Homework 2

Part 1:

1. $T(n) = T(n-3) + 3\lg n$

Our guess: $T(n) = O(n\lg n)$

Show $T(n) \le cn\lg n$ for $c > 0$

$$T(n) \le c(n-3)\lg(n-3) + 3\lg n$$
$$= c[n\lg(n-3) - 3\lg(n-3)] + 3\lg n$$

$$= c[n\lg n - n\lg 3 - 3\lg n + 3\lg 3] + 3\lg n$$
$$= cn\lg n - cn\lg 3 - 3c\lg n + 3c\lg 3 + 3\lg n$$
$$= cn\lg n - \lg 3(cn - 3c) - \lg n(3c - 3)$$
$$\le cn\lg n$$

for $c > 0$

2. $$T(n) = 4T\left(\frac{n}{3}\right) + n$$

Our guess: $T(n) = O(n^{\log_3 4})$

Show $T(n) \leq cn^{\log_3 4}$ for $c > 0$.

$$T(n) \leq 4c\left(\frac{n}{3}\right)^{\log_3 4} + n$$
$$= 4c\left(\frac{n^{\log_3 4}}{3^{\log_3 4}}\right) + n$$
$$= 4c\,\frac{n^{\log_3 4}}{4} + n$$
$$= cn^{\log_3 4} + n$$
$$\not\leq cn^{\log_3 4}$$

We guess $T(n) \leq cn^{\log_3 4} - dn$ again,

$$T(n) \leq 4\left[c\left(\frac{n}{3}\right)^{\log_3 4} - \frac{dn}{3}\right] + n$$
$$= 4\left[\frac{cn^{\log_3 4}}{4} - \frac{dn}{3}\right] + n$$
$$= cn^{\log_3 4} - \frac{4}{3}dn + n$$
$$\leq cn^{\log_3 4} - dn$$

for $d \geq 3$ and $c > 0$

3. $T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + T\left(\frac{n}{8}\right) + n$

Our guess: $T(n) \leq O(n)$

Show $T(n) \leq cn$ for $c > 0$

$$T(n) \leq \frac{cn}{2} + \frac{cn}{4} + \frac{cn}{8} + n$$
$$= c\left[\frac{n}{2} + \frac{n}{4} + \frac{n}{8}\right] + n$$
$$= c\left[\frac{4n + 2n + n}{8}\right] + n$$
$$= c\left[\frac{7}{8}n\right] + n$$
$$\leq cn$$

for $c \geq 8$

4. $T(n) = 4T\left(\frac{n}{2}\right) + n^2$

Our guess: $T(n) = O(n^2)$

Show $T(n) \leq cn^2$ for $c > 0$

$$T(n) \leq 4c\left(\frac{n}{2}\right)^2 + n^2$$
$$= 4c\frac{n^2}{4} + n^2$$
$$= cn^2 + n^2$$
$$\nleq cn^2$$

We guess $T(n) \leq n^2 \lg n$ again,

$$T(n) \leq 4c\left(\frac{n^2}{2}\right) \lg\left(\frac{n}{2}\right) + n^2$$
$$= 4c\frac{n^2}{4} \lg\left(\frac{n}{2}\right) + n^2$$
$$= cn^2 \lg n - cn^2 \lg 2 + n^2$$
$$= cn^2 \lg n - cn^2 + n^2$$
$$\leq cn^2$$

for $c > 0$

Homework Assignment 2

Abstract

Radix sort is a non-comparative sorting algorithm. It avoids comparison by creating and distributing elements into buckets according to their radix. Because radix sort is not comparison based, it is not bounded by Ω(nlogn) for running time and hence radix sort can perform in linear time. The main goal of this report is to analyse the running time of radix sort using insertion sort and counting sort for different combinations of array sizes ranging from 2500 to 7500000 and length of the random strings ranging from 25 to 45. We'll be running a test at least 5 times to compute the average running time of the sorting algorithm for different input size and string length and compare their performance.

Introduction:

Radix sort works by sorting each digit from least significant digit to most significant digit. We'll be using an array of random strings up to a given length with characters between "a" and "z" and sort a given array of strings according to the character at position d. We'll be using the ASCII representation of the character to sort the elements and radix sort would sort by the leftmost position to the right most position. To do this, we use insertion sort and counting sort as a subroutine to sort the digits for each position value. The runtime of radix sort is Θ(m(n+k)), given n m-digit numbers in which each digit can take on up to k possible values. For stable sort, it takes Θ(n+k) time.

We'll present and analyze the results of the radix sort using insertion sort as well as counting sort in the next section and see how well they perform.

Results and Evaluation:

