0-1 knapsack problem

```
class Main {
    public static int knapsack(int capacity, int[] val, int[] wt) {
        int n = val.length;
        int[] dp = new int[capacity + 1];
        for (int i = 0; i < n; i++) {
            for (int j = capacity; j >= wt[i]; j--) {
                dp[j] = Math.max(dp[j], val[i] + dp[j - wt[i]]);
            }
        }
        return dp[capacity];
    }

    public static void main(String[] args) {
        System.out.println(knapsack(4, new int[]{1, 2, 3}, new int[]{4, 5, 1}));
        System.out.println(knapsack(3, new int[]{1, 2, 3}, new int[]{4, 5, 6}));
        System.out.println(knapsack(5, new int[]{10, 40, 30, 50}, new int[]{5, 4, 6, 3}));
    }
}
```

```
Output
3
0
50

=== Code Execution Successful ===
```

Time Complexity : O(n)

Floor in sorted array

```
class Main {
    public static int findFloor(int[] arr, int k) {
        int low = 0, high = arr.length - 1;
        int floorIndex = -1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (arr[mid] <= k) {
                floorIndex = mid;
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return floorIndex;
    }
    public static void main(String[] args) {
        int[] arr1 = {1, 2, 8, 10, 11, 12, 19};
        System.out.println( findFloor(arr1, 0));
        System.out.println( findFloor(arr1, 5));
        int[] arr2 = {1, 2, 8};
        System.out.println( findFloor(arr2, 1));
    }
}
```

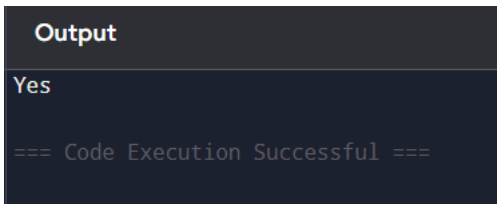
```
Output
-1
1
0

=== Code Execution Successful ===
```

Time Complexity : O(logn)

Check equal arrays

```
import java.util.Arrays;
class Main {
  public static boolean areEqual(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, 5, 5, 2};
      if (areEqual(arr1, arr2))
         System.out.println("Yes");
      else
         System.out.println("No");
   }
}
```



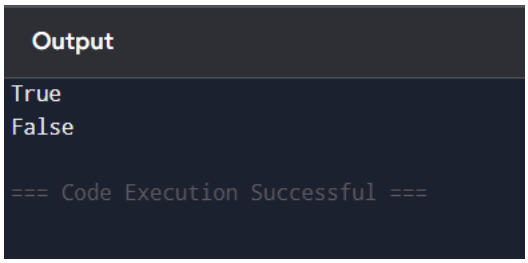
Time Complexity : O(n logn)

Palindrome linked list

```
class Main {
    Node head;
    static class Node {
        int data;
        Node next;
        Node(int d) {
            data = d;
            next = null;
        }
    }
    public boolean isPalindrome() {
        if (head == null || head.next == null) {
            return true;
        }
        Node slow = head, fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        Node secondHalf = reverseList(slow);
        Node firstHalf = head;
        while (secondHalf != null) {
            if (firstHalf.data != secondHalf.data) {
                return false;
            }
            firstHalf = firstHalf.next;
            secondHalf = secondHalf.next;
        }
        return true;
    }
```

```
private Node reverseList(Node head) {
    Node prev = null, curr = head, next = null;
    while (curr != null) {
        next = curr.next;
        curr.next = prev;
        prev = curr;
        curr = next;
    }
    return prev;
}
public void append(int data) {
    Node newNode = new Node(data);
    if (head == null) {
        head = newNode;
        return;
    }
    Node last = head;
    while (last.next != null) {
        last = last.next;
    }
    last.next = newNode;
}
public static void main(String[] args) {
    Main list = new Main();
    list.append(1);
    list.append(2);
    list.append(1);
    list.append(1);
    list.append(2);
    list.append(1);
    if (list.isPalindrome()) {
```

```
            System.out.println("True");
        } else {
            System.out.println("False");
        }
        Main list2 = new Main();
        list2.append(1);
        list2.append(2);
        list2.append(3);
        list2.append(4);
        if (list2.isPalindrome()) {
            System.out.println("True");
        } else {
            System.out.println("False");
        }
    }
}
```



Time Complexity : O(n)

Balanced tree check

```
class TreeNode {
   int data;
   TreeNode left, right;
   TreeNode(int data) {
      this.data = data;
      left = right = null;
   }
}
public class Main {
   public boolean isBalanced(TreeNode root) {
      return isBalancedHelper(root) != -1;
   }
   private int isBalancedHelper(TreeNode node) {
      if (node == null) {
         return 0;
      }
      int leftHeight = isBalancedHelper(node.left);
      int rightHeight = isBalancedHelper(node.right);
      if (leftHeight == -1 || rightHeight == -1 || Math.abs(leftHeight - rightHeight) > 1) {
         return -1;
      }
      return 1 + Math.max(leftHeight, rightHeight);
   }
   public static void main(String[] args) {
      Main tree = new Main();
      TreeNode root1 = new TreeNode(1);
      root1.left = new TreeNode(2);
      root1.left.right = new TreeNode(3);
      if (tree.isBalanced(root1)) {
         System.out.println("Tree 1 is balanced");
```

```
        } else {
            System.out.println("Tree 1 is not balanced");
        }
        TreeNode root2 = new TreeNode(10);
        root2.left = new TreeNode(20);
        root2.right = new TreeNode(30);
        root2.left.left = new TreeNode(40);
        root2.left.right = new TreeNode(60);
        if (tree.isBalanced(root2)) {
            System.out.println("Tree 2 is balanced");
        } else {
            System.out.println("Tree 2 is not balanced");
        }
    }
}
```

Output

```
Tree 1 is not balanced
Tree 2 is balanced

=== Code Execution Successful ===
```

Time Complexity : O(n)

Triplet sum in array

```
import java.util.Arrays;
public class Main {
   public static boolean findTriplet(int[] arr, int target) {
      int n = arr.length;
      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 == target) {
               return true;
            } else if (sum < target) {
               left++;
            } else {
               right--;
            }
         }
      }
      return false;
   }
   public static void main(String[] args) {
      int[] arr1 = {1, 4, 45, 6, 10, 8};
      int target1 = 13;
      System.out.println(findTriplet(arr1, target1));
      int[] arr2 = {1, 2, 4, 3, 6, 7};
      int target2 = 10;
      System.out.println(findTriplet(arr2, target2));
      int[] arr3 = {40, 20, 10, 3, 6, 7};
      int target3 = 24;
```

```
        System.out.println(findTriplet(arr3, target3));
    }
}
```

```
Output
true
true
false
```

Time Complexity : O(n log n)