Test Two (Answer Key)  
CS495/Ma475: Analysis of Algorithms  
15 May 1996

This is a closed book, 110 minute examination. It is worth 150 points. You may use one page of notes, on an 8.5x11 inch sheet of paper. There are 15 questions on six (6) pages. 

**Put your name on each page of this examination.**  
Assume that all constants are greater than zero, unless the problem states otherwise.

---

**Question 1 (12 points)** Match the following types of probabilistic algorithm with the characteristics which most closely apply to them.

- Las Vegas
- Monte Carlo
- Stochastic Preconditioning (Sherwood)
- Numerical

(a) Always gives a totally correct answer.
(b) Iterating this type of algorithm will improve the accuracy of its answer, though it may never be totally accurate.
(c) Iterating this type of algorithm will reduce the probability that this type of algorithm will err.
(d) Iterating this type of algorithm will improve the probability of getting an answer.

**Answer**

d Las Vegas  
c Monte Carlo  
a Sherwood  
b Numerical
Question 2 (5 points) ______ (True or False) If a problem is NP complete, then no algorithm exists to solve it.

Answer  False. They are only “likely to be hard”. For example, you know of an algorithm to solve 3PS, namely the brute force algorithm.

Question 3 (5 points) ______ (True or False) Any optimization problem can be solved with a Greedy algorithm.

Answer  False. Consider, for example, three processor scheduling.

Question 4 (9 points) Match the following analysis techniques with the appropriate statements about them.

______ Average case.
______ Worst case.
______ Amortized case.

(a) Analyzes sequences of operations.
(b) Generally, the easiest of these three to perform.
(c) Examines “typical” input instances.

Answer

c Average case.
b Worst case.
a Amortized case.
Question 5 (8 points) Discuss the pros and cons (give two of each) for the following claim: “A problem has a computationally feasible solution if and only if it is in P”.

**Answer** There are many. Among the pros: the strong Church Turing thesis implies that P is robust with respect to computation model, that is is remains the same regardless of your model of computation. Also, runtime greater than polynomial is almost certainly non-feasible. All polynomial time algorithms that we know of are of low degree with small multiplicative constants. Finally, and less obviously, it is closed under polynomial time reductions.

Among the cons: the hidden constants can be arbitrarily large. It is restricted to decision problems. The precise encodings of problems are not explicitly stated in the definition of the class. It ignores randomness, and probabilistic algorithms can be considered “feasible”.

Question 6 (12 points) Name two similarities and two differences between dynamic programming and divide and conquer.

**Answer** Similarities: both solve smaller instances of the problem and combine them. Both can be very efficient.

Differences: DP is “bottom up”, while DC is “top down”. DP tends to use more space. DP can be used when DC cannot—that is, whenever the principle of optimality holds.
Question 7 (15 points) Suppose you test algorithm \texttt{Foo} by implementing it in ADA and running your program on 1,000,000 randomly selected inputs. Each time you run the program, it takes exactly $40n^2 - 2n + 3$ microseconds, where $n$ is the size of the input instance. What can you conclude about the time requirements of \texttt{Foo}?

\textbf{Answer} You cannot say \textit{anything} about it.

You tested the program, not the algorithm. Also, you tested a finite number of cases, so this says nothing about the asymptotic behavior of the algorithm.

Even worse, many algorithms have different average case and worst case behavior. Assuming your ADA program correctly implemented \texttt{Foo}, you are most likely seeing average case behavior. This may in fact mislead you about worst case behavior.

Moreover, the input instances which may actually be of interest may not be randomly selected ones. So, this test may not even be appropriate.

Question 8 (10 points) The \textit{divide and conquer recurrence} is

$$t(n) = t\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + t\left(\left\lceil \frac{n}{2} \right\rceil \right) + e(n)$$

Suppose that $e(n) \in \Theta(n)$. What is the asymptotic (big $\Theta$) behavior of $t(n)$? Why?

\textbf{Answer} By the divide and conquer recurrence, $l = 2$, $b = 2$, and $k = 1$, so $l = b^k$, which implies that $t(n) \in \Theta(n \log n)$.

Question 9 (10 points) Must an algorithm must be correct to be useful? Why or why not?

\textbf{Answer} Not necessarily. Approximations, or probabilistic algorithms which may err or fail to terminate, can be very useful.

