CIS507: Design & Analysis of Algorithms
Final, Spring 2012
(VERSION WITH ANSWERS)

Duration: 100 minutes
Total weight: 30%

Student Name: –

Student ID: –

Question	Points Obtained	Points Possible
1		2
2		2
3		6
4		10
5		4
6		2
7		4
Total		30
8 (bonus)		2
Grand Total		32

Cheat Sheet: Master Method

For $T(n) = aT(n/b) + f(n)$, with $a \geq 1$, $b \geq 1$, compare $f(n)$ with $n^{\log_b a}$.

Case	Condition, for $\epsilon > 0$	Solution
1	$f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$	$T(n) = \Theta(n^{\log_b a})$ Number of leafs dominates
2	$f(n) = \Theta(n^{\log_b a})$	$T(n) = \Theta(n^{\log_b a} \lg n)$ All rows have same asymptotic sum
3	$f(n) = \Omega(n^{\log_b a + \epsilon})$	$T(n) = \Theta(f(n))$ provided that $af(n/b) \leq cf(n)$ for some $c < 1$
2 (general)	$f(n) = \Theta(n^{\log_b a} \lg^k n)$ or some constant $k \geq 0$	$T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$ They grow at 'similar' rate

1 Trace Quicksort (2 points)

Draw the recursion tree generated by Quicksort on this array:

$$A[1, \ldots, 8] = \langle 33, 55, 11, 88, 77, 22, 66, 44 \rangle$$

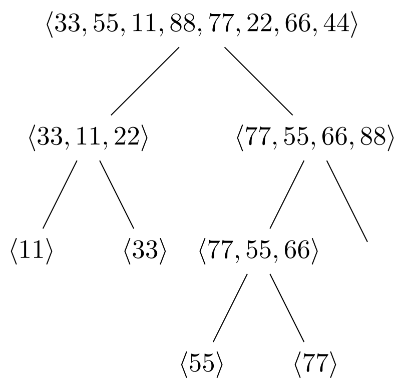
Label each node with the contents of the sub-array being sorted. So, the root should be labelled with the original contents of A, its two children will be the sub-arrays to be sorted recursively, etc.

Note: Make sure you move things around correctly as you do the partition. To help you, here is the partition sub-routine, which rearranges sub-array $A[p, \ldots, r]$ with pivot r. Note that the *last* element is chosen as the pivot.

```
PARTITION($A, p, r$)
  $x = A[r]$
  $i = p - 1$
  for $j = p$ to $r - 1$
      if $A[j] \le x$
          $i = i + 1$
          exchange $A[i]$ with $A[j]$
  exchange $A[i + 1]$ with $A[r]$
  return $i + 1$
```

ANSWER:

Note that we must use the last element as the pivot. So, in the first case, we split the array into two parts around element 44.



2 Hashing (2 point)

Suppose we use simple uniform hashing, and resolve collisions by chaining. Suppose you insert three keys into a hash table with $m = 10$ slots. What is the probability that slots 0 and 1 empty?

ANSWER:

Each key is independent of others and have equal probability inserting into each of the m slots. So for each key the chance of not inserting into slot 0 or 1 is $\frac{m-2}{m}$. Since each key is independent, the total probability is $\left(\frac{m-2}{m}\right)^3 = \left(\frac{10-2}{10}\right)^3 = \left(\frac{512}{1000}\right) = 0.512$.

3 Multiple Choice with One Correct (6 points)

For each of the following, circle the *single* correct answer.

