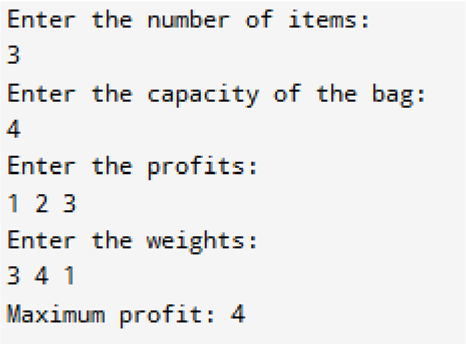
11/11/2024

1. ***0/1 Knapsack Problem:***

```
import java.util.*;

public class Knapsack {

  public static int KnapSack(int w, int n, int[] profit, int[] weight) {
    if (n == 0 || w == 0) {
      return 0;
    }
    if (weight[n - 1] > w) {
      return KnapSack(w, n - 1, profit, weight);
    } else {
      return Math.max(
        KnapSack(w, n - 1, profit, weight),
        profit[n - 1] + KnapSack(w - weight[n - 1], n - 1, profit, weight)
      );
    }
  }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
```

```
        System.out.println("Enter the number of items:");
        int N = sc.nextInt();

        System.out.println("Enter the capacity of the bag:");
        int W = sc.nextInt();

        int[] profit = new int[N];
        System.out.println("Enter the profits:");
        for (int i = 0; i < N; i++) {
            profit[i] = sc.nextInt();
        }

        int[] weight = new int[N];
        System.out.println("Enter the weights:");
        for (int i = 0; i < N; i++) {
            weight[i] = sc.nextInt();
        }

        System.out.println("Maximum profit: " + KnapSack(W, N, profit,
weight));
    }
}
```

Output:



```
Enter the number of items:
3
Enter the capacity of the bag:
4
Enter the profits:
1 2 3
Enter the weights:
3 4 1
Maximum profit: 4
```

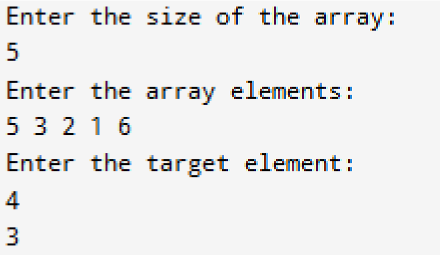
Time Complexity: O(2^n).

2. ***Floor in sorted array:***

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;
public class FloorSortedArray {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the size of the array:");
        int N = sc.nextInt();
        int[] nums = new int[N];
        System.out.println("Enter the array elements:");
        for(int i=0;i<N;i++) {
            nums[i]= sc.nextInt();
        }
        System.out.println("Enter the target element:");
        int x = sc.nextInt();
        ArrayList<Integer> arr = new ArrayList<Integer>();
        for(int i=0;i<nums.length;i++) {
            if(nums[i]<=x) {
                arr.add(nums[i]);
            }
        }
        int max = Collections.max(arr);
```

```
                    System.out.println(max);
          }

}
```

Output:



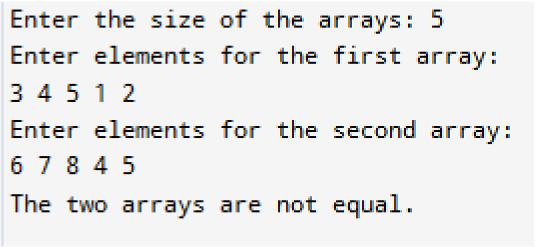
```
Enter the size of the array:
5
Enter the array elements:
5 3 2 1 6
Enter the target element:
4
3
```

Time Complexity: O(n).

3. *Check equal arrays:*

```
import java.util.Arrays;
import java.util.Scanner;

public class CompareArrays {

    public static boolean check(int[] arr1, int[] arr2) {
        if (arr1.length != arr2.length) {
            return false;
        }

        Arrays.sort(arr1);
        Arrays.sort(arr2);

        for (int i = 0; i < arr1.length; i++) {
            if (arr1[i] != arr2[i]) {
                return false;
            }
        }

        return true;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the size of the arrays: ");
        int n = sc.nextInt();

        int[] array1 = new int[n];
        int[] array2 = new int[n];

        System.out.println("Enter elements for the first array:");
        for (int i = 0; i < n; i++) {
            array1[i] = sc.nextInt();
        }

        System.out.println("Enter elements for the second array:");
        for (int i = 0; i < n; i++) {
            array2[i] = sc.nextInt();
        }

        if (check(array1, array2)) {
            System.out.println("The two arrays are equal.");
        } else {
            System.out.println("The two arrays are not equal.");
        }

        sc.close();
    }
}
```

Output:



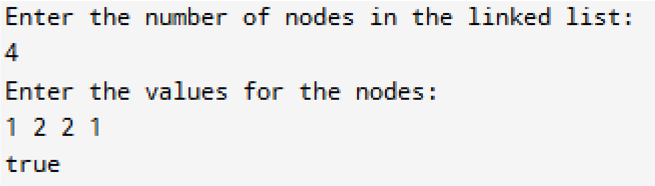
```
Enter the size of the arrays: 5
Enter elements for the first array:
3 4 5 1 2
Enter elements for the second array:
6 7 8 4 5
The two arrays are not equal.
```

Time Complexity: O(n log n).

4. Palindrome linked list:

