CS 456: Advanced Algorithms

Fall 2014

Midterm Exam

Name	KEY
SIUE ID (last 3-digits)	

Instructions

- Write your name and the last 3-digits of your SIUE ID in the space provided above
- This is a *closed-book, closed-note* examination
- There are 7 total questions printed on 12 pages (including the title page). Please confirm that all pages are present
  - All students **must** answer Q1-Q5
  - Master's students **must** answer Q6
  - Q7 is an optional bonus question that all students can attempt
- Total points: 120* points for BS Students; 140* points for MS Students
- Total time: ***1 hour and 15 mins***

*

Question	Points
1	/20
2	/20
3	/20
4	/20
5	/20
6	/20
7*	/20

*Including the bonus points

Short Answers

Q1. **[20 points]** State whether the following statements are **true** or **false** and provide a 1 sentence justification if **false**

a. **[2 points]** A function with a faster order of growth is more efficient than a one with a slower order of growth

false. faster order of growth means less efficient

b. **[2 points]** If $f(n) \in O(g(n))$ and $h(n) \in \Omega(g(n))$ then $f(n) \in \Omega(h(n))$

false $h(n) \in \Omega g(n) \Rightarrow g(n) \in O\, h(n)$ thus, $f(n) \in O(h(n))$

c. **[2 points]** If $f(n) \in \Theta(g(n))$ then $g(n) \in \Theta(g(n))$

True. they are both in the same complexity class.

d. **[2 points]** Partial correctness = total correctness + termination

False T.C = P.C + Termination

e. **[2 points]** By definition, a spanning tree is cyclic

False acyclic

f. **[2 points]** Divide-and-conquer strategy always chooses the locally optimal solution

False Greedy

g. **[2 points]** Both dynamic programming and greedy strategies solve problems by dividing them into several overlapping subproblems

False. Dynamic Programming and D-n-Q.

h. **[2 points]** Kruskal's algorithm greedily grows a minimum spanning tree while Prim's algorithm greedily combines a forest to arrive at a minimum spanning tree

False. Otherway around.

i. **[2 points]** Codewords that are prefixes of other codewords are called prefix codes

False. Prefix codes are actually prefix-free.

j. **[2 points]** Dijkstra's algorithm applies to both directed and undirected graphs

True although undirected graphs may not yeild efficient use of Dijkstra.

Long Answers

Q2. **[20 points]** For each of the following cases, calculate whether $f(n) \in O(g(n))$ or $f(n) \in \Omega(g(n))$ or $f(n) \in \Theta(g(n))$.

a. $f(n) = \sqrt{n}$, $g(n) = log(n)$

$$\lim_{n\to\infty} \frac{\sqrt{n}}{\log n} \to \lim_{n\to\infty} \frac{\frac{1}{2}\frac{n(\ln 2)}{\sqrt{n}}}{} = \lim_{n\to\infty} \left(\frac{\ln 2}{2}\right)\sqrt{n}$$

$\to \infty$

thus, $f(n) \notin \Omega\, g(n)$

numerator.

denominator (~~divisor~~ divisor)

b. $f(n) = log(n^2)$, $g(n) = log(n)^2$

$$\lim_{n\to\infty} \frac{(\log(n^2))}{(\log n)^2} \to \lim_{n\to\infty} \frac{2\log n}{(\log n)^2} = \lim_{n\to\infty} \frac{2}{\log n} \to 0$$

thus $f(n) \notin O\, g(n)$

c. $f(n) = n^3 2^n$, $g(n) = 3^n$

$\lim_{n \to \infty} \frac{n^3 2^n}{3^n} \longrightarrow \lim_{n \to \infty} n^3 \left(\frac{2}{3}\right)^n \longrightarrow 0$

