Lab 3 (Practice)
Loop, array

Q1. Loop

Write a program to print a diamond with a specified size using asterisks (*). The program receives an **odd positive** integer as the diagonal length of the diamond, which represents the number of asterisks (*) in the middle row. As diamonds are symmetric, the number of asterisks (*) in rows from top to bottom should be 1,3,5…5,3,1. Expected output is as follows:

Expected Outcomes:

Example 1

```
Enter the diagonal length:
5
  *
 ***
*****
 ***
  *
```

Example 2

```
Enter the diagonal length:
11
     *
    ***
   *****
  *******
 *********
***********
 *********
  *******
   *****
    ***
     *
```

Example 3

```
Enter the diagonal length:
4
Please enter an odd positive number!
-1
Please enter an odd positive number!
5
  *
 ***
*****
 ***
  *
```

*NOTE: The input is required to be **odd positive** integers. Validity check should be provided.*

Q2. Loop, array

A dice has 6 faces, each with one unique value: 1, 2, 3, 4, 5, and 6. The face value of a dice means the value showing upward.

a) Find the number of occurrences of the sum of face values of the two dices. Show all possible sums.

Expected Outcomes

```
Example
1 occurrence(s) of the sum 2
2 occurrence(s) of the sum 3
3 occurrence(s) of the sum 4
4 occurrence(s) of the sum 5
5 occurrence(s) of the sum 6
6 occurrence(s) of the sum 7
5 occurrence(s) of the sum 8
4 occurrence(s) of the sum 9
3 occurrence(s) of the sum 10
2 occurrence(s) of the sum 11
1 occurrence(s) of the sum 12
```

b) Sort the result by the number of occurrences and rearrange the output.

Expected Outcomes

```
Example
1 occurrence(s) of the sum 2
1 occurrence(s) of the sum 12
2 occurrence(s) of the sum 3
2 occurrence(s) of the sum 11
3 occurrence(s) of the sum 4
3 occurrence(s) of the sum 10
4 occurrence(s) of the sum 5
4 occurrence(s) of the sum 9
5 occurrence(s) of the sum 6
5 occurrence(s) of the sum 8
6 occurrence(s) of the sum 7
```

Q3. Loop, array

Consider the problem of coins change. Assume that we now have 4 different kinds of coins, 1 cent, 2 cents, 5 cents, 10 cents. We make changes with these coins for a given amount of money. Now we want to find the total amount of different ways of making changes for a given amount of money.

For example, for 6 cents, we can make changes with one 5-cent coin and one 1-cent coin, three 2-cent coins, two 2-cent coins and two 1-cent coins, one 2-cent coin and four 1-cent coins, or six 1-cent coins. So, there are five ways of making changes for 6 cents with the above coins. Note that we count that there is one way of making change for zero cent.

The input is an **integer** indicating the value of money for changes. The integer is smaller than 500. Output the total number of different ways of making changes.

Expected Outcomes:

Example 1
Enter the value of the money: 5 There are 4 different ways in total.
Example 2
Enter the value of the money: 216 There are 18975 different ways in total.
Example 3
Enter the value of the money: 0 There is only 1 way in total.

Q4. Loop, array

Now, consider a program to find whether a number exists in an array. The simple idea is to go through the whole array to check whether any elements equal to the number. However, if the array is sorted, we have a better approach to solve the problem called binary search.

For example:
Assume that we know have a sorted array in ascending order, and want to check whether a number **n** exists in the array or not.

1. We keep three indices **low**, **mid**, and **high**. Initially, we assign 0 to **low**, as well as the length of the array to **high**.
2. Then we calculate the average of the **low** and **high**, and store the result in **mid**. Then compare the element at the position of **mid** in the array with **n**.
    a. If they are equal, we get the answer.
    b. If **n** is larger, we can know the possible range of **n** must be the interval from **mid+1** to **high**, as the array is sorted, and we assign the value of **mid+1** to **low**.
    c. If **n** is smaller, the range must be the interval from **low** to **mid-1**, and we assign the value of **mid-1** to **high**.
3. We repeat step 2 until we find the position of **n**, or **high** is larger than **low**, which means **n** does not exist in the array.

Implement the binary search to find that whether the input number exists or not after sorting the input array as in the previous problem. Output the result as well as the number of iterations to find the result.

NOTE: you may compare the speed of binary search with simple brute force search.

Expected Outcomes:

Example 1

```
Enter the element in the array:
6
4
7
2
10
5
The sorted array is:
2, 4, 5, 6, 7, 10
Enter the number to search:
2
The position of the element is: 0
2 iteration used.
```

Example 2

```
Enter the element in the array:
6
4
7
2
10
5
The sorted array is:
2, 4, 5, 6, 7, 10
Enter the number to search:
11
The number does not exist in the array.
3 iteration used.
```