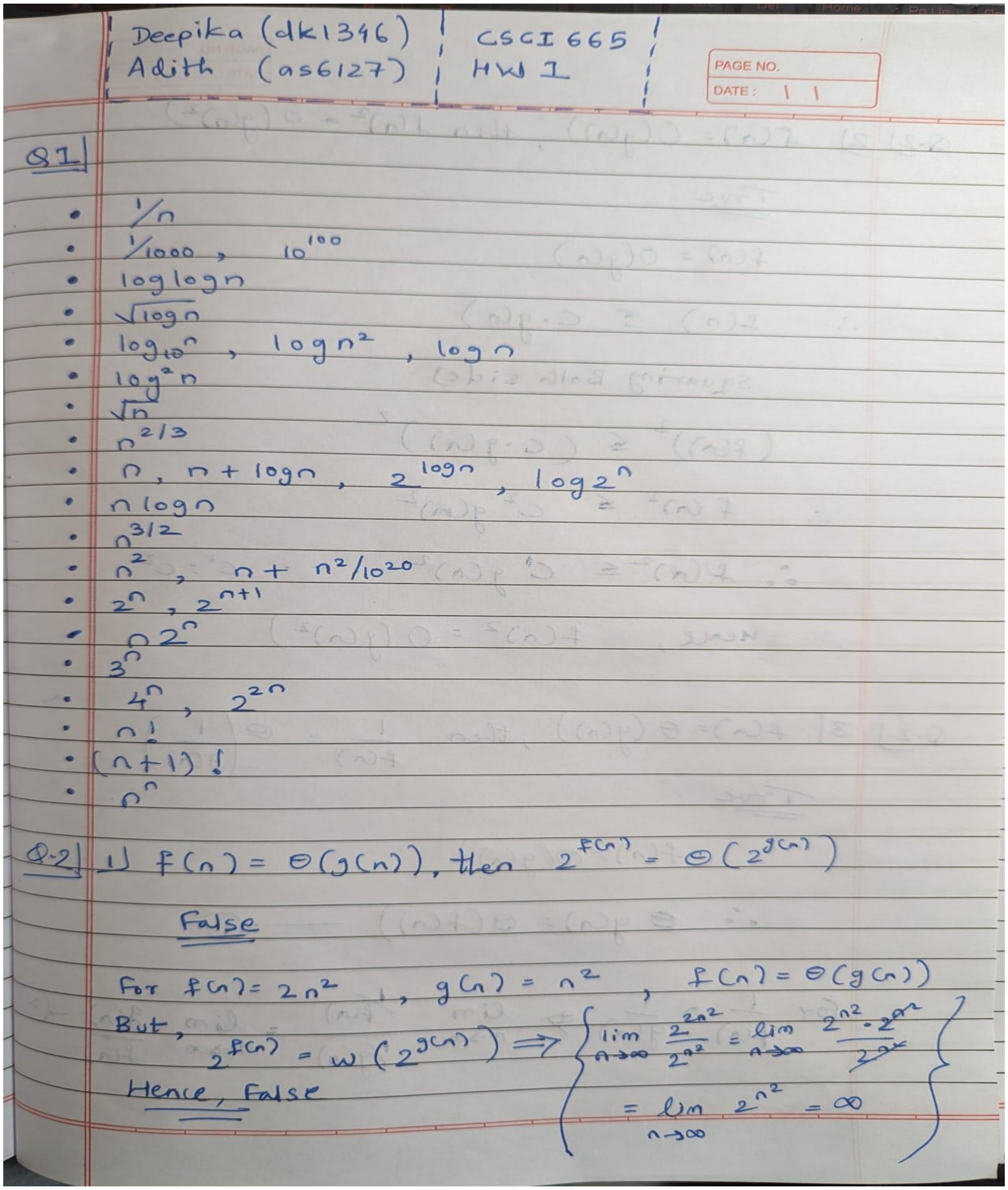
HW 1: CSCI 665
Deepika Kini (dk1346)
Adith Shetty (as6127)

