classmate

Date ______
Page ______

CS-590 HW 2 - Recurrences and Sorting.

* Name: DAKSH BHUVA

* CWID: 10475468

* 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 some $c > 0$

→ $\lg n$ is monotonically increasing for $n > 0$ ..... (i)

⇒ Base case; $n = 1 \Rightarrow n\lg n = 0$

• Inductive Step; $T(n-3) = (n-3)\lg(n-3)$

$$\begin{aligned}\therefore T(n) &= T(n-3) + 3\lg n\\ &\le c(n-3)\lg(n-3) + 3\lg n\\ &\le cn\lg(n-3) - 3c\lg(n-3) + 3\lg n\\ &\le cn\lg n - 3c\lg n + 3\lg n\end{aligned}$$

↳ [from (i)]

→ for some $c > 0$, ignoring lower order terms,

$\therefore T(n) \le cn\lg n$

→ Hence, our guess is proved.

② $T(n) = 4T(\frac{n}{3}) + n$ | Our guess: $T(n) = O(n^{\log_3 4})$

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

⇒ Base Case; $n = 1 \Rightarrow 1^{\log_3 4} = 1$

• Inductive Step; $T(n/3) = (n/3)^{\log_3 4}$

$\therefore T(n) = 4T(n/3) + n$

$\leq 4c\,(n/3)^{\log_3 4} + n$

$\leq 4c\left(\frac{n^{\log_3 4}}{3^{\log_3 4}}\right) + n$

$\leq 4c\left(\frac{n^{\log_3 4}}{4^{\log_3 3}}\right) + n$

$\leq$ ~~[illegible]~~ $4c\left(\frac{n^{\log_3 4}}{4}\right) + n$

$\leq c n^{\log_3 4} + n$

~~[illegible]~~

$\therefore T(n) \leq c n^{\log_3 4}$; for some $c > 0$

→ Hence, our guess is proved.

③ $T(n) = T(n/2) + T(n/4) + T(n/8) + n$

Our Guess : $T(n) = O(n)$

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

⇒ Base Case; $n = 1 \Rightarrow 1$

• Inductive Step; $T(n/2) = n/2$
$T(n/4) = n/4$
$T(n/8) = n/8$

$\therefore T(n) = T(n/2) + T(n/4) + T(n/8) + n$

$\leq c(n/2) + c(n/4) + c(n/8) + n$

$\therefore T(n) \leq (4cn + 2cn + cn + 8n)/8$

$$\leq \frac{(7c+8)n}{8}$$

$$\therefore T(n) \leq \frac{(7c)n}{8}$$

$\therefore T(n) \leq cn$; for some $c > 0$

→ Hence, our guess is proved

4) $T(n) = 4T(n/2) + n^2$ | Our guess: $T(n) = O(n^2)$

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

⇒ Base case; $n = 1 \Rightarrow 1$

• Inductive step; $T(n/2) = n^2/2$

$\therefore T(n) = 4T(n/2) + n^2$

$\leq 4(cn^2/2) + n^2$

[illegible]

$\therefore T(n) \leq 2cn^2 + n^2 \nleq n^2$

⇒ Our initial guess failed, hence taking our new guess as,

$T(n) = O(n^2 \lg n)$

$\therefore T(n) = 4T(n/2) + n^2$

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

$\leq cn^2 \lg\left(\frac{n}{2}\right) + n^2$

$\leq cn^2 \lg n - cn^2 \lg 2 + n^2$

$\leq cn^2 \lg n - cn^2 + n^2$

$\therefore T(n) \leq cn^2 \lg n - n^2(c-1)$

$\therefore T(n) \leq cn^2 \lg n$; for some $c > 0$

→ Hence, our guess is proved

Assignment 2: Recurrences and Sorting

Abstract:

I have implemented and analyzed the time-complexity of Radix Sort using Insertion Sort and Radix Sort using Counting Sort. By comparing the run-times for different values ‘n’ and ‘m’, I have documented the results and plotted the graphs of runtime for a better understanding of these algorithm's running times.