One should be careful, though. Safety critical applications may require total correctness, for example.
Question 10 (10 points) Which contributes more to the runtime efficiency of a program: the algorithm it implements, the programming language in which it is written, or the machine which runs it? Why?

Answer Only by changing the algorithm can you have more than a constant speedup in the runtime. The algorithm is therefore by far the most important factor contributing to runtime performance.

But, if your program will be limited to a small number of inputs then asymptotic considerations might not apply, and you would be better off looking at special purpose hardware or compilers with excellent optimizations. However, most program designers who made this assumption have found it to be false: once more powerful hardware is available, people want to process larger input instances. Only changing the algorithm can accommodate this growing demand with better than constant improvement in performance.

Question 11 (10 points) Why is it important to know that a problem is NP complete?

Answer So that you won’t waste your time trying to find an efficient algorithm for it. You’re better off seeking an approximation or a heuristic, or perhaps trying to find a restriction of the problem which is more tractable.

Question 12 (10 points) Suppose you are looking for an algorithm to solve problem $F\ddot{u}$, and you are able to show that $F\ddot{u} \leq_p \text{SAT}$ (recall that SAT is NP complete). What can you conclude about $F\ddot{u}$?

Answer Very little. To show $F\ddot{u}$ NP complete, you would need to show that SAT $\leq_p F\ddot{u}$, not the other way around. Intuitively, the reduction in the question shows that $F\ddot{u}$ is no harder than SAT, but this leaves open three possibilities: it may be feasible (in P), it may be infeasible but not NP complete, or it may be equally as hard as SAT.

Actually, one can conclude from the given reduction that $F\ddot{u} \in \text{NP}$, since NP is closed downward under $\leq_m$ (that is, if $B \in \text{NP}$ and $A \leq_m B$, then $A \in \text{NP}$). But we did not prove this property in class. And this knowledge helps very little when trying to classify $F\ddot{u}$. 
Question 13 (15 points) Analyze the worst case complexity of the following algorithm.

Procedure Foo(x);
Begin
   If x > 0
      Then
         Begin
            z := Foo(x/2) + Foo(x/2);
            Prod := 1;
            For i := 1 to x Do
               Prod := Prod * z;
         End
      Else
         Return 3
   End;
End;

Answer If $T(n)$ is the time for this algorithm to process an input $x = n$, then $T(n) = 2T(n/2) + n + c$ for some constant $c$. Note that this is the recurrence from question 8, so we have already shown that it is in $\Theta(n \log n)$.

But, here is the derivation the hard way: Assume $n = 2^k$ and let $T(n) = T(2^k) = t_k$. Then

$$t_k = 2t_{k-1} + 2^k + c$$

So

$$t_k - 2t_{k-1} = 2^k + c$$

The characteristic equation for this recurrence is

$$(x - 2)^2(x - 1) = 0$$

which has roots 2, of multiplicity 2, and 1, of multiplicity 1.

The closed form for this recurrence is, therefore,

$$t_k = c_0 2^k + c_1 k 2^k + c_2 1^k$$

for some constants $c_0$, $c_1$, and $c_2$.

This is clearly $O(k 2^k) = O(n \log n)$. 
Question 14 (10 points) Suppose that the algorithm $\text{Bar}(n)$ requires time $\Theta(n)$. What is the worst case complexity of the following algorithm? Use either Big-Oh or Big-Theta notation.

Procedure $\text{Foo}(n)$;
Begin
    $x := 0$;
    For $i := 1$ to $n$ Do
        $x = x + \text{Bar}(i)$;
End;

Answer $\sum_{i=1}^{n} \Theta(i) \in \Theta(n^2)$.

Question 15 (9 points) Prove or disprove the following

1. $2n^2 \in \Theta \left( \frac{n^2 + n}{2} \right)$
2. $(n + 2)^3 \in O(n^3)$
3. $n^4 \in \Omega(n^2)$

Answer Notice that
$$\lim_{n \to \infty} \frac{2n^2}{(n^2 + n)/2} = \lim_{n \to \infty} \frac{4n^2}{n^2 + n} = 4$$

So (1) follows from the limit rule.

For (2), notice that
$$(n + 2)^3 = n^3 + 6n^2 + 12n + 8$$

By the max rule, this is in $O(n^3)$.

(3) follows from the limit rule, since
$$\lim_{n \to \infty} \frac{n^4}{n^2} = \infty$$