~~$f(n) \in \Omega\, g(n)$~~

$f(n) \in O(g(n))$

d. $f(n) = (n+1)!$, $g(n) = n!$

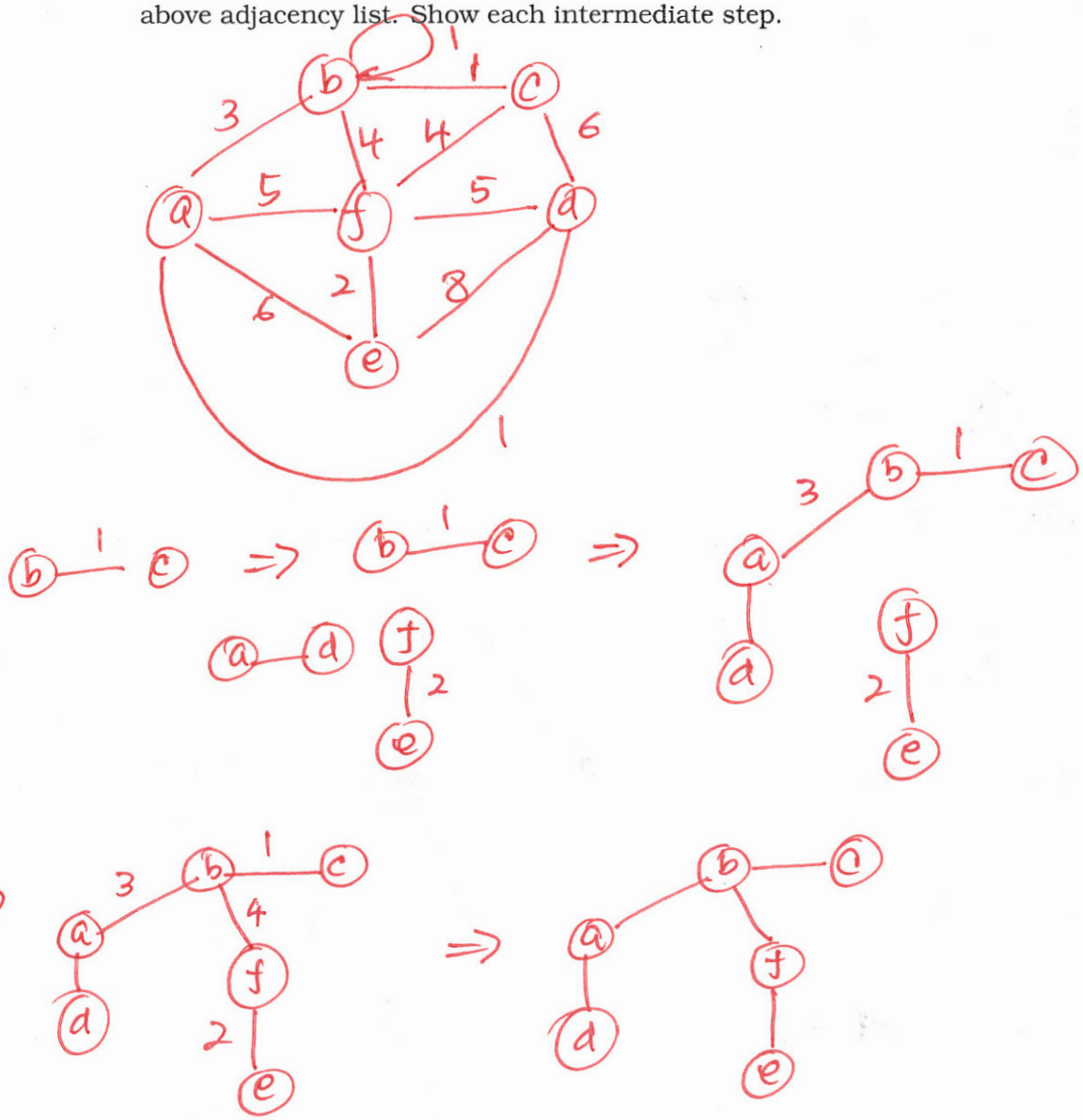
$\lim_{n \to \infty} \frac{(n+1)!}{n!} \longrightarrow \frac{n! \times (n+1) \times n!}{n!} \longrightarrow \infty$

