Answers **- Joy Bugalia**

1.

```
class Program
{
        0 references
        static void Main(string[] args)
        {
            int[] arr = { 1, 2, 3, 4, 5 };
            int[] rev_arr = Reverse(arr);
        Console.WriteLine("[{0}]", string.Join(", ", rev_arr));
        }
        1 reference
        static int[] Reverse(int[] arr)
        {
            int[] rev_arr = new int[arr.Length];
            for (int i = 0; i < arr.Length; i++)
            {
                rev_arr[i] = arr[arr.Length - i - 1];
            }
            return rev_arr;
        }
    }
}
```

Now, if we want to reverse the array in place, we can iterate through half the elements of the array and swap the elements at the opposite ends of the array.

```
0 references
class Program
{
        0 references
        static void Main()
        {
        int[] arr = { 1, 2, 3, 4, 5 };
        int arr_len = arr.Length;
        for (int i = 0; i < arr_len / 2; i++)
        {
            int temp = arr[i];
            arr[i] = arr[arr_len - i - 1];
            arr[arr_len - i - 1] = temp;
        }
        Console.WriteLine("[{0}]", string.Join(", ", arr));
    }
}
```

2. The length would be 2.41 units. Initially it would be 1.41 units from 0,0,0 to 1,1,0 and then 1 unit from 1,1,0 to 1,1,1.

3. ⌊log2N⌋+1

4.

```
class Program
{

    static Dictionary<int, int> findPairs(int[] arr, int N)
    {
        Dictionary<int, int> mypairs = new Dictionary<int, int>();
        int L = arr.Length;
        for (int i = 0; i < L; i++)
        {
            for (int j = i + 1; j < L; j++)
                if (arr[i] - arr[j] == N || arr[j] - arr[i] == N)
                    mypairs.Add(arr[i], arr[j]);
        }

        return mypairs;
    }
    static void Main(string[] args)
    {
        int[] arr = { 1, 2, 3, 4 };
        int N = 1;

        Dictionary<int, int> mypairs = findPairs(arr, N);

        if (mypairs.Any())
        {
            foreach (KeyValuePair<int, int> pair in mypairs)
            {
                Console.WriteLine("{0},{1}", pair.Key, pair.Value);
            }
        }
        else
        {
            Console.WriteLine("No Pairs found.");
        }
    }
   }
```

5. Here we need to find the max(prices[j]−prices[i]), for every i and j such that i > j. The time complexity for this algorithm will be O(n^2). To improve this we can maintain two variables minp and maxp corresponding to maximum and minimum profit for each rally. The time complexity for this be O(n). Example:

```
for i in range(len(prices)):
  if prices[i] < minp:
    minp = prices[i]
  elif prices[i] - minp > maxp:
    maxp = prices[i] – minp
return maxp
```

6.

```
namespace removeDuplicates
{
    class Node
    {
        static void Main(string[] args)
        {
            LinkedList<int> list = new LinkedList<int>();
            list.AddLast(3);
            list.AddLast(5);
            list.AddLast(10);
            list.AddLast(5);
            list.AddLast(3);

            removeDuplicates(list);
            foreach (int i in list)
            {
                Console.WriteLine(i);
            }
        }

        public static void removeDuplicates(LinkedList<int> list)
        {
            HashSet<int> set = new HashSet<int>();
            LinkedListNode<int> cur = list.First;
            while (cur != null)
            {
                if (set.Contains(cur.Value))
                {
                    LinkedListNode<int> next = cur.Next;
                    list.Remove(cur);
                    cur = next;
                }
                else
                {
                    set.Add(cur.Value);
                    cur = cur.Next;
                }
            }
        }
    }
}
```

7.

a) Stable Sort
b) Comparator function – (IComparer<T> in C#)
c) Depth–first search
d) Regular Expressions
e) Shunting-yard algorithm
f) Hash tables

g) Levenshtein distance