1. Introduction

This assignment have been completed into the following enviroment and the system specification

1. GCC Compiler, HP – 2000, RAM – 2GB
4. Processor - Intel® Core™ i3-3110M CPU @ 2.40GHz × 4
5. Operating System (OS) – Ubuntu 16.04 (64 bit)

2.1. Counting Sort

2.1. The Pseudocode of the Counting Sort is

Ans:

```
countingSort (a[],  b[], n, k)
1. c[k];
// Initialize array C
for  i=0 to k
        c[i] = 0;
// Store number of element equal to i index
for  i= 0 to (n-1)
        c[a[i]]++;

// Store cummulative frequency of each element
for i = 1 to k
        c[i] = c[i] + c[i-1];
// Sort the elements in linear time
for (n-1)>= i>=0
        b[c[a[i]]-1] = a[i];
        c[a[i]] = c[a[i]] -1;
```

2.2. Graphical representation of the Time Complexity corresponding to different input size (datasets) ‘n’ with range key (0-60,000) is as below.

Ans:

1. The execution time of the couting sort **increases linearly** in time with the increase of the size of input dataset.

2. The time follows the behavior of the O(n) where ‘N’ is the size of the dataset.

3. It is because, in each of the loop, there is constant amout of time is needed to complete comparison and assignment operation. From the last loop the over all time complexity is O(N).

Size of Dataset	Elapsed Time (Counting Sort)	Elapsed Time (Quick Sort)
100000	0.003	0.02
500000	0.013	0.1259
1000000	0.027	0.2497

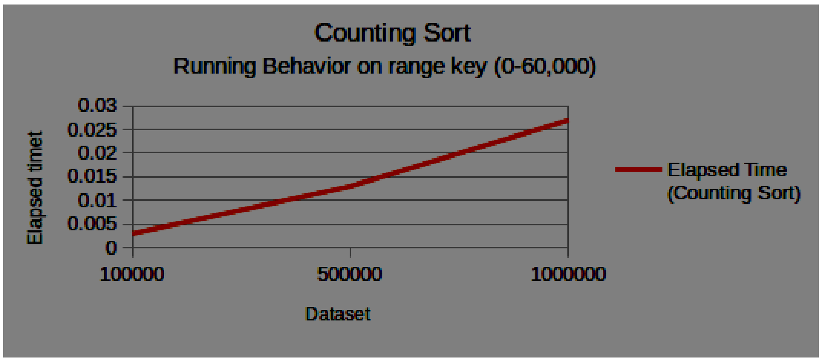


Figure 2.1. Performance of counting sort O(N) in time vs Dataset of size N.

3. Quick Sort performance on the dataset used for analysis of the counting Sort Technique

3.1. Pseudocode of the quick Sort :

Ans:

```
partition(array , l, h)
1.      pivot = array[i], i =l;
2.      for j = (l+1) to (h-1)
3.              if  array[j]<= pivot
4.                      i = i+1,
5.                      swap(array[i], array[j])
8.      swap(array[i], array[l]);
10.     return (i);

Algorithm quickSort(array, l, h)
//
1.      if (l<h)
2.              pivot = partition(array, l, h);
3.              quiclSort(array, l, pivot)
4.              quickSort(array, pivot+1, h);
```

3.2. Graphical representation and the behavior of the quick sort

Ans:

Size of Dataset	Elapsed Time (Counting Sort)	Elapsed Time (Quick Sort)
100000	0.003	0.02
500000	0.013	0.1259
1000000	0.027	0.2497

