Shubham Sanjay Jain

CWID: 10456815

Stevens Institute of Technology

Master of Science in Computer Science

Submitted to: Prof. Iraklis Tsekourakis

CS 590 Algorithms

Report & Solution: Assignment 2

CS590 HomeWork 2 – Recurrences and Sorting
Name: Shubham Jain
CWID: 10456815
Course: CS590

Part 1 (20 points)

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 constant $c > 0$.
   (Note: lg n is monotonically increasing for $n > 0$)

Solution:
To Prove: For $T(n) = T(n-3) + 3\lg n$
$$\mathbf{T(n) \le c\, n \lg n}$$

Guess: $T(n) = O(n \lg n)$
$T(n) \le c\, n \lg n$
For all c = >1 and $n_0 < n$

Proof:
Base Case $n = 1 \Rightarrow n \log n = 1.0 = 0$

Inductive step $T(n-3) = (n-3).\lg (n-3)$

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

Since lg n is monotonically increasing for $n > 0$
Hence, lg n is greater than lg(n-3)

$$\le c.n\lg n - 3c\lg n + 3\lg n$$

Considering,
$c \ge 1$
Removing lower order terms,

$$\le c.n\lg n$$

Therefore, $T(n) \le c.n\lg n$, and proved our guess.

2. $T(n) = 4T(n/3) + n$. Our guess: $T(n) = O(n^{\log_3 4})$
   Show $T(n) = c.\ n^{\log_3 4}$ for some constant $c > 0$.

Solution:

To Prove: For T(n) = 4T(n/3) +n

$$\mathbf{T(n) = c.\ n^{\log_3 4}}$$

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

$T(n) \le c.\ n^{\log_3 4}$

For all c = >1 and n₀ < n

Solution:

Base case: n = 1 => $1^{\log_3 4}$ => 1

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

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

$\le 4.\ c.\ (n/3)^{\log_3 4} + n$

$\le 4.\ c.\ 4^{\log_3 (n/3)} + n$

$\le 4.\ c.\ 4^{\log_3 (n) - \cancel{\log_3 (3)} 1} + n$

$\le 4.c.4^{\log_3 (n) - 1} + n$

$\le 4.c.(4^{\log_3 (n)})/4 + n$

$\le c.\ 4^{\log_3 (n)} + n$

$\le c.n.^{\log_3 4} + n$

After Subtracting lower order term,

Our initial guess failed

So, taking new guess,

$T(n) \le c.n.^{\log_3 4} - dn$

Solving,

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

$T(n) \le 4.c.(n/3)^{\log_3 4} - (4/3)dn + n$

$\le 4.\ c.\ 4^{\log_3 (n) - \cancel{\log_3 (3)} 1} + n - (4/3)dn$

$\le 4.c.4^{\log_3 (n) - 1} + n - (4/3)dn$

$\le 4.c.(4^{\log_3 (n)})/4 + n - (4/3)dn$

$\le c.\ 4^{\log_3 (n)} + n - (4/3)dn$

$\le c.n.^{\log_3 4} + n - (4/3)dn$

Considering, $-dn/3 + n \le 0$,

We get,

$d \ge 3$.

After Subtracting lower order term,

$T(n) \le c.\ n^{\log_3 4} - 3n$

Therefore, $T(n) \le c.\ n^{\log_3 4}$, and proved our 2nd guess.

3. $T(n) = T(n/2) + T(n/4) + T(n/8) + n$. Our guess: $T(n) : O(n)$.
   Show $T(n) \leq c.n$ for some constant $c > 0$.

Solution:

To prove: For $T(n) = T(n/2) + T(n/4) + T(n/8) + n$,

$T(n) \leq c.n$

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

$T(n) \leq c.\ n$

For all $c = >1$ and $n_o < n$

Solution:

Base case: $n = 1 => 1$

Inductive Step : $T(n/2) = (n/2)$

$T(n/4) = (n/4)$

$T(n/8) = (n/8)$

$$\begin{aligned} T(n) &= T(n/2) + T(n/4) + T(n/8) + n \\ &\leq c.(n/2) + c(n/4) + c(n/8) + n \\ &\leq (4cn + 2cn + cn + 8n)/8 \\ &\leq (7c + 8).n/8 \end{aligned}$$

Considering, $(7c+8)/8 \geq 1$,

$c \geq$

$T(n) \leq c'.\ n$

Therefore, $T(n) \leq c.n$, and proved our guess.

4. $T(n) = 4T(n/2) + n^2$. Our Guess: $T(n) = O(n)$.
   Show $T(n) \leq c.n^2$ for some constant $c > 0$.

Solution :

To Prove: For $T(n) = 4T(n/2) + n^2$

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

$\mathbf{T(n) \leq c.n^2}$

Proof:

Base Case: $T(1) = 1$

Inductive step : $T(n/2) = n/2$

$$\begin{aligned} T(n) &= 4T(n/2) + n^2 \\ &\leq 4.c.(n/2)^2 + n^2 \\ &\leq \cancel{4}.c.n^2/\cancel{4} + n^2 \\ &\leq cn^2 + n^2 \end{aligned}$$

After Subtracting lower order term,

Our initial guess failed.