1. **(1 point)** Recurrence $T(n) = 3T(n/3) + n$ has solution $T(n) =$
   (a) $\Theta(n)$ (b) $\Theta(n \lg n)$ (c) $\Theta(\lg n)$ (d) Unknown

2. **(1 point)** Recurrence $T(n) = 11T(n/7) + n^3$ has solution $T(n) =$
   (a) $\Theta(n)$ (b) $\Theta(n \lg n)$ (c) $\Theta(n^3)$ (d) Unknown

3. **(1 point)** You are given an adjacency-list representation of a directed graph with n vertices and m edges. Let k denote the maximum in-degree of a vertex. Each vertex maintains an array of its outgoing edges (but *not* its incoming edges). How long does it take, in the worst case, to compute the in-degree of a given vertex? (Recall that the in-degree of a vertex is the number of edges that enter it.)
   (a) $\Theta(k)$ (b) $\Theta(m)$ (c) $\Theta(n)$
   (d) $\Theta(k + m)$ (e) $\Theta(n + m)$ (f) $\Theta(k * m)$

4. **(1 point)** The $\leq_p$ relation (polynomial-time reducible) is a transitive relation.
   (a) Yes (b) No (c) For some classes of problems (d) Unknown

5. **(1 point)** A directed graph is called *strongly connected* if there is a path from each vertex in the graph to every other vertex. The *strongly connected components* of a directed graph G are its maximal strongly connected subgraphs. On adding one extra edge to a directed graph G, the number of strongly connected components:
   (a) might remain the same.
   (b) always decreases.
   (c) always increases.
   (d) always changes.

6. **(1 point)** We can prove that problem X is NP-Hard by:
   (a) Showing a polynomial time reduction from 3-SAT to X
   (b) Showing a polynomial time reduction from X to 3-SAT
   (c) Either of the above
   (d) None of the above

ANSWER:

1. (b) Using the notation of the Master Theorem, we have $a = 3$, $b = 3$, so $n^{\log_b a} = n$, and $f(n) = n$. Thus, by Case 2 of the Master Theorem, $T(n) = \Theta(n \lg n)$.

2. (c) Using the notation of the Master Theorem, we have $a = 11$, $b = 7$, so $n^{\log_b a} \approx n^{1.23}$, and $f(n) = n^3$. The regularity condition is easy to verify: $af(n/b) = 11(n/7)^3 = (11/343)n^3 \leq \frac{1}{2}n^3$. Thus, by Case 3 of the Master Theorem, $T(n) = \Theta(n^3)$.

3. (b): Without explicitly maintaining a list of incoming edges, you might have to scan all the edges to identify the incoming arcs.
4. (a) Yes.
5. (a). For example, the graph might already be strongly connected.
6. (a) Reduction from an NP-Complete problem.

4 True or False (10 points)

1. **(1 point)** If $T_1(n) = \mathcal{O}(f(n))$ and $T_2(n) = \mathcal{O}(f(n))$, then $T_1(n)+T_2(n) = \mathcal{O}(f(n))$.
2. **(1 point)** If $T_1(n) = \mathcal{O}(f(n))$ and $T_2(n) = \mathcal{O}(f(n))$, then $\frac{T_1(n)}{T_2(n)} = \mathcal{O}(1)$.
3. **(1 point)** If $T_1(n) = \mathcal{O}(f(n))$ and $T_2(n) = \mathcal{O}(f(n))$, then $T_1(n) = \mathcal{O}(T_2(n))$.
4. **(1 point)** Using a comparison sorting algorithm, we can sort any 7 numbers with up to 12 comparisons.
5. **(1 point)** Counting sort is *not* a comparison sorting algorithm.
6. **(1 point)** Running merge sort on an array of size n which is already correctly sorted takes $\mathcal{O}(n)$ time.
7. **(1 point)** If a problem is $NP-complete$, then no polynomial time algorithm exists for solving it.
8. **(1 point)** Suppose we know that a problem X is NP-complete. If we find a polynomial-time algorithm for X, this mean that we can solve SATISFIABILITY in polynomial time.
9. **(1 point)**) Suppose we know that a problem X is in NP. If we find a polynomial-time algorithm for X, then we can solve SATISFIABILITY in polynomial time.
10. **(1 point**) Suppose we find an $\mathcal{O}(n^2)$ algorithm for SATISFIABILITY. This implies that every problem in NP can be solved in time $\mathcal{O}(n^2)$.