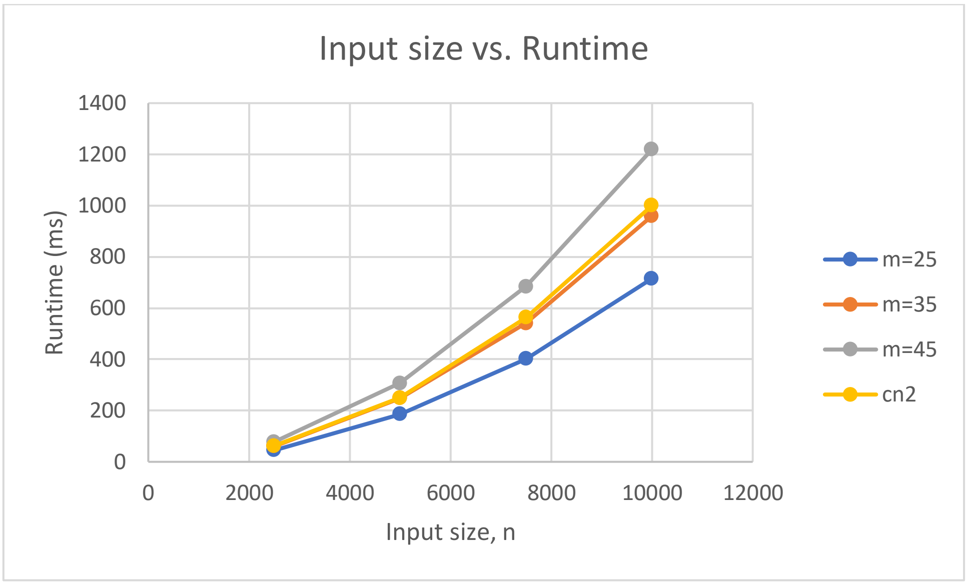
The average runtime of radix sort using insertion sort for different combination of input sizes and length of random strings is described in the table below.

Radix Sort using Insertion Sort Runtime in ms (Average Summary)

n	m=25	m=35	m=45
2500	44.8	60.3	77.8
5000	185.8	248.2	307.4
7500	401.4	541.4	683.8
10000	715.4	959.6	1219.2

The insertion sort has a time complexity of O(n^2) and hence as the input size increases the running time of the algorithm increases exponentially. The radix sort itself has a time complexity of Θ(m(n+k)). Hence, the increase in the length of the random strings also increases the runtime of the algorithm. It takes Θ(n+k) time for stable sort and as the worst-case time complexity for insertion sort is itself O(n^2), there is an exponential increase in the running time of the algorithm with increase in the input size.

Below is the Input Size vs. Runtime comparison for different input sizes and length of the random strings.



From the above results, we can see that the runtime of the radix sort is comparable to the time complexity of the insertion sort (O(n^2)) as insertion sort is being used to sort the digits for each position value.

The average runtime of radix sorts using counting sort for different combination of input sizes and length of random strings is described in the table below.

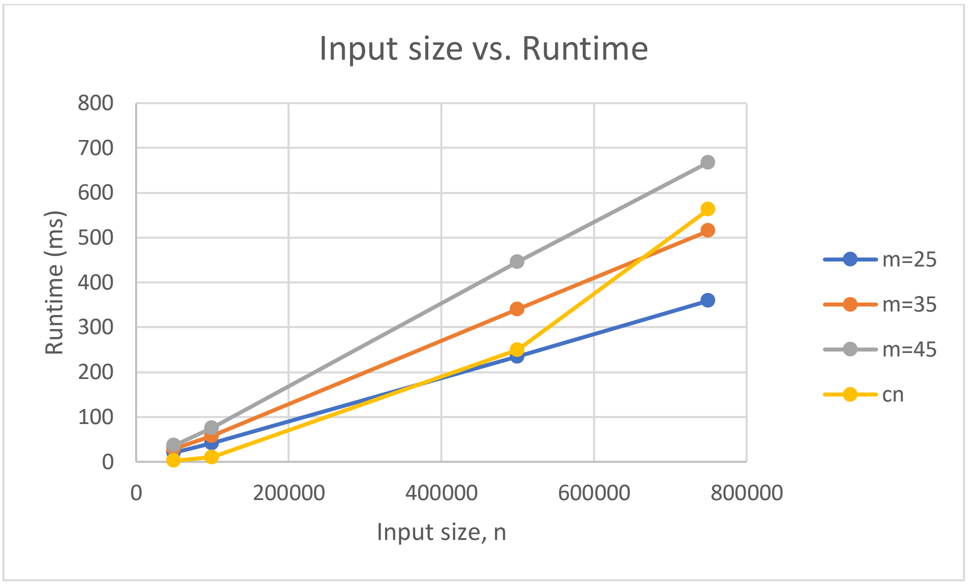
Radix Sort using Counting Sort Runtime in ms (Average Summary)

n	m=25	m=35	m=45
50000	20.6	29	36.4
100000	41.2	57.8	76
500000	234.75	340.6	446.2
750000	359.2	514.8	667.8

The radix sort using counting sort performs better than the radix sort using insertion sort as the time complexity of counting sort is computationally less compared to the insertion sort. The counting sort has a time complexity of O(n) and hence as the input size increases the running time of the algorithm does not increase exponentially.

The analysis of the running time depends on the stable sort used as the intermediate sorting algorithm. When each digit is in the range 0 to k-1 (so that it can take on k possible values), and k is not too large, counting sort is the obvious choice. Each pass over n d-digit numbers then take time Θ(n+k). There are d passes, and so the total time for radix sort is Θ(d(n+k)). When d is constant and k = O(n), we can make radix sort run in linear time. The running time of counting sort is $\Theta(n+k)$ which is $\Theta(n)$ if k=O(n).

Below is the Input Size vs. Runtime comparison for different input sizes and length of the random strings.



From the above results, we can see that the runtime of the radix sort is linear since counting sort is being used as the intermediate sorting algorithm.

Conclusion:

We’re successfully able to test the runtime performance of radix sort algorithm for different input sizes and string lengths using insertion and counting sort and compare the efficiency of these algorithms. From the above results and analysis, we can conclude that the radix sort using counting sort is the more efficient algorithm since it runs in linear time. For small input sizes, the radix sort using insertion sort performs well but as the size of the input increases, it becomes less efficient.