Q13:

```
class QuickSort
{
    int partition(int arr[], int low, int high)
    {
        int pivot = arr[high];
        int i = (low-1);
        for (int j=low; j<high; j++)
        {
            if (arr[j] <= pivot)
            {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        int temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;

        return i+1;
    }


    void sort(int arr[], int low, int high)
    {
        if (low < high)
        {
            int pi = partition(arr, low, high);
            sort(arr, low, pi-1);
            sort(arr, pi+1, high);
        }
    }

    static void printArray(int arr[])
    {
        int n = arr.length;
        for (int i=0; i<n; ++i)
            System.out.print(arr[i]+" ");
        System.out.println();
    }
```

```
    public static void main(String args[])
    {
        int arr[] = {10, 7, 8, 9, 1, 5};
        int n = arr.length;

        QuickSort ob = new QuickSort();
        ob.sort(arr, 0, n-1);

        System.out.println("sorted array");
        printArray(arr);
    }
}
```

Answer:

Because this case is not either best one or worst one, so it should be average case. So the T(N) is $T(n) = T(n/9) + T(9n/10) + \theta(n)$

Solution is : $\theta(n \log(n))$

Q17:

Expression	Dominant term(s)	O(...)
$5 + 0.001n^3 + 0.025n$	$0.001n^3$	$O(n^3)$
$500n + 100n^{1.5} + 50n \log_{10}n$	$100n^{1.5}$	$O(n^{1.5})$
$0.3n + 5n^{1.5} + 2.5 \cdot n^{1.75}$	$2.5n^{1.75}$	$O(n^{1.75})$
$n^2 \log_2 n + n(\log_2 n)^2$	$n^2 \log_2 n$	$O(n^2 \log_2 n)$
$n \log_3 n + n \log_2 n$	$n \log_3 n$, $n \log_2 n$	$O(n \log n)$
$3 \log_8 n + \log_2 \log_2 \log_2 n$	$3 \log_8 n$	$O(n \log n)$
$100n + 0.01n^2$	$0.01n^2$	$O(n^2)$
$0.01n + 100n^2$	$100n^2$	$O(n^2)$
$2n + n^{0.5} + 0.5n^{1.25}$	$0.5n^{1.25}$	$O(n^{1.25})$
$0.01n \log_2 n + n(\log_2 n)^2$	$n(\log_2 n)^2$	$O(n(\log_2 n)^2)$
$100n \log_3 n + n^3 + 100n$	n^3	$O(n^3)$
$0.003 \log_4 n + \log_2 \log_2 n$	$0.003 \log_4 n$	$O(\log n)$

Q28:

```
class BubbleSort
{
    void bubbleSort(int arr[])
    {
        int n = arr.length;
        for (int i = 0; i < n-1; i++)
            for (int j = 0; j < n-i-1; j++)
                if (arr[j] > arr[j+1])
                {
                    // swap arr[j+1] and arr[i]
                    int temp = arr[j];
```

```
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
    }

    /* Prints the array */
    void printArray(int arr[])
    {
        int n = arr.length;
        for (int i=0; i<n; ++i)
            System.out.print(arr[i] + " ");
        System.out.println();
    }

    // Driver method to test above
    public static void main(String args[])
    {
        BubbleSort ob = new BubbleSort();
        int arr[] = {64, 34, 25, 12, 22, 11, 90};
        ob.bubbleSort(arr);
        System.out.println("Sorted array");
        ob.printArray(arr);
    }
}
```

Answer:

Best Case Time Complexity: O(n). Best case occurs when array is already sorted. However, in this case it's not. So **Worst and Average Case Time Complexity: $O(n^2)$.**

Q29:

```
public class LinearSearch {

    public static int search(int arr[], int x)
    {
        int n = arr.length;
        for(int i = 0; i < n; i++)
        {
            if(arr[i] == x)
                return i;
        }
        return -1;
    }
```

```
    public static void main(String args[])
    {
        int arr[] = { 2, 3, 4, 10, 40 };
        int x = 10;

        int result = search(arr, x);
        if(result == -1)
            System.out.print("Element is not present in array");
        else
            System.out.print("Element is present at index " + result);
    }
}
```

Answer:

Best case is number which is needed be searched at the first index, the time complexity is O(1). In this case it's not, so time complexity is **O(n).**