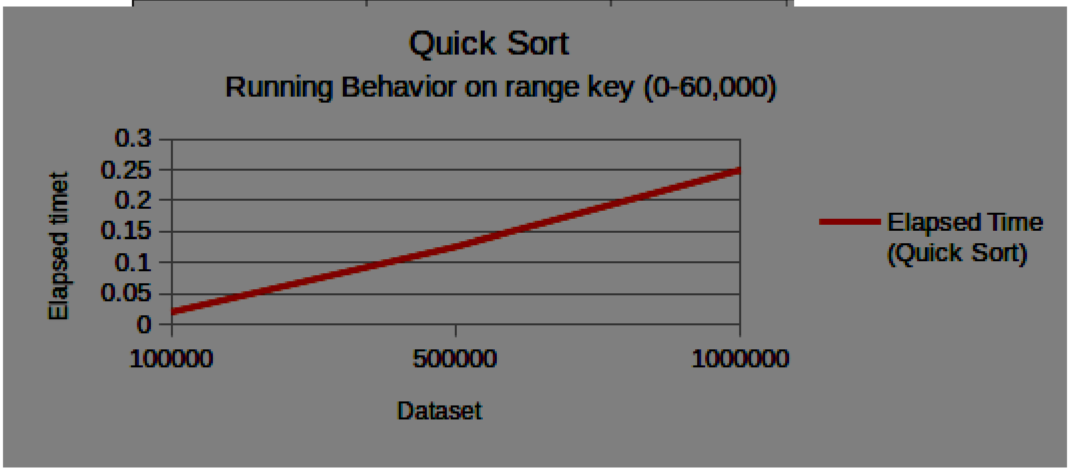
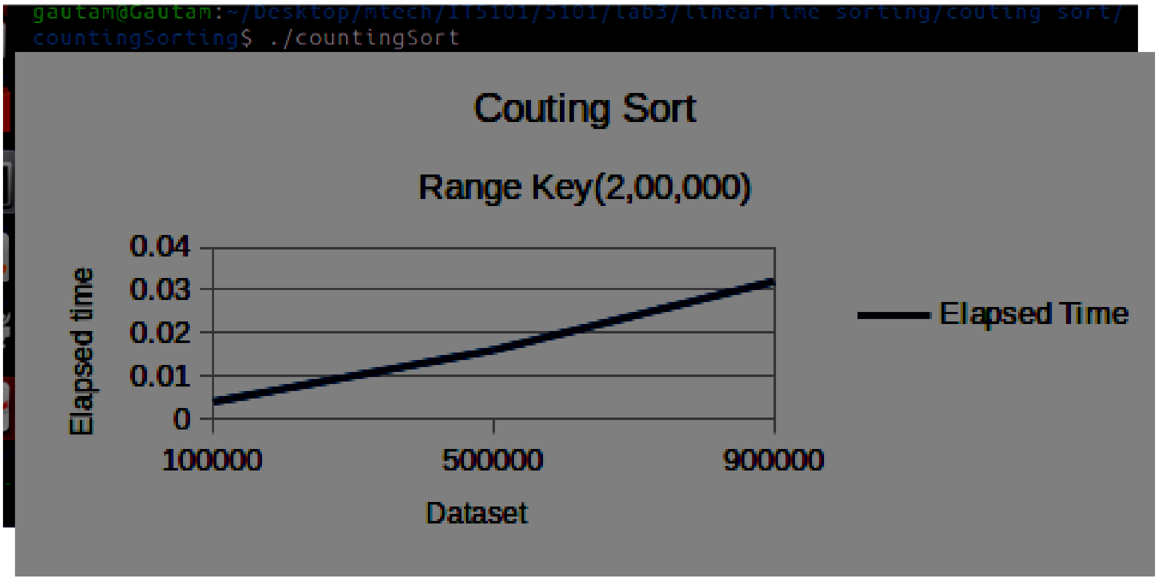


Figure 3.1. Performance of quick sort on the dataset ove range keys (0-60,000).

