STEVENS INSTITUTE OF TECHNOLOGY

Computer Science (Master's)

CS 590 A - ALGORITHMS
ASSIGNMENT 2

Submission to:
Professor In Suk Jang

Submission By:
Chetan Goyal
CWID: 20005334

Date: October 22, 2021

PART 1
RECURRENCES

1. T(n) = T(n - 3) + 3 logn.

   To prove: T(n) <= cn logn for some constant c > 0 and n > n1 for some n1 > 0

   Given: Subtract off a lower order term to make the substitution proof work or adjust the guess in case the initial substitution fails

   Our guess: T(n) = O(n logn)

   Proof:

   Since the number of inputs has to be 1 or greater, for T(1), T(1) will be equal to T(-2) + 3 log(1). And T(-2) will be undefined, so n has to be greater than 3.

   n > 3 - (a)

   Now, as per our guess, T(n - 3) = O((n - 3) log(n - 3)).
   Hence, T(n - 3) <= c(n - 3) log(n-3)
   Therefore, from T(n) = T(n - 3) + 3logn, we can say,

   T(n) <= c(n - 3) log(n-3) + 3 logn
   = cn log(n - 3) + 3 (-c log(n - 3) + logn)
   = cn log(n - 3) + 3 (log((n - 3) ^ (-c)) + logn)
   = cn log(n - 3) + 3 (log(n / ((n - 3) ^ c))) - (b)

   From (a)- since n > 3, so, n - 3 > 0. Now for n > 4, n - 3 > 1.

   Hence, for some c which is big enough, n / ((n - 3) ^ c) < 1.

   If we check by inputting n = 5, we get that c > 2 will give us the above desired result. So let's take c > 2 as base.

   Hence c - 2 > 0. Let's call c - 2 as our new constant c.

   Therefore 3 log(n / ((n - 3) ^ c)) will be negative.

So we can say for equation (b) that:
cn log(n - 3) + 3 (log(n / ((n - 3) ^ c))) <= cn log(n - 3)

Therefore,
Continuing on equation (b):

T(n) <= cn log(n - 3)

And for any n >= 4, log(n - 3) <= log(n)

So, we can say that T(n) <= cn log(n), for some c > 0 and n > 4.

Hence proved.

2. T(n) = 4T(n / 3) + n

   To prove: T(n) <= c (n ^ log3(4)) for some constant c > 0 and n > n1 for some n1 > 0

   Given: Subtract off a lower order term to make the substitution proof work or adjust the guess in case the initial substitution fails

   Our guess: T(n) = O(n ^ log3(4))

   Proof:

   Based on our guess, T(n) = O(n ^ log3(4)), implying that T(n / 3) <= c(n / 3) ^ log3(4) for some constant c > 0 and n > n1 for some n1 > 0

   So in T(n) = 4T(n / 3) + n, we can say that,

   T(n) <= 4c (n / 3) ^ log(3(4)) + n
   = (4c / (3 ^ log3(4))) * (n ^ log3(4)) + n

   Now we know that log3(4) > 1 which will imply that n ^ log3(4) > n.
   Since we are given that we can subtract off a lower order term to make the substitution work, we will eliminate n.
   Which will make the result as:

T(n) <= (4c / (3 ^ log3(4))) * (n ^ log3(4))

Now, we since c > 0, we can say that (4 / 3 ^ log3(4)) * c > 0. Also it will be smaller than the initial value of c.
Therefore let's call, (4 / 3 ^ log3(4)) * c, as our new constant c.
So, T(n) <= c (n log3(4)) for some c > 0.

Hence proved.

