0-1 knapsack problem:

```
class knapsack {

   static int knapSack(int W, int wt[], int val[], int n) {
      if (n == 0 || W == 0)
         return 0;

      if (wt[n - 1] > W)
         return knapSack(W, wt, val, n - 1);

      else
         return Math.max(knapSack(W, wt, val, n - 1),
          val[n - 1] + knapSack(W - wt[n-1], wt, val, n-1));
   }
}
```

Time Complexity = O(2^n)

Space Complexity = O(n)

Hidden Test Cases:

int[] profit = [60]

int[] weight = [10]

int W = 50

output = 0

int[] profit = [60]

int[] weight = [100]

int W = 50

output = 0

Floor in sorted array:

```
class floor_sorted {
    public int searchInsert(int[] nums, int target) {
        int left = 0;
        int right = nums.length - 1;
        while(left<=right){
          int mid = (right+left)/2;
          if(nums[i]==target){
           return i;
          }
          else if(nums[mid]>target){
           right-=1;
          }
          else{
           left+=1;
          }
        }
        return left;
    }
}
```

Time Complexity = O(log n)

Space Complexity = O(1)

Hidden Test Case:

nums = [10, 20, 30, 40, 50], target = 5

output = 0

nums = [1, 2, 3, 3, 4, 5], target = 3

output = 2

Check equal arrays

```
class equalArrays {
   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;
   }
}
```

Time Complexity = O(n log n)

Space Complexity = O(n)

Hidden Test Cases:

arr1 = [1, 2, 3], arr2 = [1, 2]

output = false

arr1 = [10, 20, 30, 30, 20, 10], arr2 = [30, 20, 10, 20, 30, 10]

output = true

Palindrome linked list:

```
class Solution {
    public boolean isPalindrome(ListNode head) {
        List<Integer> list = new ArrayList();
        while(head!=null){
            list.add(head.val);
            head = head.next;
        }

        int left = 0;
        int right = list.size()-1;

        while(left<right && list.get(left)==list.get(right)){
            left++;
            right--;
        }
        return left>=right;
    }
}
```

Time Complexity = O(N)

Space Complexity = O(N)

Hidden Test Cases:

Head = [1, 2, 2, 1]

Output = true

Head = [1, 2, 3, 4]

Output = false

Balanced tree check

```
class BalancedTree {
   public boolean isBalanced(TreeNode root) {
      if (root == null)  return true;
      if (Height(root) == -1)  return false;
      return true;
   }

   public int Height(TreeNode root) {
      if (root == null)  return 0;
      int leftHeight = Height(root.left);
      int rightHeight = Height(root.right);
      if (leftHeight == -1 || rightHeight == -1)  return -1;
      if (Math.abs(leftHeight - rightHeight) > 1)  return -1;
      return Math.max(leftHeight, rightHeight) + 1;
   }
}
```

Time Complexity = O(n)

Space Complexity = O(n)

Hidden Test Cases:

Input: root = [3,9,20,null,null,15,7]

Output: true

Input: root = [1,2,2,3,3,null,null,4,4]

Output: false

Triplet sum in array

```
class 3sum {
    public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> res = new ArrayList<>();
        Arrays.sort(nums);

        for (int i = 0; i < nums.length; i++) {
            if (i > 0 && nums[i] == nums[i-1]) {
                continue;
            }

            int j = i + 1;
            int k = nums.length - 1;

            while (j < k) {
                int total = nums[i] + nums[j] + nums[k];

                if (total > 0) {
                    k--;
                } else if (total < 0) {
                    j++;
                } else {
                    res.add(Arrays.asList(nums[i], nums[j], nums[k]));
                    j++;

                    while (nums[j] == nums[j-1] && j < k) {
                        j++;
                    }
                }
            }
```

```
        }
        return res;
    }
}
```

Time Complexity = O(N^2)

Space Complexity = O(N)

Hidden Test Cases:

int[] nums = [1, 2, 3, 4, 5]

output = []

int[] nums = [-1, 0, 1, 2, -1, -4]

output = [[-1, -1, 2], [-1, 0, 1]]