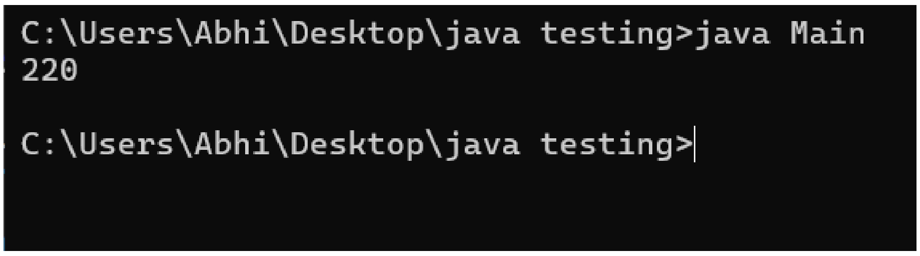
DSA PRACTICE SET-2

1) 0-1 knapsack problem

CODE

```
public class Main {
static int knapsack(int[] profit, int[] weight, int W, int curr, int n){
if(W<0 || curr==n){
return 0;
}
if(weight[curr]>W){
return knapsack(profit, weight, W, curr+1, n);
}else{
return Math.max(knapsack(profit, weight, W, curr+1, n), profit[curr]+knapsack(profit, weight, W-
weight[curr], curr+1, n));
}}
public static void main(String[] args) {
int[] profit = { 60, 100, 120 };
int[] weight = { 10, 20, 30 };
int W = 50;
int n = profit.length;
System.out.println(knapsack(profit, weight, W, 0, n));
}}
```

OUTPUT



```
C:\Users\Abhi\Desktop\java testing>java Main
220

C:\Users\Abhi\Desktop\java testing>
```

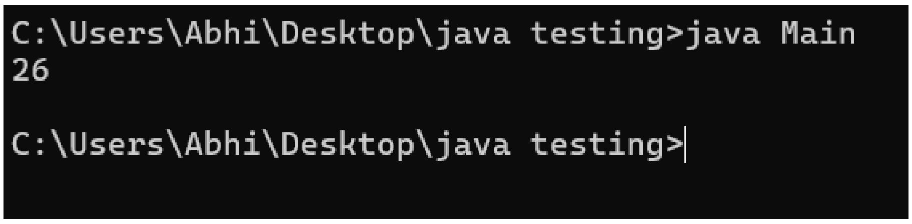
TIME COMPLEXITY: O(n^2)
SPACE COMPLEXITY: O(n)

2) Floor in sorted array

CODE

```
import java.util.*;
class Main{
public static void main(String[] args) {
int[] arr = {3,4,89,26,45};
int k = 30;
Arrays.sort(arr);
if(arr.length>0){
if(arr[0]>k){
System.out.println(-1);
}else{
for(int i=0; i<arr.length; i++){
if(arr[i]>k){
System.out.println(arr[i-1]);
break;
}}}
}else{
System.out.println(-1);}}}
```

<u>OUTPUT</u>



```
C:\Users\Abhi\Desktop\java testing>java Main
26

C:\Users\Abhi\Desktop\java testing>
```

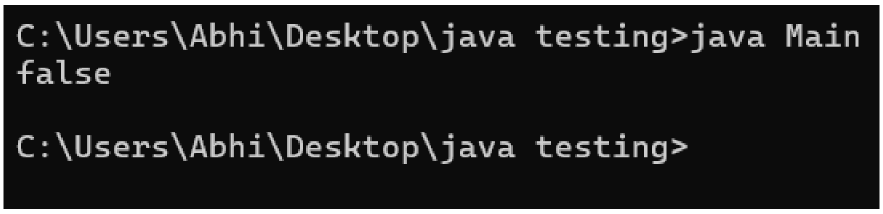
TIME COMPLEXITY: O(nlog(n))
SPACE COMPLEXITY: O(1)

3) Check equal arrays

<u>CODE</u>

```
import java.util.Arrays;
public class Main {
public static void main(String[] args) {
int[] arr1 = {1, 2, 5, 4, 0};
int[] arr2 = {2, 4, 5, 1};
Arrays.sort(arr1);
Arrays.sort(arr2);
System.out.println(Arrays.equals(arr1, arr2));
}}
```

<u>OUTPUT</u>



```
C:\Users\Abhi\Desktop\java testing>java Main
false

C:\Users\Abhi\Desktop\java testing>
```

TIME COMPLEXITY: O(nlog(n))

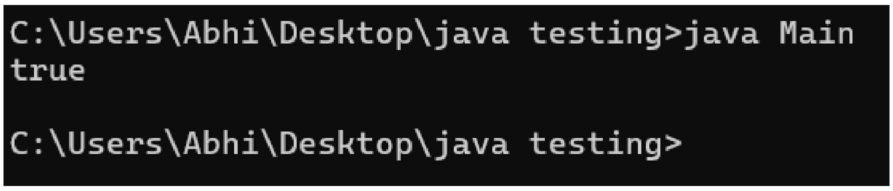
SPACE COMPLEXITY: O(1)