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

   To prove: T(n) <= cn for some constance c > 0 and n > 0

   Given: Subtract off a lower order term to make the substitution proof work or adjust the guess in case the initial substitution fails

   Our guess: T(n) = O(n)

   Proof:

   Based on our guess, T(n / 2) = O(n) implying that T(n / 2) <= c * (n / 2);
   T(n / 4) <= c * (n / 4); T(n / 8) <= c * (n / 8) for some c > 0 and n > 0

   So in T(n) = T(n / 2) + T(n / 4) + T(n / 8) + n, we can say that,

   T(n) <= (cn / 2) + (cn / 4) + (cn / 8) + n
   = (4cn / 8) + (2cn / 8) + (cn / 8) + (8n / 8)
   = ((7c + 8) / 8) * n

   Now, since c > 0, then (7c + 8) / 8 > 0. Hence we can replace (7c + 8) / 8 with another constant and lets call it c only.
   Therefore,

   T(n / 2) <= cn for some constant c > 0

   Hence proved.

4. T(n) = 4T(n / 2) + (n ^ 2)

To prove: T(n) <= c * (n ^ 2)

Given: Subtract off a lower order term to make the substitution proof work or adjust the guess in case the initial substitution fails

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

Proof:

Based on our guess, T(n / 2) = O((n / 2) ^ 2) implying that T(n / 2) <= c * (n ^ 2) for some constant c > 0

So in T(n) = 4T(n / 2) + (n ^ 2), we can say that,

T(n) <= 4c ((n / 2) ^ 2) + (n ^ 2)
= 4c ((n ^ 2) / 4) + (n ^ 2)
= c(n ^ 2) + (n ^ 2)
= (c + 1) * (n ^ 2)

Since c > 0; c + 1 will also be greater than 0.
So, we replace c + 1 with another constant c and let's call it c only.

Therefore,

T(n) <= c * (n ^ 2)

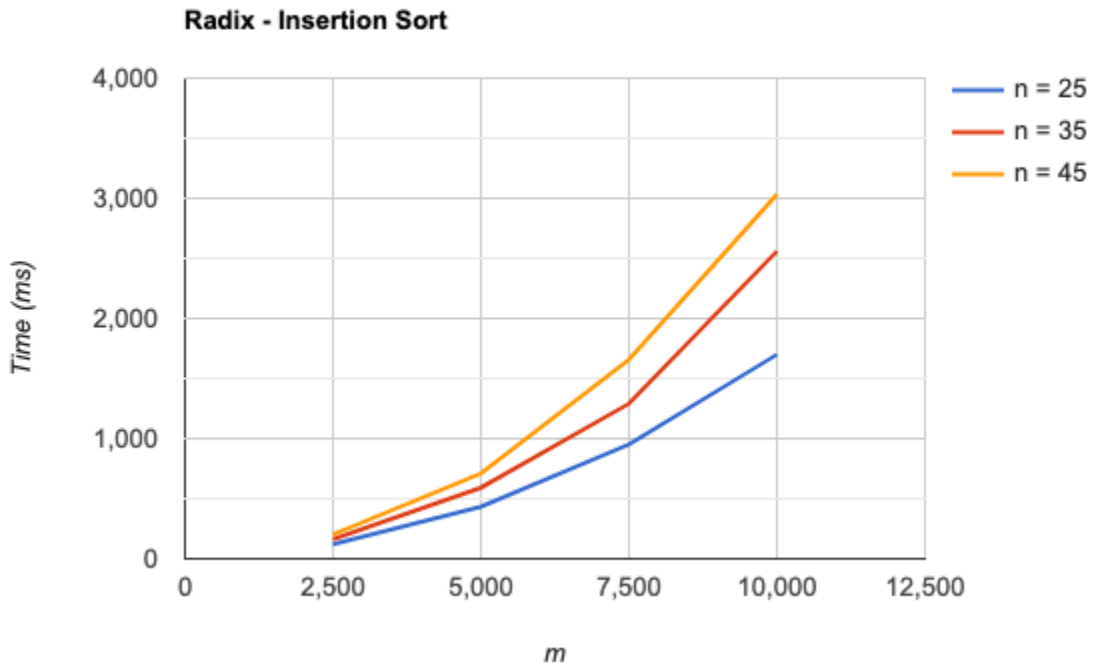
Hence proved.

