Stevens Institute of Technology

Department of Computer Science

CS590: ALGORITHMS

Prof. Iraklis Tsekourakis

Homework Assignment 2

Submitted by: **Malav Shastri**

CWID**: 10456244**

Part 1 (20 Points)

Recurrences: Solve the following recurrences using the substitution method. Subtract off a lower order term to make the substitution proof work or adjust the guess in case the substitution fails.

1. T(n) = T(n-3) + 3 log n
   Our guess:-T(n)$\leq cnlog\ n$ (where there exist c>0).

   Using Induction method,
   Base Case: n = 1, nlg n=1.0=0
   Inductive step:
   T(n)=$T(n-3)+3\lg n$

   $\leq c(n-3)\ \lg(n-3)+3\lg(n)$

   $\leq cnlg(n-3)-3clg(n-3)+3\lg(n)$

   $\leq cnlg(n-3)+3\lg(n)$ (Ignoring the term -3clg(n-3) wouldn't effect the inequality)

   $\leq cnlg(n)+3\lg(n)$ (lg n is monotonically increasing)

   $\leq cnlg(n)+3nlg(n)$

   $\leq (c+3)nlg(n)$

   = $c'nlg(n)$

   Where c' is equal to c+3, hence $c'>3>0$,

   So our initial guess was correct. T(n) = $O(nlg\ n)$.

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

   Our guess:-T(n)$\leq cn^{log_3^4}$ (where there exist c>0).

   Using Induction method,

   Base Case: n = 1, $n^{log_3^4} = 1$

   Inductive Step:

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

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

   $\leq 4cn^{log_3^4} + n$ (Because $\log_3 4 > 1$ and n>n/3)

   Which is $\neq cn^{log_3^4}$

   So we subtract the lower order terms,

   Improved guess:- T(n)$\leq cn^{log_3^4} - dn$

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

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

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

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

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

For $d \geq 3$,

   $\leq cn^{log_3^4} - 4n + n$

   $\leq cn^{log_3^4} - 3n$

   Therefore, our improved guess T(n)$\leq \boldsymbol{cn^{log_3^4} - dn}$ is true.

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

Our guess:-T(n)$\leq cn$ (where there exist c>0).

Using Induction method,
Base Case: n = 1, $T(n) = 1$
Inductive Step:

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + T\left(\frac{n}{8}\right) + n$$
$$\leq \frac{cn}{2} + \frac{cn}{4} + \frac{cn}{8} + n$$
$$\leq c\left(\frac{n}{2} + \frac{n}{4} + \frac{n}{8}\right) + n$$
$$\leq c\left(\frac{7n}{8}\right) + n$$
$$\leq n(\frac{7c}{8} + 1)$$

Let $\frac{7c}{8} + 1 = c_1$,

$$\leq nc_1$$

So our initial guess was correct. T(n) = $O(n)$.

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

Our guess:-T(n)$\leq cn^2$ (where there exist c>0).

Using Induction method,
Base Case: n = 1, $T(n) = 1$
Inductive Step:

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

In this case we usually subtract the lower order terms, but here there are no lower order terms thus, we make a new guess using recurrence tree where there will be total (log n) levels and sum at every level would be n^2. So total complexity would be in order of n^2logn so,

Improved guess:- T(n)$\leq cn^2 \log n$

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

Therefore, our improved guess T(n)$\leq \boldsymbol{cn^2 \log n}$ is true.

Please find part 2 in next page

Part 2 Radix-sort on strings (80 points)

Introduction:

Given problem is all about sorting an array of characters (Strings) using radix sort. We are supposed to perform it using two different stable sorting algorithm, 1) Insertion Sort. 2) Counting Sort. Counting sort is a non-comparing linear sorting algorithm. Compared to insertion sort which is in it's worst case can take an order of n^2. Talking about Radix sort it is also a stable sorting algorithm, in which we sort the data point one by according to its position, more generally starting from the least significant digits position i.e. from left to right.

The strings which are being generated has different random length but we provide max value for that length. String lengths can't go beyond that max value. Here while performing radix sort with any of the above mentioned stable sorting algorithms we may find that for particular dth iteration from total iterations equal to max value (max would be the least significant one) value at (d-1)th index would not be present, the problem also demands to consider the ascii value at that missing position as zero and so consider that particular string for that particular iteration lesser then the string which has a character at that position.