So, taking new guess,

Using Recurrence Tree to evaluate guess,

We get,

$T(n) \le c_1.n^2\log n - c_2n$

Solving,

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

$T(n) \le 4.c_1.(n^2\log n/2)/4 - 4.c_2n/4 + n^2$

$\le c_1.(n^2\log n/2) - c_2n + n^2$

$\le c_1.(n^2\log n) - c_1.(n^2\log 2) - c_2n + n^2$

$\le c_1.(n^2\log n) - (c_1 \log 2\text{-}1).(n^2) - c_2n$

$T(n) =$

Considering,

$C_2 \ge 0$ and $c_1 \log 2+1 \ge 0$,

$C_1 \ge 1 / \log 2$

Therefore, $T(n) \le c_1.n^2\log n - c_2n$, and proved our 2nd guess.

Part 2 : Radix Sort on Strings

Question 1: Implement an insertion sort algorithm for strings that sorts a given array of strings according to the character at position *d*. It is necessary to include the length of each string (*array A len*) as it is unclear whether or not the digit *d* exists. A non-existing digit *d* is interpreted as a 0 in the sorting process. Add a parameter *int d* and *int* A_len* to the algorithm arguments and modify the given

insertion sort algorithm accordingly. This algorithm should be implemented in the method below:

void insertion_sort_digit(char** A, int* A_len, int l, int r, int d)

Solution: I have developed the function called insertion_sort_digit which takes various arguments such as A as 2d vector array which stores the n number of strings of various length ranging from o to Max. Also, I have created one more function called string_compare_new(char* s1, char *s2, int d, int length) which compares 2 string at a specific digit and returns the comparison outcome. And from the output of the newly created function, I swap the values.

Question 2: Use this modified insertion sort algorithm to implement radix sort for an array of strings. Measure the runtime performance for arrays of random strings 10 times for every combination of array size n = 10000; 25000; 50000; 75000; 100000 and length of the random strings m = 25; 50; 75. Discuss your results. (You might have to adjust the value for *n* dependent on your computers speed, but allow each test to take up to a couple of minutes). This algorithm should be implemented in the method below:

void radix_sort_is(char** A, int* A_len, int n, int m)

Solution: I just ran the insertion_sort_digit function from last digit to its previous one and so on. And after running the function for last time on the first digit, the output we get is the sorted array of strings. But, the time required is higher enough because, Time Complexity is $O(n^2.d)$ where d is number of digits and n is input.

Question 3: Develop a counting sort algorithm for strings that sorts a given array of strings according to the character at position *d*. As for the insertion sort on digit *d*, a non-existing digit *d* is treated as a 0 throughout the counting sort. This algorithm should be implemented in the method below:

void counting_sort_digit(char** A, int* A_len, char** B, int* B_len, int n,int d)

Solution: The counting sort is linear time sorting algorithm which sorts using the count array where it contains counts of each character to say. And each count is added to precious one in order to get the index position of String in the newly created output array.

Question 4:Use this new counting sort algorithm to implement radix sort for an array of strings. Measure the runtime performance for arrays of random strings 10 times for every combination of array size n = 100000; 250000; 500000; 750000; 1000000 and length of the random strings m = 25; 50; 75. Discuss your results. (You might have to adjust the value for *n* dependent on your computers speed, but allow each test to take up to a couple of minutes). This algorithm should be implemented in the method below:

void radix_sort_cs(char** A, int* A_len, int n, int m)

Solution: Implemented Counting sort for each and every digit starting from last to the first. The time complexity is just linear time because of stable counting sort which also takes linear time to operate. The total time complexity is O((n+256)*d) where n is input size and d is digit of input size. Which turns out to be O(n) considering 256 and d as constant and won't affect for larger input values.

Following is comparison chart of both radix sort using Insertion sort and Counting Sort.

We can see that for larger input values, difference is 9,395 times more faster for counting sort. But, for smaller values, the counting sort is just 800 times faster. We can see for larger input values, the difference between them rises exponentially in terms of Time Complexity.

Radix Sort with Insertion Sort

Length of String	Length of Random Array	(Time in Milli Seconds)

25 50 75	**10000**	5274 10470 25675
25 50 75	**25000**	58787 53234 76469
25 50 75	**50000**	132301 217142 310650
25 50 75	**75000**	301159 480734 717170
25 50 75	**100000**	554341 1051610 1722534

Radix Sort with Counting Sort

Length of String	Length of Random Array	(Time in Milli Seconds)
25 50 75	**100000**	59 130 273
25 50 75	**25000**	309 880 1558
25 50 75	**500000**	1100 2444 3912
25 50 75	**750000**	1874 4133 6498
25 50 75	**1000000**	2685 5773 8869