ANSWER:

1. True. Since $T_1(n) = \mathcal{O}(f(n))$ and $T_2(n) = \mathcal{O}(f(n))$, it follows by definition that there exists constants $c_1, c_2 > 0$ and positive integers n_1, n_2, such that $T_1(n) \leq c_1 f(n)$ for $n \geq n_1$ and $T_2(n) \leq c_2 f(n)$ for $n \geq n_2$. This implies that $T_1(n)+T_2(n) \leq (c_1+c_2)f(n)$ for $n \geq \max\{n_1, n_2\}$. Therefore $T_1(n) + T_2(n) = \mathcal{O}(f(n))$.
2. False. A counter-example is $T_1(n) = n^2$, $T_2(n) = n$, and $f(n) = n^2$. Then $T_1(n) = \mathcal{O}(f(n))$ and $T_2(n) = \mathcal{O}(f(n))$. But $\frac{T_1(n)}{T_2(n)} = \frac{n^2}{n} = n \neq \mathcal{O}(1)$
3. False. A counter-example is $T_1(n) = n^2$, $T_2(n) = n$, and $f(n) = n^2$. Then $T_1(n) = \mathcal{O}(f(n))$ and $T_2(n) = \mathcal{O}(f(n))$. But $n^2 \neq \mathcal{O}(n)$.
4. False. The binary tree must have $7! = 5,040$ leaves to capture all possible orderings of the 7 numbers. The number of leaves of a complete binary tree of height 10 is $2^{12} = 4,096$. This is not enough.

5. True. No comparisons are actually made.

6. False. The merge sort algorithm presented in class always divides and merges the array $\mathcal{O}(\log n)$ times, so the running time is always $\mathcal{O}(n \log n)$.

7. False. This is true only if $P \neq NP$, which is an open question.

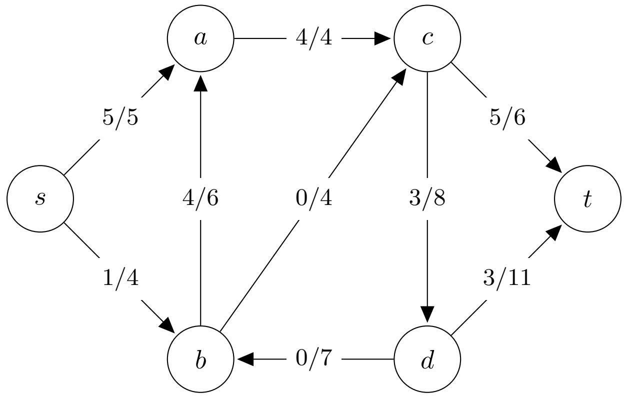
8. True. If X is NP-complete, then, by definition of NP-completeness, there is a reduction from Satisfiability to X. This reduction combined with the algorithm for X provides an algorithm for SATISFIABILITY.

9. False. It just means that we know X is in P as well as NP. But every problem in P is also in NP. This tells us nothing about NP-complete problems.

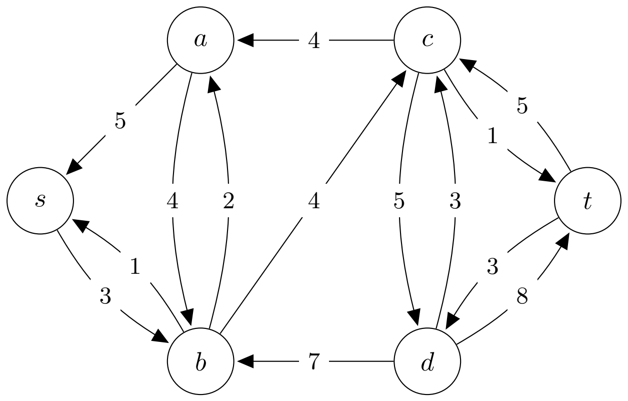
10. False. It shows that $P = NP$ since for every problem X in NP there exists a polynomial time reduction to Satisfiability and this reduction combined with the $\mathcal{O}(n^2)$ algorithm for SATISFIABILITY provides an algorithm for X. But the running time of this algorithm for X depends on the running time of the reduction from X to SATISFIABILITY which might not be $\mathcal{O}(n^2)$.