4) Palindrome linked list

CODE

```
class ListNode {
int val;
ListNode next;
public ListNode(int val) {
this.val = val;
}}
public class Main {
static boolean isPallindrome(ListNode head) {
if (head == null || head.next == null) {
return true;
}
ListNode slow = head;
ListNode fast = head;
while (fast != null && fast.next != null) {
slow = slow.next;
fast = fast.next.next;
}
ListNode prev = null;
ListNode curr = slow;
while (curr != null) {
ListNode nextTemp = curr.next;
curr.next = prev;
prev = curr;
curr = nextTemp;
}
ListNode firstHalf = head;
ListNode secondHalf = prev;
while (secondHalf != null) {
if (firstHalf.val != secondHalf.val) {
return false;
}
firstHalf = firstHalf.next;
secondHalf = secondHalf.next;
}
return true;
}
public static void main(String[] args) {
ListNode head = new ListNode(1);
head.next = new ListNode(2);
head.next.next = new ListNode(3);
head.next.next.next = new ListNode(2);
head.next.next.next.next = new ListNode(1);
boolean result = isPallindrome(head);
System.out.println(result);
}}
```

OUTPUT



```
C:\Users\Abhi\Desktop\java testing>java Main
true

C:\Users\Abhi\Desktop\java testing>
```

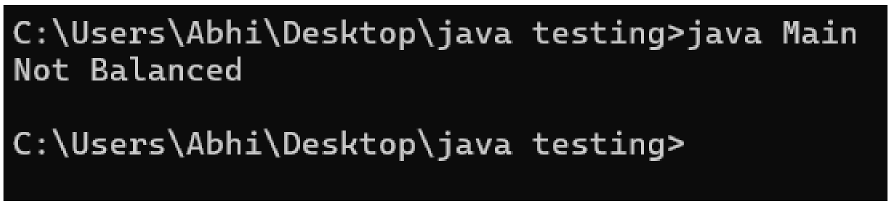
TIME COMPLEXITY: O(n)
SPACE COMPLEXITY: O(1)

5) Balanced tree check

CODE

```
class TreeNode{
int val;
TreeNode left = null, right = null;
public TreeNode(int val){
this.val = val;
}}
public class Main {
static int height(TreeNode root){
if(root==null){
return 0;
}
return 1+Math.max(height(root.right), height(root.left));
}
static boolean isBalanced(TreeNode root){
if(root==null){
return true;
}
int lh = height(root.left);
int rh = height(root.right);
return (Math.abs(lh-rh)<=1 && isBalanced(root.right) && isBalanced(root.left));
}
public static void main(String[] args) {
TreeNode root = new TreeNode(1);
root.left = new TreeNode(2);
root.right = new TreeNode(3);
root.left.left = new TreeNode(4);
root.left.right = new TreeNode(5);
root.left.left.left = new TreeNode(8);
if(isBalanced(root)){
System.out.println("Is Balanced");
}else{
System.out.println("Not Balanced");
}}}
```

OUTPUT



```
C:\Users\Abhi\Desktop\java testing>java Main
Not Balanced

C:\Users\Abhi\Desktop\java testing>
```

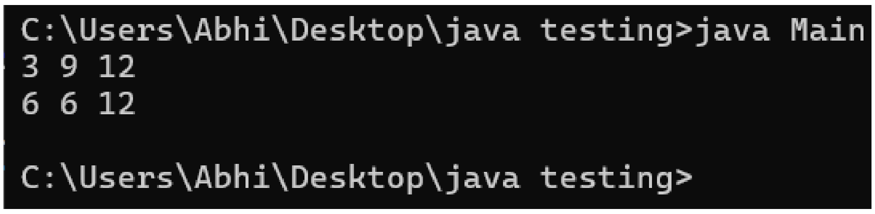
TIME COMPLEXITY: O(n^2)
SPACE COMPLEXITY: O(h);

6) Triplet sum in array

CODE

```
import java.util.*;
public class Main {
public static void main(String[] args) {
int[] arr = {12,12,12,12, 3, 4,4,4, 4, 1, 6,6,6,6,6, 9};
int target = 24;
Arrays.sort(arr);
List<List<Integer>> res =  new ArrayList<>();
for(int i=0; i<arr.length-1; i++){
if(i>0 && arr[i]!=arr[i-1]){
int start = i+1;
int end = arr.length-1;
while(start<end){
int temp = arr[i]+arr[start]+arr[end];
if(temp==target){
res.add(new ArrayList<>(Arrays.asList(arr[i], arr[start], arr[end])));
start+=1;
end-=1;
while (start < end && arr[start] == arr[start - 1]) start++;
while (start < end && arr[end] == arr[end + 1]) end--;
}else if(temp>target){
end-=1;
}else{
start+=1;
}}
}else{
continue;
}}
for(List<Integer> i: res){
for(int j: i){
System.out.print(j+" ");
}
System.out.println();
}}}
```

OUTPUT



```
C:\Users\Abhi\Desktop\java testing>java Main
3 9 12
6 6 12

C:\Users\Abhi\Desktop\java testing>
```

TIME COMPLEXITY: O(n^2)
SPACE COMPLEXITY: O(n)