Functions used:

- **insertion_sort_digit():** insertion_sort_digit() sorts the input string according to provided dth position. Here when we say dth position that means we need to check at (d-1)th index value. So insertion_sort_digit will take multidimensional array as an input along with the length array consisting lengths of all the strings. Normal insertion logic will be performed but for comparing two strings it will call the function string_compare_digit() which will compare two strings based on the provided value of d. This insertion sort function is having complexity of $O(n^2)$ in average and worst cases.
- **string_compare_digit():** string_compare_digit() will take two strings, might be of two different lengths will compare based on the character at (d-1)th index and would return a binary value according to whichever is greater than the other one. This function also takes care of

the cases where actually the (d-1)th position is missing i.e. the string length is lesser then the provided value of position d.

- **radix_sort_is ():** radix_sort_is() is performing radix sort on the string but the internal stable sorting algorithm which is being used is insertion sort. Radix sort operates according to the position of digits. Starting from least significant digit to most or in our case from left most position to righter most position. We can also say that it runs the loop for provided maximum length of a randomly generated string. Lets say it runs total d times then the complexity of radix_sort_is() would be $O(d.n^2)$. Here n^2 is the average case and worst case complexity of insertion sort. Of course for smaller values of d we can ignore it and can say complexity is $O(n^2)$. For best case we can say it would take O(n) but the possibility of that case in our randomly generated strings problem is none.
- **counting_sort_digit ():** counting_sort_digit() also sorts the input string according to provided dth position. Here again when we say dth position that means we need to check at (d-1)th index value. So similar to insertion_sort_digit will take multidimensional array as an input along with the length array consisting lengths of all the strings it will also take an another output array in input to store the intermediate results. Counting sort algorithm is a non-comparing sorting algorithm which runs in a linear time. An auxiliary counter array is also used to store the number of counts a particular data point occurs.

  In the problem author has suggested to use the fix 256 sized counter array. Which represents a prime assumption we take while using. Total Complexity: O(k)+O(n)+O(k)+O(n) = O(n) if k=O(n)
- **radix_sort_cs():** radix_sort_is() is performing radix sort on the string but the internal stable sorting algorithm which is being used is counting sort. Radix sort operates according to the position of digits. Starting from least significant digit to most or in our case from left most position to righter most position. While Radix sort uses counting sort as an internal algorithm it is a linear sorting algorithm with complexity of O(dn) here again for smaller values of d compared to n we can ignore d and the complexity is O(n). which is linear in nature.

(1). Radix sort with Insertion Sort(Non linear complexity)

Radix Sort (Insertion Sort)		
m	**n**	**Real Time(ms)**
25	10000	9601
	25000	60773
	50000	246064
	75000	548400
	100000	>10 mins
50	10000	17433
	25000	110640
	50000	435490
	75000	>10 mins
	100000	>10 mins
75	10000	25750
	25000	169198
	50000	646551
	75000	>10 mins
	100000	>10 mins

(2) Radix sort with Counting Sort(Linear complexity)

Radix Sort (Counting Sort)		
m	**n**	**Real Time(ms)**
25	100000	150
	250000	650
	500000	1640
	750000	2700
	1000000	3826
50	100000	333
	250000	1467
	500000	3562
	750000	5793
	1000000	8166

75	100000	590
	250000	2417
	500000	5655
	750000	9165
	1000000	12837

Conclusion:

Radix sort originally is a linear sorting algorithm that, it is running faster than nlogn but while using it with insertion sort it could go up to n^2. While using it with counting sort it retrieves its characteristic and gives the running time of order of n. From the result also we can see for such smaller values of n ranging from only 10000 to 100000 in radix sort with insertion sort table the algorithm takes lots of time compared to radix with counting sort and in that too the input values n for counting sort algorithm is 10 times greater than the radix with insertion. Reason as discussed in above section insertion sort is a stable sorting algorithm but it's a comparison based and that in its average and worst cases makes it in order of (n^2) and radix using it as its internal stable sort also becomes an algorithm of order n^2 loose it's characteristic of being a linear sorting algorithm as running time is greater than nlogn.

Another important observation to make is that when we discussed radix in above section we said that its running time O(dn) but for smaller values of d we can say its O(n). Out observation proves that there are differences for different values of m but they are not changing the entire order of algorithm or they are not drastically increasing the running time even for the very larger values of n.

Thus to sum up we can say that speed of Radix sort depends on the inner basic operations. Insert delete swapping all this kind of operations can be inside and if these operations have some flaws or are not efficient enough or not solving the purpose or not suitable with the radix sort, Radix sort can be much solver than others. And here in this assignment we have experienced it with the radix with insertion sort.