5 Flow (4 point)

Draw the residual graph of the following flow graph:



<u>ANSWER:</u>



6 Amortized Analysis (2 point)

Consider a linked list that has the following operations defined on it:

Operation	Description	Cost
$AddLast(x)$	Adds the element x to the end of the list	1
$RemoveFourths()$	Removes every fourth element in the list i.e. removes the first, fifth, ninth, etc., elements of the list.	Equals the number of elements in the list

1. Assume we perform n operations on the list. What is the worst case (asymptotic) run time of a call to $RemoveFourths()$?
2. Using the accounting method, compute the amortized cost per operation for a sequence of these two operations (give the amounts that you will charge $AddLast()$ and $RemoveFourths()$, and show how you will use these charges to pay for the actual costs of these operations). Keep your answer brief.

ANSWER:

1. Worst case is $\mathcal{O}(n)$ which happens when we call $AddLast()$ for $n-1$ times and then call $RemoveFourths()$.
2. $AddLast(x)$ gets charged 5 dollars. $RemoveForths()$ gets charged 0 dollars. When we call $AddLast(x)$, we spend 1 dollar immediately to pay for the cost of the call, we then store the remaining 4 dollars with the item added to the list. When we call $RemoveFourths()$, every element in the list has 4 dollars stored with it. We take the 4 dollars from each item that is deleted to pay for the cost of the call to $RemoveFourths()$. We can do this since $n/4$ items are deleted and so there are n dollars on these deleted items. Thus, at the end of the call to $RemoveFourths()$ all remaining elements in the list still have 4 dollars stored on them. The amortized cost per operation is therefore $\mathcal{O}(1)$.

7 Multiple Choice with Zero or More Correct (4 points)

For each of the following, circle all (zero or more) correct answer(s). You will lose 0.25 points per incorrect choice.

1. **(1 point)** Showing a polynomial time reduction from 3-SAT to problem X proves that X is:
   (a) P (b) NP (c) NP-Complete (d) NP-Hard

2. **(1 point)** If X is NP-Complete, this implies that X is:
   (a) NP (b) EXP (c) P (d) NP-Hard

3. **(2 point)** 3-SAT is:
   (a) P (b) NP (c) CoNP (d) NP-Hard
   (e) coNP-Hard (f) NP-Complete (g) CoNP-Complete (h) EXP

ANSWER:

1. (d) This only shows NP-hardness, since X is not necessarily in NP.
2. (a), (b), (d): Definition of NP-complete. Moreover, $NP \subseteq EXP$.
3. (b), (d), (f), (h): 3-SAT is NP-complete. So it is also NP, NP-Hard. It is also EXP since an exponential time algorithm can solve it.

8 Bonus Question (2 points)

0.5 point per choice.

Consider graphs that are undirected, unweighted, and connected. The *diameter* of a graph is the maximum, over all choices of vertices s and t, of the shortest-path distance between s and t. Next, for a vertex s, let $l(s)$ denote the maximum, over all vertices t, of the shortest-path distance between s and t. The *radius* of a graph is the minimum of $l(s)$ over all choices of the vertex s. Which of the following inequalities always hold (i.e., in every undirected connected graph) for the radius r and the diamter d?

(a) $r \geq d/2$ (b) $r \leq d$ (c) $r \leq d/2$ (d) $r \geq d$

ANSWER:

Answer is (a) and (b): Explanation as follows:

- $r \geq d/2$. Let c minimize $l(s)$ over all vertices s. Since every pair of vertices s and t have paths to c with at most r edges, stitching these paths together yields an $s-t$ with only $2r$ edges; of course, the shortest $s-t$ path is only shorter.
- $r \leq d$. By the definitions, $l(s) \leq d$ for every single choice of s.
- $r \nleq d/2$. The triangle is a counterexample.
- $r \ngeq d$.