Course ID: CS 501
Name: Mengmeng Xue Student ID: 19519
Hw4. 02/10
Description:
http://npu85.npu.edu/~henry/npu/classes/algorithm/geeksforgeeks/slide/exercise_geeksforgeeks.html
Q13 ==> What is the Big-O time complexity of QuickSort?
Q17 ==> Dominant term(s) and Big-O
Q28 ==> What is the Big-O Time Complexity Analysis of BubbleSort?
Q29 ==> What is the Big-O Time Complexity Analysis of Linear Search?

13. What is the Big-O time complexity of QuickSort?

```
// Java program for implementation of QuickSort
class QuickSort
{
    /* This function takes last element as pivot,
       places the pivot element at its correct
       position in sorted array, and places all
       smaller (smaller than pivot) to left of
       pivot and all greater elements to right
       of pivot */
    int partition(int arr[], int low, int high)
    {
        int pivot = arr[high];
        int i = (low-1); // index of smaller element
        for (int j=low; j<high; j++)
        {
            // If current element is smaller than or
            // equal to pivot
            if (arr[j] <= pivot)
            {
                i++;

                // swap arr[i] and arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // swap arr[i+1] and arr[high] (or pivot)
        int temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;

        return i+1;
    }


    /* The main function that implements QuickSort()
      arr[] --> Array to be sorted,
      low  --> Starting index,
```

```
      high  --> Ending index */
    void sort(int arr[], int low, int high)
    {
        if (low < high)
        {
            /* pi is partitioning index, arr[pi] is
              now at right place */
            int pi = partition(arr, low, high);

            // Recursively sort elements before
            // partition and after partition
            sort(arr, low, pi-1);
            sort(arr, pi+1, high);
        }
    }

    /* A utility function to print array of size n */
    static void printArray(int arr[])
    {
        int n = arr.length;
        for (int i=0; i<n; ++i)
            System.out.print(arr[i]+" ");
        System.out.println();
    }

    // Driver program
    // Output
    //    Sorted array:
    //    1 5 7 8 9 10
    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);
    }
}
/*This code is contributed by Rajat Mishra */
```

Answer:

Partition( ) – O() = nlog n;

Sort( ), O( ) = n;

Print ( ) , O( ) = n;

So, O (QuickSort) = nlog n + n+ n= O(nlog n).

17. Assume that each of the expressions below gives the processing time T(n) spent by an algorithm for solving a problem of size n. Select the dominant term(s) having

the steepest increase in n and specify the lowest Big-Oh complexity of each algorithm.

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

28. What is the Big-O Time Complexity Analysis of BubbleSort?

```
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:

In Bubble Sort, n-1 comparisons will be done in the 1st pass, n-2 in 2nd pass, n-3 in 3rd pass and so on. So the total number of comparisons will be, (n-1) + (n-2) + (n-3) + ... + 3 + 2 + 1. Hence the time complexity of Bubble Sort is O(n^2). (The best case time complexity will be O(n), it is when the list is already sorted. Not work on this case)

So, the time complexity of Bubble Sort is O(n^2)。

29. What is the Big-O Time Complexity Analysis of Linear Search?

```
// Java code for linearly search x in arr[]. If x
// is present then return its location, otherwise
// return -1

class GFG
{
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:

search ( ) , O() = n;

So, O (Linear Search) = O(n).