$f(n) \in \Omega\, g(n)$

Q3. [**20 points**] Greedy Strategy

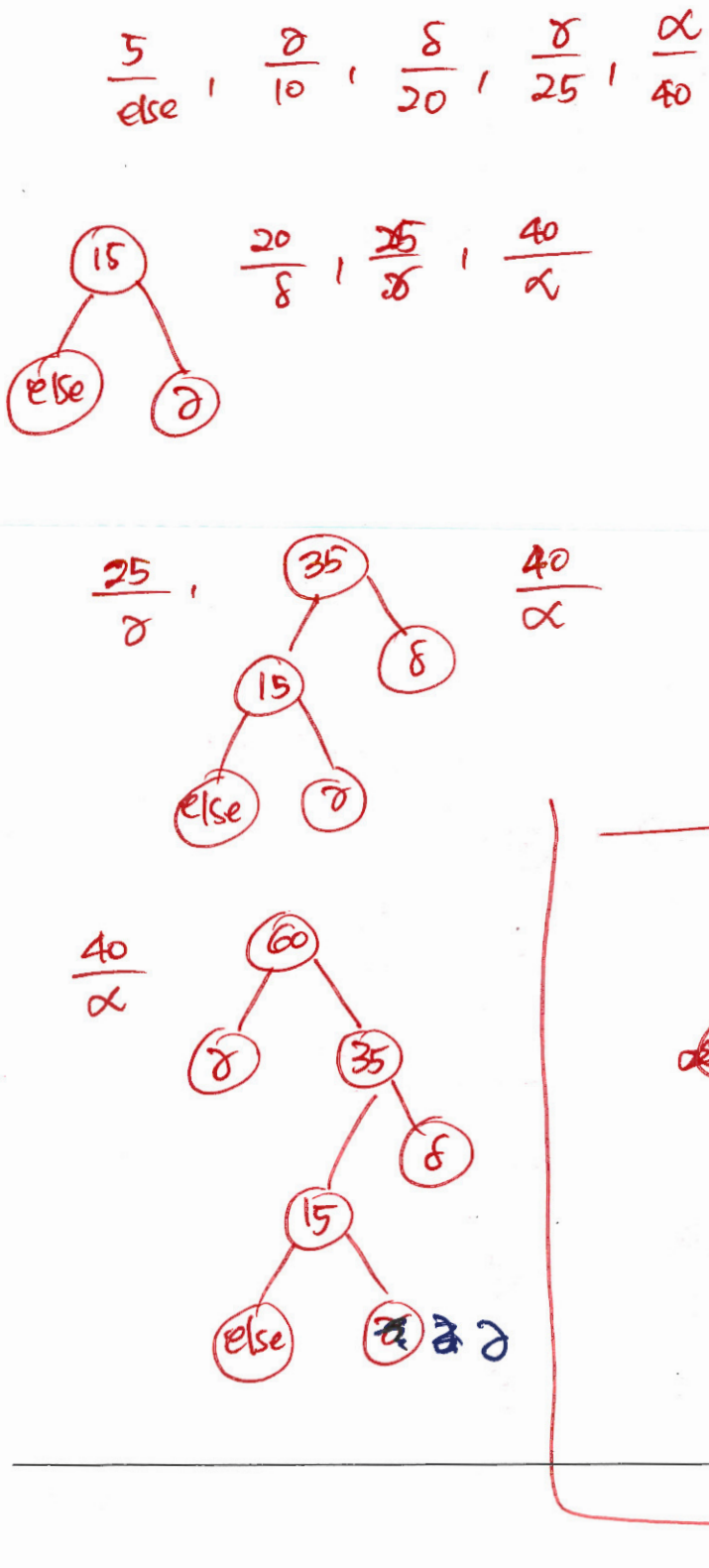
	a	b	c	d	e	f
a	0	3	0	1	6	5
b	3	1	1	0	0	4
c	0	1	0	6	0	4
d	0	0	6	0	8	5
e	6	0	0	8	0	2
f	5	4	4	5	2	0

a. [**10 points**] Use Kruskal's algorithm to find the MST for graph represented by the above adjacency list. Show each intermediate step.



b. **[10 points]** Construct a Huffman code for the following data. Show your work.

symbol	δ	∂	γ	a	else
frequency	20	10	25	40	5



Q4. [**20 points**] Divide-and-Conquer

a. [**10 points**] Find the order of growth for the following recurrences. Assume that $T(n)$ is constant for $n \leq 2$. Justify your answers.

i. $T(n) = 2T(n/4) + \sqrt{n}$

$a = 2$

$b = 4$

$n^{\log_b a} = n^{\log_4 2} = \sqrt{n}$

$f(n) = \sqrt{n} \Rightarrow f(n) = \Theta(n^{\log_b a})$

Case 2: $T(n) \in \Theta(\sqrt{n} \log n)$

ii. $T(n) = 2T(n/2) + n^2$

$a = 2$

$b = 2$

$n^{\log_b a} = n$

$f(n) = n^2 \Rightarrow f(n) = \Omega(n^{\log_b a + \varepsilon})$ where $\varepsilon = 1$.