```
import java.util.Scanner;

class Node {
    int data;
    Node next;

    Node(int data) {
        this.data = data;
        this.next = null;
    }
}

public class Main {
    boolean isPalindrome(Node head) {
        if (head == null || head.next == null) {
            return true;
        }

        Node slow = head;
        Node fast = head;

        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        Node prev = null;
        Node current = slow;

        while (current != null) {
            Node nextNode = current.next;
            current.next = prev;
            prev = current;
            current = nextNode;
        }

        Node left = head;
        Node right = prev;

        while (right != null) {
            if (left.data != right.data) {
                return false;
            }
            left = left.next;
            right = right.next;
        }

        return true;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the number of nodes in the linked
list:");
        int n = scanner.nextInt();

        if (n <= 0) {
            System.out.println("The linked list cannot be empty.");
            return;
        }

        System.out.println("Enter the values for the nodes:");
        Node head = new Node(scanner.nextInt());
        Node current = head;

        for (int i = 1; i < n; i++) {
            current.next = new Node(scanner.nextInt());
            current = current.next;
        }

        Main linkedList = new Main();
        boolean result = linkedList.isPalindrome(head);
        System.out.println(result);

        scanner.close();
    }
}
```

Output:



```
Enter the number of nodes in the linked list:
4
Enter the values for the nodes:
1 2 2 1
true
```

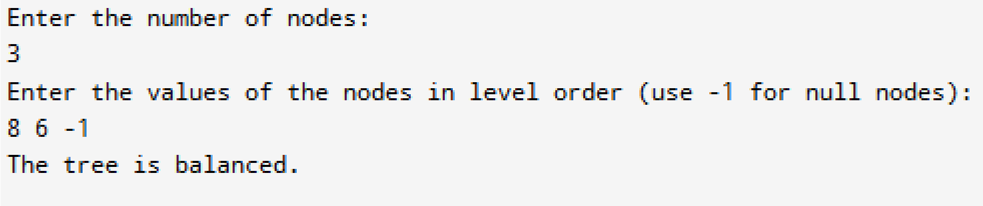
Time Complexity:O(n).

5. Balanced tree check:

```
import java.util.Scanner;

class Node {
    int data;
    Node left, right;

    Node(int data) {
        this.data = data;
        this.left = this.right = null;
    }
}
```

```
class Tree {
    boolean isBalanced = true;

    int solve(Node root) {
        if (root == null) return 0;
        int leftHeight = solve(root.left);
        int rightHeight = solve(root.right);
        if (Math.abs(leftHeight - rightHeight) > 1) {
            isBalanced = false;
        }
        return Math.max(leftHeight, rightHeight) + 1;
    }

    boolean isBalanced(Node root) {
        solve(root);
        return isBalanced;
    }
}

public class Main {
    public static Node insertLevelOrder(int[] arr, int i) {
        Node root = null;
        if (i < arr.length && arr[i] != -1) {
            root = new Node(arr[i]);
            root.left = insertLevelOrder(arr, 2 * i + 1);
            root.right = insertLevelOrder(arr, 2 * i + 2);
        }
        return root;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of nodes:");
        int n = sc.nextInt();
        System.out.println("Enter the values of the nodes in level order (use -1 for null nodes):");
        int[] values = new int[n];
        for (int i = 0; i < n; i++) {
            values[i] = sc.nextInt();
        }

        Node root = insertLevelOrder(values, 0);
        Tree tree = new Tree();
        boolean result = tree.isBalanced(root);
        System.out.println(result ? "The tree is balanced." : "The tree is not balanced.");

        sc.close();
    }
}
```

Output:



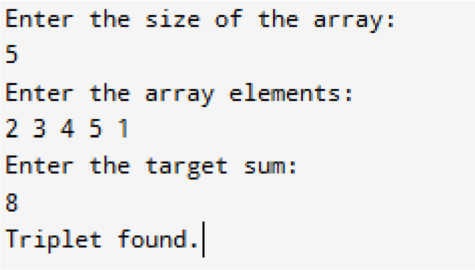
```
Enter the number of nodes:
3
Enter the values of the nodes in level order (use -1 for null nodes):
8 6 -1
The tree is balanced.
```

Time Complexity:O(n).

6. *Triplet sum in array:*

```
import java.util.Arrays;
import java.util.Scanner;

public class Main {
    public static boolean findTriplet(int[] arr, int n, int x) {
        Arrays.sort(arr);
        for (int i = 0; i < n - 2; i++) {
            int left = i + 1;
            int right = n - 1;
            while (left < right) {
                int sum = arr[i] + arr[left] + arr[right];
                if (sum == x) return true;
                else if (sum < x) left++;
                else right--;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the size of the array:");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the array elements:");
        for (int i = 0; i < n; i++) arr[i] = sc.nextInt();
        System.out.println("Enter the target sum:");
        int x = sc.nextInt();
        boolean result = findTriplet(arr, n, x);
        System.out.println(result ? "Triplet found." : "No triplet found.");
        sc.close();
    }
}
```

Output:



```
Enter the size of the array:
5
Enter the array elements:
2 3 4 5 1
Enter the target sum:
8
Triplet found.
```

Time Complexity:O(n^2).