Result:

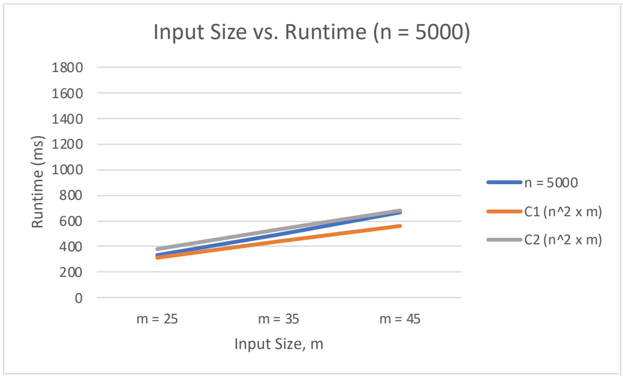
A. Radix Sort using Insertion Sort

Table 1: Radix Sort with Insertion Sort Runtime in ms

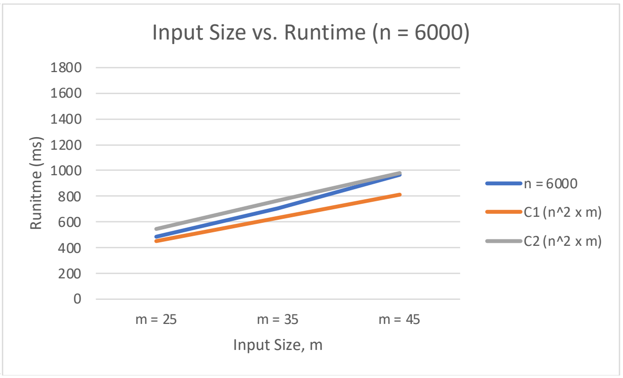
	n = 5000	n = 6000	n = 7000	n = 8000
m = 25	335.8	482.2	652.2	858.4
m = 35	495.2	707.6	956.2	1259.8
m = 45	669.6	962.8	1301.2	1699.4

$C1 = 5 \times 10^{-7}$

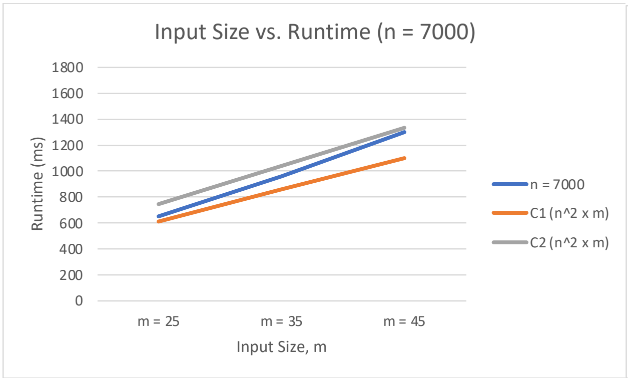
$C2 = 6.06 \times 10^{-7}$



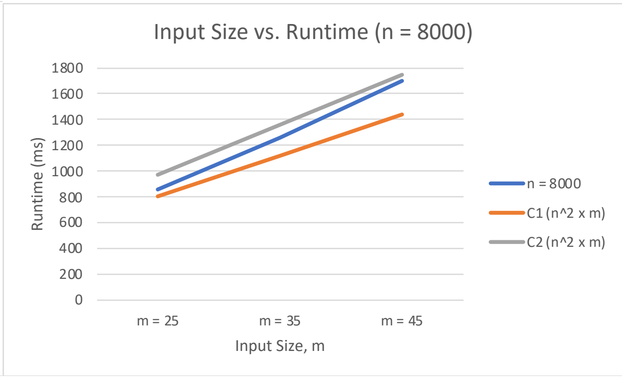
Graph 1.1



Graph 1.2



Graph 1.3



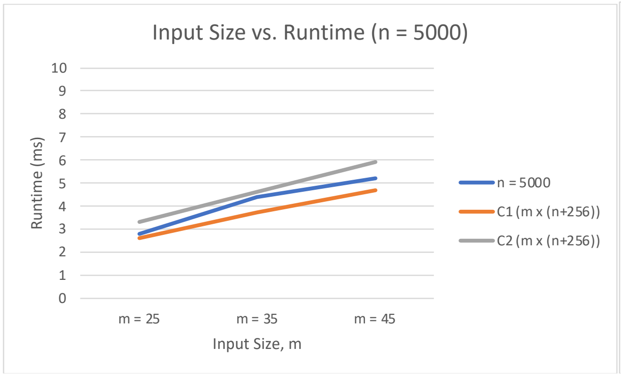
Graph 1.4

B. Radix Sort using Counting Sort

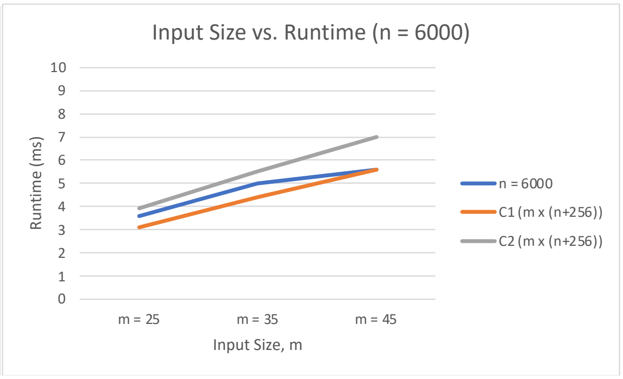
Table 2: Radix Sort with Counting Sort Runtime in ms

	n = 5000	n = 6000	n = 7000	n = 8000
m = 25	2.8	3.6	4.8	5
m = 35	4.4	5	5.8	6.6
m = 45	5.2	5.6	7.4	8.2