*** PART 2 CONTINUED BELOW ***

PART 2
RADIX SORT ON STRINGS

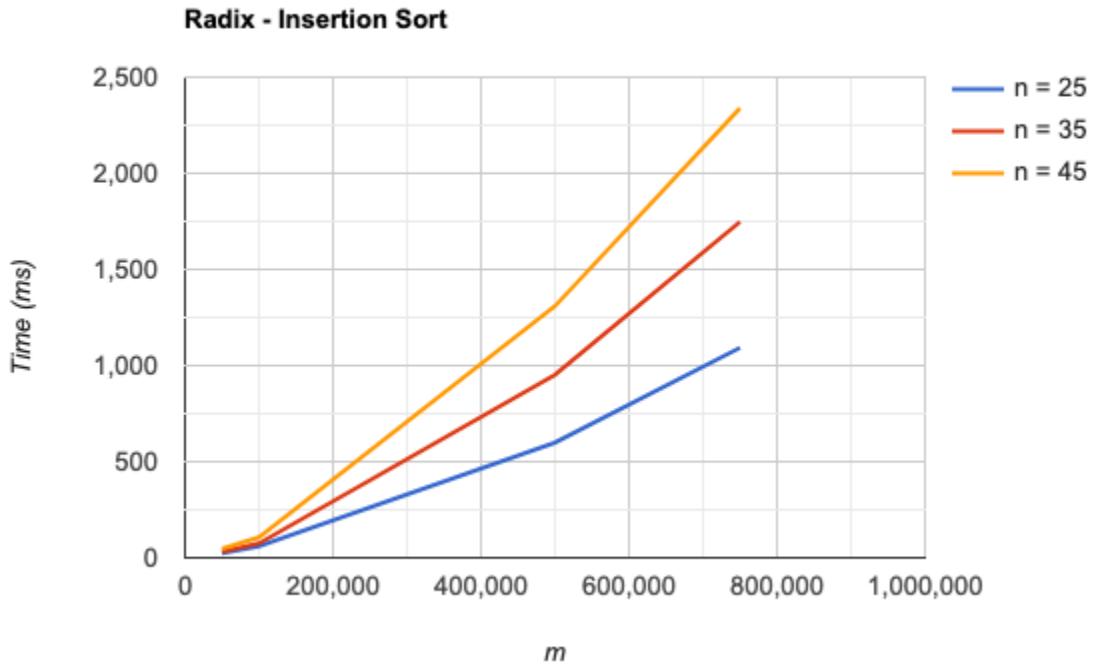
Radix Using Insertion Sort

	n	25	35	45
m	Time (ms)			
2500		120	163	203
5000		433	592	711
7500		951	1291	1659
10000		1700	2561	3034



Radix Using Counting Sort

	n	25	35	45
m	Time (ms)			
50000		24	32	47
100000		60	75	107
500000		599	952	1309
750000		1093	1748	2339



ANALYSIS

From the tabular data and the graph, we can clearly see that Radix Sort when used with Insertion sort, takes a lot more time when compared to the Radix Sort implemented via Counting Sort.

When we observe closely the insertion sort graph, we can see that at our given values for m, there is a sudden increment in the slope and as m increases the slope also increases. If we had taken more values of m, we could have seen clearly that the graph is rising quadratically. This is consistent with Insertion sort's running time of O(n^2).

However, as we increase the value of n, we can see that the lines go above and the time increases more rapidly. This is because Radix sort itself runs d times, wherein d here is m. So Radix sort through insertion sort here is **O(m*(n^2))**, which can be further generalised to **O(d*(n^2))**.

Now if we observe the counting sort graph closely, we can see that our given values for m, the graph increases linearly only. This is consistent with the Counting Sort's running time which is O(n). However, when we increase the value of m, the running time increases and it is multiple of the value of m. This is clearly in accordance with radix sort running m times, and so the counting sort will also run m times. Hence, the total running time will be **O(m*n)**. This can be generalised to **O(d*n)**, wherein d is the length of the array which needs to be iterated for radix sort.

We can see that our results are consistent with the theory we have learnt, as mentioned above.