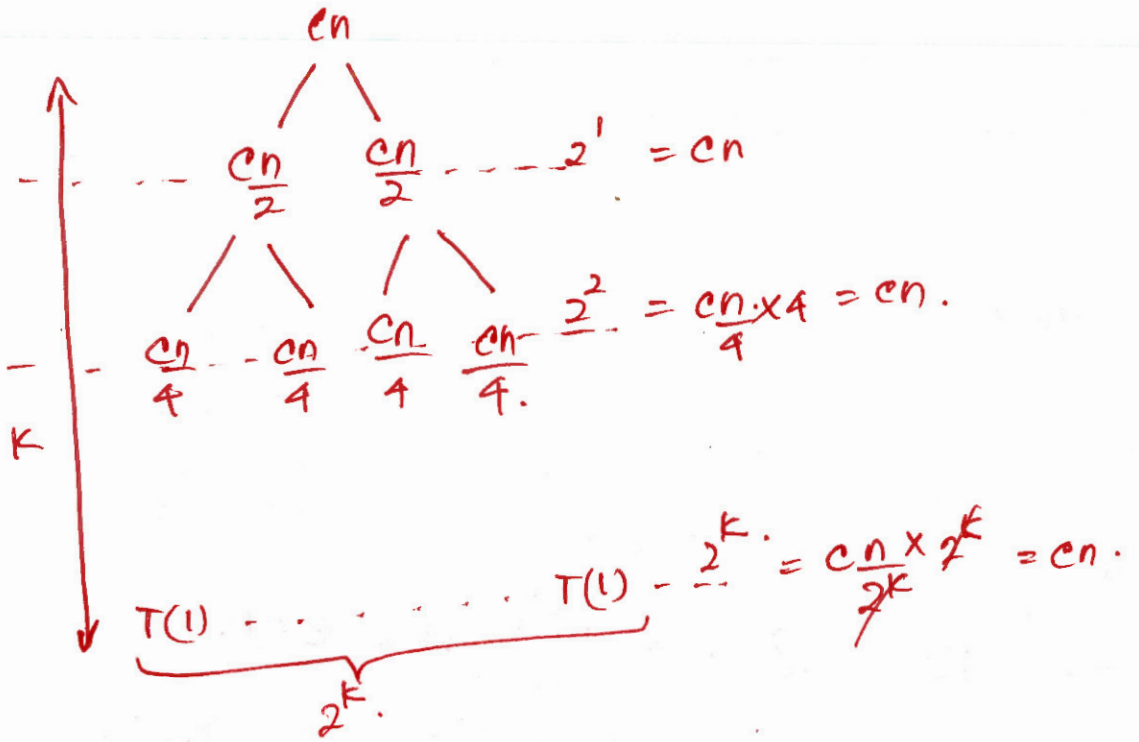
for sufficiently large n, $af\left(\frac{n}{b}\right) = 2 \cdot \left(\frac{n}{2}\right)^2 = \frac{1}{2}n^2 < cf(n)$ with $c = 1$.

$T(n) = \Theta(n^2)$

b. [**10 points**] Develop a recurrence as a function of n to determine the number of times the following code print Jumbo!. Solve your recurrence using a recurrence tree. (You may assume $n = 2^k$)

```
void jumbo(int n)
{
    if (n > 1)
    {
        jumbo(n/2);
        jumbo(n/2);
        for (i = 0; i < n; i++)
        {
            cout << "Jumbo!" << endl;
        }
    }
}
```

$T(n) = 2T\left(\frac{n}{2}\right) + cn.$



$T(n) = k \cdot cn$

$= n \log n$

$T(n) = \Theta(n \log n)$

Q5. **[20 points]** Present the optimal order of parenthesizing $M_1 \times M_2 \times M_3 \times M_4 \times M_5 \times M_6$ using the following dynamic programming tabulation. Show each step clearly.

			$m_{1,6}$ = best_k =		
		$m_{1,5}$ = best_k =		$m_{2,6}$ = best_k =	
	$m_{1,4}$ = best_k =		$m_{2,5}$ = best_k =		$m_{3,6}$ = best_k =
$m_{1,3}$ = 330 best_k = 3		$m_{2,4}$ = 330 best_k = 3		$m_{3,5}$ = 240 best_k = 5		$m_{4,6}$ = 480 best_k = 5
$m_{1,2}$ = 150 best_k = 1		$m_{2,3}$ = 360 best_k = 2		$m_{3,4}$ = 180 best_k = 3		$m_{4,5}$ = 240 best_k = 4		$m_{5,6}$ = 120 best_k = 5
$m_{1,1}$ = 0 [5x10]		$m_{2,2}$ = 0 [10x3]		$m_{3,3}$ = 0 [3x12]		$m_{4,4}$ = 0 [12x5]		$m_{5,5}$ = 0 [5x4]		$m_{6,6}$ = 0 [4x6]