Problems 1 and 2:



Deepika (dk1346)
Adith (as6127)

CSCI 665
HW 1

PAGE NO.
DATE : / /

Q1

- $1/n$
- $1/1000$, 10^{100}
- $\log\log n$
- $\sqrt{\log n}$
- $\log_{10} n$, $\log n^2$, $\log n$
- $\log^2 n$
- $\sqrt{n}$
- $n^{2/3}$
- n, $n + \log n$, $2^{\log n}$, $\log 2^n$
- $n \log n$
- $n^{3/2}$
- n^2, $n + n^2/10^{20}$
- 2^n, 2^{n+1}
- $n 2^n$
- 3^n
- 4^n, 2^{2n}
- $n!$
- $(n+1)!$
- n^n

Q.2 1) $f(n) = \Theta(g(n))$, then $2^{f(n)} = \Theta(2^{g(n)})$

False

For $f(n) = 2n^2$, $g(n) = n^2$, $f(n) = \Theta(g(n))$

But, $2^{f(n)} = \omega(2^{g(n)}) \Rightarrow$

$$\left\{ \lim_{n\to\infty} \frac{2^{2n^2}}{2^{n^2}} = \lim_{n\to\infty} \frac{2^{n^2} \cdot \cancel{2^{n^2}}}{\cancel{2^{n^2}}} = \lim_{n\to\infty} 2^{n^2} = \infty \right\}$$

Hence, False

PAGE NO.
DATE:

Q.2) 2) $f(n) = O(g(n))$, then $f(n)^2 = O(g(n)^2)$

True

$f(n) = O(g(n))$

$\therefore \quad f(n) \leq c \cdot g(n)$

Squaring Both sides

$(f(n))^2 \leq (c \cdot g(n))^2$

$\therefore \quad f(n)^2 \leq c^2 g(n)^2$

$\therefore \; f(n)^2 \leq c' g(n)^2 \quad \because c' = c^2$

Hence, $f(n)^2 = O(g(n)^2)$

Q.2) 3) $f(n) = \Theta(g(n))$, then $\frac{1}{f(n)} = \Theta\left(\frac{1}{g(n)}\right)$

True

$f(n) = \Theta(g(n))$

$\therefore$ ~~Θ~~ $g(n) = \Theta(f(n))$ —— ①

For $\frac{1}{f(n)}, \frac{1}{g(n)} \Rightarrow \lim_{n\to\infty} \frac{1/f(n)}{1/g(n)} = \lim_{n\to\infty} \frac{g(n)}{f(n)}$ — ②

From ①, we know that $g(n) = \Theta(f(n))$

Hence, $\lim_{n \to \infty} \frac{g(n)}{f(n)} =$ Non zero constant

Hence, $\frac{1}{f(n)} = \Theta\left(\frac{1}{g(n)}\right)$ — from ① & ②

Q.2] 4] $f(n) = \Theta(f(n-1))$

False

for $f(n) = n!$, $f(n-1) = (n-1)!$

$\lim_{n \to \infty} \frac{f(n)}{f(n-1)} = \lim_{n \to \infty} \frac{n!}{(n-1)!} = \lim_{n \to \infty} n = \infty$ //

$f(n) = \omega(f(n-1))$ for $f(n) = n!$ //

Hence, false //.

Problem 3:
The algorithm goes as follows:

- Overall: We use binary search to traverse through the list.
- Cost: Considering we take the midpoint and from there, we recursively break down the list into 2 equal lists and calculate the midpoint sums it into a O(log(n))/o(n) time complexity. (ignoring one half of list at every recursive call)
- Correctness:
  - Basically we get the mid point of the sorted list and compare if A[k] = k
  - Since the values are **integers** and **no duplicates** are present in the **sorted** list, if the list index value is greater than the value it holds, there is no possibility that the future array values will correspond to its index. (since index increases by just one integer value every time)
    - In this case, we ignore the latter half of the list and update the end pointer of the list to the midpoint and call the function recursively.
  - Similarly, if the index value is less than the value it holds, it means all the previous values in array too will never have a value equivalent to its index value.
    - In this case, we ignore the first half of the list and update the start pointer of the list to the midpoint and call the function recursively.