1. The quick sort takes much greater time to sort the same dataset.
2. To sort dataset of size of 10^5 the counting sort took only 0.003 pulses while the quick sort took 0.02 pulses.
3. Hence quick sort is NOT better to sort such type of datasets while counting sort outperforms well.
4. T(N) = O(N log N) for quick sort

4. Performance of Counting sort on dataset having keys from the range (0 - 2,00,000)

Size of Dataset	Elapsed Time
100000	0.004
500000	0.016
9000000	0.032



1. The counting sort outperforms well and runs in linear time.
2. While, in case of much repeation of the keys in the dataset, the counting sort takes time less by an episilon time factor, in case of the less repeation of keys in the dataset, the algorithm takes episilon factor more time in linear.

5. Redix Sort Technique

CountingSort(a[], exp, n)
1. c[10] = {0};
2. b[n];

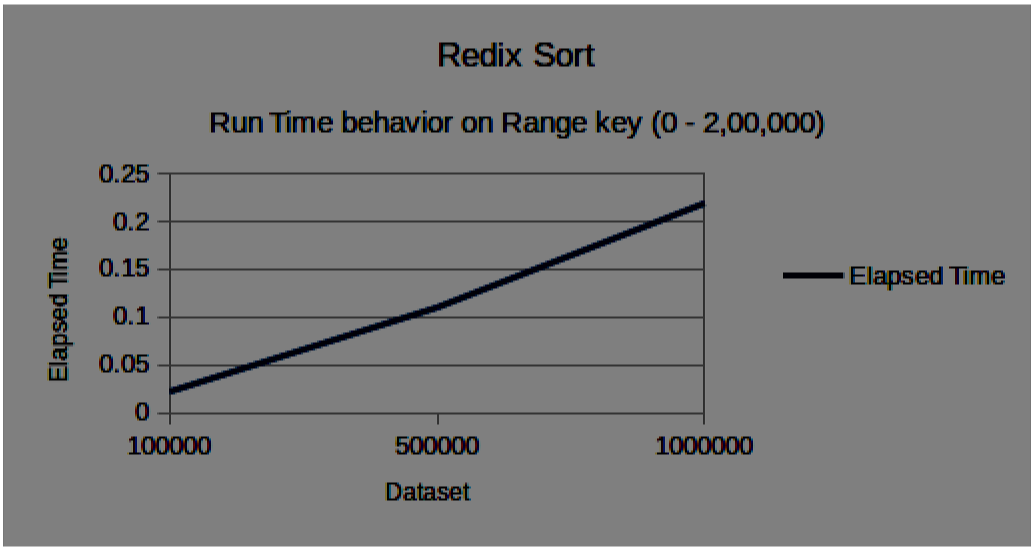
3. for i=0 to (n-1)
4.   c[(a[i]/exp)%10] = c[(a[i]/exp)%10]+1;

5. for i=1 to (10-1)
6.   c[i] = c[i]+c[i-1];

7. for i=(n-1) to i>=0
8.   b[c[(a[i]/exp)%10]-1] = a[i];
9.   c[(a[i]/exp)%10] = c[(a[i]/exp)%10] - 1;

10. for i=0 to (n-1)
11.   a[i] = b[i];

RedixSort(int *a, int size)
1. m = max(a,size);
2. for (int exp = 1; m / exp > 0; exp *= 10)
3.   countingSort(a,exp,size);



5. 1. Time complexity of redix sort

1. Running time of counting sort = O(n+k)
2. Let each number is of b bit, and can be partitioned into r bit pieces
3. Then, k = 2^r
4. Then, T(n,b) = Theta(b/r (n + 2r))

Here,
The redix sort runs into the same behavior
In each iteration the counting sort takes O(n) amout of time
So, over all time complexity is = O(n+k), where k = Number of iterations.

References:

1. Cormen, Thomas H., et al. *Introduction to algorithms, (30, 37)*. MIT press, 2010.
2. Horowitz, Ellis. "Sartaj Sahni.., Fundamentals of Computer Algorithms, (145, 154)." (1998).
3. https://en.wikipedia.org/wiki/counting_sort
4. https://en.wikipedia.org/wiki/redix_sort