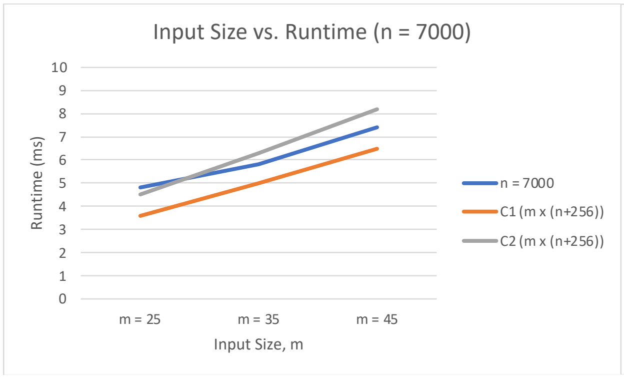
C1 = 2 x 10^{-5}
C2 = 2.5 x 10^{-5}



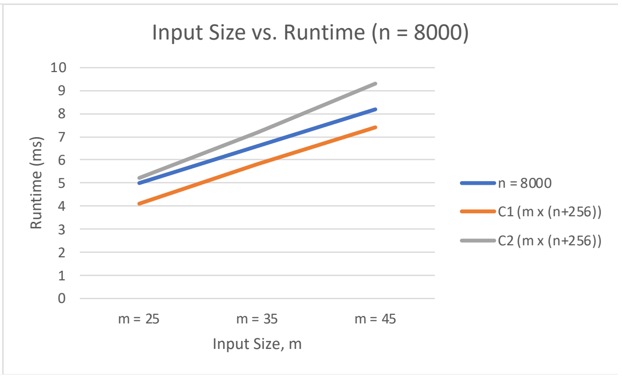
Graph 2.1



Graph 2.2



Graph 2.3



Graph 2.4

Discussion:

a. **Insertion Sort Algorithm for an array of strings**
   - In this algorithm, I created a new function "*modified_string_compare*" which takes two strings as parameters and compares them at a specific digit 'd' and returns the output.
   - I swap the respective elements using the output of the above function.
   - Insertion Sort is quadratic time sorting algorithm.
   - It has a time-complexity of O (n^2 x m).

b. **Radix Sort using Insertion Sort**
   - In this algorithm, I implemented Radix Sort by using Insertion Sort starting from string with longest length and decrementing the length by 1 on each iteration.
   - The current string length in Radix Sort is used parameter 'd' for the insertion sort.
   - It has a time-complexity of O (n^2 x m).
   - As the value of n increases, there is exponential growth in the running time of the algorithm and can be inferred from Table 1 and Graphs 1.1 – 1.4
   - 'n' has more impact on the running time as compared to 'm' and can be deducted from Table 1 and Graphs 1.1 – 1.4

c. **Counting Sort Algorithm for an array of strings**
   - In this algorithm, I have used '*int count[256]*' which stores the count of each character according to the character at position 'd'.
   - Each count is added to previous one to get the index position of the string for the newly created output array.
   - Counting Sort is linear time sorting algorithm.
   - Generally Counting Sort has a time-complexity of O (n + k). In our case it is O (n + 256), but the values of n >= 5000 and hence time-complexity becomes O (n + 256) ≅ O (n).

d. **Radix Sort using Counting Sort**
   - In this algorithm, I implemented Radix Sort by using Counting Sort starting from string with longest length and decrementing the length by 1 on each iteration.
   - The current string length in Radix Sort is used parameter 'd' for the Counting Sort.
   - It has time-complexity of O (m x (n + 256)), as Counting Sort has a time-complexity of (n + 256) and Radix Sort runs 'm' times.
   - As the value of n increases, there is linear growth in the running time of the algorithm and can be inferred from Table 2 and Graphs 2.1 – 2.4
   - 'n' has more impact on the running time as compared to 'm' and can be deducted from Table 1 and Graphs 2.1 – 2.4

Conclusion:

From the above results I conclude that Radix Sort using Counting Sort work better as compared to Radix Sort using Insertion Sort. Radix Sort with Counting Sort reduces the running time to great extent against Radix Sort with Insertion Sort.