$m_{1,3} = \min \{ M_1 \times (M_2 \times M_3) , (M_1 \times M_2) \times M_3 \}$ + = 150 + 180 = 330

150 + 5x10x12 , 150 + 5x3x12

360 k =

$m_{2,4} = \min \{ M_2 \times (M_3 \times M_4) , (M_2 \times M_3) \times M_4 \}$ = 180 + 150 = 330

+180 360 + 10x12x5

10x3x5

$m_{3,5} = \min \{ M_3 \times (M_4 \times M_5) , (M_3 \times M_4) \times M_5 \}$ = 180 + 60 = 240

3x12x4 + 240 180 + 3x5x4

$m_{4,6} = \min \{ M_4 \times (M_5 \times M_6) , (M_4 \times M_5) \times M_6$ = 360 + 120 = 480.

12x5x6 +120 240 + 12x4x6.

360 + 120 240 + 288

480 528

Q6. **[20 points]** Asymptotic Order of Growth

a. **[5 points]** Let $f(n) = 4n^2 + 9n - 81$ and $g(n) = n^2$. Find c_1, c_2, and n_0 to formally show that $f(n) \in \Theta(g(n))$

$c_1 n^2 \leq 4n^2 + 9n - 81 \leq c_2 n^2$

$0 \leq c_1 \leq 4 + \frac{9}{n} - \frac{81}{n^2} \leq c^2$

$n = 9, \Rightarrow c_1 \leq 4 + \frac{9}{9} - 1 \leq c_2$

$c_1 \leq 4 \leq c_2$

$n_0 = 9$

$c_1 = 3$

$c_2 = 5$.

b. **[5 points]** Let $h(n) = n^3$. Formally show that $f(n) \notin \Omega(h(n))$

$4n^2 + 9n - 81 \geq c_1 n^3 \geq 0$.

$0 \leq c_1 n \leq 4 + \frac{9}{n} - \frac{81}{n^2}$

$0 \leq c_1 \leq \frac{4}{n} + \frac{9}{n^2} - \frac{81}{n^3}$ → no c_1 can be found for all $n \geq n_0$

c. **[5 points]** Consider two algorithms with running times $T_A(n) = \frac{1}{2}n^2 \log_2 n$ and $T_B(n) = 100n^2$. At what value of n does algorithm$_A$ start to take less time than algorithm$_B$.

$\frac{1}{2} \cdot n^2 \cdot \log_2 n \leq 100 n^2$

$\log n \leq 200$

$n \leq 2^{200}$. → upto $n = 2^{200}$ T_A takes less time than T_B. after which it starts to take more time.

d. **[5 points]** If a step in algorithm$_A$ takes 1 ns, how much is the above runtime in years?

$\frac{1}{2} \cdot (2^{200})^2 \cdot \log_2 2^{200}$ ~~seconds~~ . steps.

$= \frac{100 \times 2^{400} \times 10^{-9} \text{ seconds}}{\text{seconds per year.}}$

Extra Credit

Q7. [**20 points**] Algorithmic correctness proofs

a. Using mathematical induction, prove that $6^n - 1$ is divisible by 5 for any $n \geq 1$.

Base case $n=1$ $\frac{6-1}{5} = 1$ ✓

I.H : $n=k$ $= \frac{6^k - 1}{5}$ is divisible by 5

I. Step $6^{k+1} - 1$ ~~$\Rightarrow 6(6^k - 1) - 1$~~

$6(6^k - 1) + 6 - 1 \Rightarrow 6\underbrace{(6^k - 1)}_{\text{divides by 5.}} + 5$

divisible by 5

b. Prove the correctness of the following pseudocode by considering the invariant {I: At the beginning of j^{th} iteration, $A[1 \ldots j-1]$ is sorted and is a permutation of the original $A[1 \ldots j-1]$}

```
{I}
for j = 2 → A.length do
    {I ∧ j ≤ A.length}
    key = A[j]
    i = j − 1
    while (i > 0) ∧ A[i] > key do
        A[i + 1]A[i]
        i = i − 1
    end
    A[i + 1] = key
    {I}
end
{I ∧ j = A.length}
```

Initialization: $j = 1$ $A[1 \ldots 1]$ trivially sorted

Check your notes. This problem was discussed in class