Problem 4:

Algorithm:
<u>Definition of GS:</u>
We use Gale Shapley to get the stable matching pairs for the two disjoint sets of same size where one set is an asker and the other becomes the responder.
<u>Significance of GS:</u>
Gale Shapley provides the best case for the asker set and the responder gets its worst case pair. Changing the sequence of askers doesn't change the pairing combination.
<u>How are we using this significance:</u>
We change the asker and responder sets, running the algorithm twice. This gives us an elements'/entities' best and worst match in its preference list.

Correctness of our assumption that if Gale Shapley provides 1 outcome for both runs, all other matching combinations will have the fixed pair:

For an asker a1, the preference will be from:

Best case → … moderate preferences .... → Worst case

which means for there to be a case where an asker is moderately preferred by a responder, there has to be a worst case responder that is different from the best case responder.
Hence when we have the same partner for asker's best and worst case, the responder too will have the best and worst as the asker, hence proving that in every other combination of stable matching, both the partners would prefer each other.
An example we could take is of two entities in different sets having each other as first preference. Then their worst and best case will be each other. This is proven by contradiction in class.

Running time of the algorithm:

- Data structures used to hold the free askers: Stack (top has the highest priority)
    - Reduces the complexity from O (n) to O (1) for finding the next available asker.
    - O (1) operation to pop a free asker, and O (1) to push an asker who is freed.
- · Preference sequence for askers: List
    - Index based pointer used to point to the next available responder for an asker.
    - This helps to assign the asker with the next preferred responder in O (1) time, instead of traversing through the responders who have rejected the asker.
- · Preference sequence for responders: List (stored inversely)
    - For each responder, preference position is stored for each entity.
    - Hence, O (1) time to get the preferred asker for each responder.

With the help of these changes, the work done in each iteration will be of a constant time, making the overall algorithm run in $O(n^2)$ time.

Problem 5

Testing the three sorting algorithms on uniform distribution of data

	Merge Sort	**Insertion Sort**	**Bucket Sort**
n=100	0.0003104000352323055	0.001106299925595522	0.00010879989713430405
n=1000	0.0036843998823314905	0.1358205999713391	0.0012630000710487366
n=10,000	0.05084359995089471	13.763781700050458	0.00983319990336895
n=100,000	0.7698776670731604	>3 min	0.2303623342886567

Testing the three sorting algorithms on normal distribution of data

	Merge Sort	**Insertion Sort**	**Bucket Sort**
n=100	0.0002973000518977642	0.0010631999466568232	0.0002848999574780464
n=1000	0.003949300153180957	0.13284160010516644	0.023229900049045682
n=10,000	0.04520540009252727	13.650703199906275	1.2002802998758852
n=100,000	0.6915262499824166	>3 min	23.10129175009206

Comparison of the algorithms and final results

- For Uniform distribution, the bucket sort algorithm is the fastest algorithm which is expected as it runs in O(n) time.
- The Merge sort algorithm is fast, but comparatively takes longer as the input size increases.

- Insertion sort is the worst performing algorithm, and for the largest input size, takes longer than 3 minutes, hence terminated.
- For Gaussian, the behavior is the same, but the bucket sort performs worse when compared to its performance with the uniform distribution. The reason for this is that the data is quite clustered in a normal distribution towards the mean of the range, which results in some buckets having more number of elements. Hence, the internal sorting of these buckets take more time, causing the algorithm to perform worse.
- As for insertion sort (comparative sort), same outcome as uniform data.
- Overall, merge sort shows steady time complexity.