CSCI 665

HOMEWORK 1.

ANUBHUTI PUPPALWAR
ap1401@g.rit.edu

RISHABH ARORA
ra8851@g.rit.edu.

Q1. 1) let $f(n) = 2n \Rightarrow 2^{f(n)} = 2^{2n} = 4^n$

let $g(n) = n \Rightarrow 3^{g(n)} = 3^n$

$$\lim_{n \to \infty} \frac{4^n}{3^n} = \infty$$

$$\Rightarrow 2^{f(n)} \neq O(3^{g(n)})$$

Hence, False.

2) $f_1(n) = O(g_1(n))$

$\Rightarrow f_1(n) \leq c_1 g_1(n)$ —— (1)

$f_2(n) = O(g_2(n))$

$\Rightarrow f_2(n) \leq c_2 g_2(n)$ —— (2)

Adding (1) & (2)

$f_1(n) + f_2(n) \leq c_1 g_1(n) + c_2 g_2(n)$

let $c_3 = \max(c_1, c_2)$

$\Rightarrow f_1(n) + f_2(n) \leq c_3 g_1(n) + c_3 g_2(n)$

$\Rightarrow f_1(n) + f_2(n) = O(g_1(n) + g_2(n))$

Hence, True.

3) let $f_1(n) = n^2$ let $f_2(n) = n$

let $g_1(n) = n^2$ let $g_2(n) = n^2$

$f_1(n) + f_2(n) = n + n^2$

$g_1(n) + g_2(n) = 2n^2$

$$\lim_{n\to\infty} \frac{n+n^2}{2n^2} = \lim_{n\to\infty} \frac{1/n + 1}{2} = \frac{1}{2}$$

$\Rightarrow f_1(n) + f_2(n)$ is $\Theta(g_1(n) + g_2(n))$

if Θ is true $\Rightarrow$ o not true

$\Rightarrow f_1(n) + f_2(n) \neq o(g_1(n) + g_2(n))$

Hence false.

4) let $f(n) = 2^n$

$f(n/2) = 2^{n/2}$

$$\lim_{n\to\infty} \frac{2^n}{2^{n/2}} = 2^{n/2} = \infty$$

$\Rightarrow f(n) = \omega(f(n/2))$

$\Rightarrow$ Hence, False

Q2. Ranking :

1) $1/n$
2) $\frac{1}{1000}$, 10^{1000}
3) $\log \log n$
4) $\sqrt{\log n}$
5) $\log n$, $\log_{10} n$, $2 \log n$

6) $\log^2 n$

7) $\sqrt{n}$

8) $n^{2/3}$

9) $n, 2^{\log n}, n + \log n, n \log 2$

10) $n \log n$

11) $n^{3/2}$

12) $n^2, n + \frac{n^2}{10^{20}}$

13) $2^n, 2^{n+1}$

14) $n \cdot 2^n$

15) 3^n

16) $4^n, 2^{2n}$

17) $n!$

18) $(n+1)!$

19) n^n

(Q3) Since the given array is sorted we have used the concept of binary search.

We initialize left as 0 and right as arraysize - 1. We find the middle element from $\frac{left + right}{2}$. We return duplicate element if array[middle] == array[middle+1] array[middle] == array[middle - 1] because middle-1 or middle+1 element is duplicate. If element in middle is equal to $\frac{(array[left] + array[right])}{2}$ then we initialize left to middle+1 because this means that there is no duplicate between left to middle index, as middle element is at the correct position, else it would have been at right or left.

If $\frac{(array[left] + array[right])}{2}$ is not equal to middle element then initialize right as mid-1.

This way we find the duplicate element.

Since we are dividing the array into half each time in while loop by reinitializi left and right like binary search. so complexity of this algorithm is $O(\log n)$.

---

4) To find if more than 1 stable matchings possible, we run the code twice. While running second time we make requester as responders and responders as requesters. We change the role of two sets. If we get different output

from both ⇒ there are more stable matchings possible. If we get same output of matches from both ⇒ only 1 type of matching is possible. Algorithm optimises for the the requesters and gives worst persimal result for the responders. If we get same output after changing role ⇒ optimal and worst output is same. So, more stable matchings cannot be created. Hence, this algorithm to swap the roles of groups and find matchings gives correct output.

Time complexity here is $O(n^2)$.
We check for each requester in group 1 in while loop ($O(n)$), each requester goes through all the responder till it gets perfect match. ($O(n)$).

Since there are n requester and n responders. If Person from group 2 prefers other person from group 1 than the one assigned → this check is done is O(1), other operations are also O(1). hence, overall timecomplexity is $O(n^2)$.

---

5)

Uniform distribution size	Time for Merge Sort	Time for Insertion Sort	Time for Bucket Sort
100	0.00014424	0.000180006	0.0000619888
1000	0.001696825	0.019852161407	0.0006389617
10000	0.021655797	1.96440815925	0.0063779354
100000	0.266664028	> 3 min	0.0696229934

Time complexity of Insertion sort is $O(n^2)$, Merge sort is $O(n \log n)$, Bucket-sort is $O(n)$. Since insertion sort has higest complexity, it takes higest time, then merge sort and least taken by bucket sort. This is observed on all distribution sizes. When the input size is 100000 Insertion sort takes more than 3 min. These observations are as per expected.

Gaussian (Normal) distribution size	Time for Merge Sort	Time for Insertion sort	Time for Bucket Sort
100	0.000118017	0.0001573526	0.00013303756
1000	0.001551151	0.01909470558	0.00953006744
10000	0.0210511684	1.984954118	0.536585092
100000	0.261665105	>3 min	5.6142061711

For gaussian distribution:
Insertion sort takes highest time then bucket sort and least taken by merge sort, for all distribution sizes.
When the size is 100000, Insertion sort takes more than 3 min.
We observe in this distribution that bucket sort takes more time than merge sort even though time complexity of merge sort ($O(n\log n)$) is more than bucket sort ($O(n)$) for all types of distributions.
