Assignment: 05 Name: Methew Guda
Due Date: 28-11-2015 A#: A00381751

Q1 (a): Quicksort works by taking a pivot on a given list. The worst case of quick may occur if we choose a pivot where all the elements are lower than the pivot or higher than the pivot; it, then recursively sorts the two sub lists in the same way by doing N – 1 comparisons, but create one partition that has N – 1 elements and the other will have no elements. This situation sometimes occurs if the list is already sorted in same order or reverse order.

Q1 (b): The size of the list N = 10000000

Quicksort Average case: $O(N \log_2 N) = (10000000 \log_2 10000000)$
$= 232534966.6$ microseconds
≈ 232.5349666 seconds
≈ 3.88 minutes

Quicksort Worst Case: $O(N^2) = (10000000^2) = 100000000000000$ microseconds
$= 100000000$ seconds
≈ 3.17 years

Q1 (c): There are two choices for first partitioning - the first or the last of the elements in the permutation. After pivot is chosen there are two choices for second partitioning - the first or the last of the remaining elements. After first and second pivot are chosen there are two choices for third partitioning, and so forth, until we reach final pivot, which has to be the one element left.

Q1 (d): Worst case partitioning: The worst-case behavior for quicksort occurs when the partitioning routine produces one partition with n - 1 elements and other partition with only 1 element. For a list size N, there are O(N) partition procedures where N = Last Element – First Element + 1, which is the number of the keys in the list.

Worst-Case:

Let T(N) be the worst-case time for quicksort on input size N. We have a recurrence

$T(N) = \max_{1 \le pivot \le n-1} (T(pivot) + T(N-pivot)) + O(N)$ --------- (1)

where pivot runs from 1 to N-1, since the partition produces two regions, each having size at least 1.

Now we guess that $T(N) \le CN^2$ for some constant C.

Substituting our guess in equation 1. We get

$$T(n) = \max_{1 \le pivot \le n-1} (C(pivot)^2) + C (N – (pivot)^2)) + O(N)$$
$$= C \max (pivot^2 + (N - pivot)^2) + O (N)$$

Since the second derivative of expression $pivot^2 + (N\text{-}pivot)^2$ with respect to pivot is positive. Therefore, expression gets a maximum over the range $1 \le pivot \le N -1$ at one of the endpoints.

This gives the bound $\max (pivot^2 + (N - pivot)^2)) 1 + (N -1)^2 = N^2 + 2(N -1)$.

Continuing with our bounding of T(N) we get

$$T(N) \le C [N^2 - 2(N-1)] + O (N)$$
$$= CN^2 – 2C(N-1) + O (N)$$

Since we can pick the constant so that the 2C(N -1) term dominates the O (N) term we have $T(N) \le CN^2$

Thus the worst-case running time of quick sort is $O(N^2)$.

Number of Comparisons:

Elements only in the same partition will be compared. If either first or last element is chosen as the pivot before any other element in the list, then that element will be compared to all of the elements of the list except itself.

Since there are (N - 0 + 1) elements in the list, and pivots are chosen randomly and independently, the probability that any one of them is chosen first is $\frac{1}{N-0+1}$.

So the average number of comparisons performed by quicksort is

(First Element chosen in the list) + (Last Element chosen in the list)

$$= \frac{2}{N-0+1} = \frac{2}{N+1}$$

Q1 (e).
```
using System;

        namespace QuickSort
        {
            class Program
            {
                private static int compCount = 0;
                private static int worstCase = 0;

                private static void Swap<T>(ref T lhs, ref T rhs)
                {
                    T temp;
                    temp = lhs;
                    lhs = rhs;
                    rhs = temp;
                }

                private static int Partition(int[] list, int first, int last)
                {
                    int PivotValue = list[first];
                    int PivotPoint = first;
                    for (int index = first + 1; index < last; index++)
                    {
                        if (list[index] < PivotValue)
                        {
                            compCount++;
                            worstCase++;
                            PivotPoint = PivotPoint + 1;
                            Swap(ref list[PivotPoint], ref list[index]);
                        }
                    }

                    Swap(ref list[first], ref list[PivotPoint]);
                    return PivotPoint;
                }
                public static void QuickSort(int[] list, int first, int last)
                {
                    if (first < last)
                    {
                        compCount++;
                        int pivot = Partition(list, first, last);
                        QuickSort(list, first, pivot - 1);
                        QuickSort(list, pivot + 1, last);
                    }
                }

                static void Main(string[] args)
                {
                    int ListSize = 50;
                    Random rand = new Random();

                    int[] list = new  int[ListSize];

                    int first = list[1];
                    int last = ListSize;

                    for (int i = 0; i < ListSize; i++)
                    {
                        list[i] = rand.Next(1, 99);
                    }

                    Console.Write("Unsorted List: ");

                    for (int i = 0; i < ListSize; i++)
                    {
                        Console.Write("{0} ", list[i]);
                    }

                    QuickSort(list, first, last);

                    Console.Write("\n  Sorted List: ");

                    for (int i = 0; i < ListSize; i++)
                    {
                        Console.Write("{0} ", list[i]);
                    }

                    Console.WriteLine("\n  Comparisons: {0}", compCount);
                    Console.WriteLine("\n   Worst Case: {0}", worstCase);

                    Console.ReadKey();
                }
            }
        }
```

Q2 (a). 264 has 9 bits because 256 < 264 < 512 => $2^8 < 264 < 2^9$.

- Option Smaller: 3 passes with 3-bit pieces
- Option Larger: 9 passes with 1-bit pieces

Q2 (b). For a key is a string of 40 characters, it is 320-bit

- Option Smaller: 10 passes with 32-bit pieces
- Option Larger: 32 passes with 10-bit pieces

Q2 (c). If we use arrays for the buckets, these will need to be extremely large arrays. Using arrays means that we will need 10N additional space if the keys are numeric, 26N additional space if the keys are alphabetic, and even more if the keys are alphanumeric or if case matters in alphabetic characters. If we use arrays, we also have the time to copy the records to the buckets in the distribution step and from the buckets back into the original list in the coalescing step.