This is a closed book, closed notes exam.

CS 495/MA 405: Analysis of Algorithms. Test 1

11 March 1994

Problem 1-1: (10 points) Describe a greedy solution to the three processor scheduling problem. Be sure to identify the following functions: Select(), Solution(), the objective function, and Feasible().

Answer:
Several answers are possible. For each of them, Solution() will determine whether every task has been scheduled, and the objective function will be the makespan, or maximum runtime of any processor with a given partial schedule. The Feasible() function will determine whether a particular extension is legal, and is therefore trivial.

Several Select() functions lead to different algorithms. For example, Select() might return an assignment of an unassigned task to the processor on which it runs fastest. Or it might return an assignment which minimizes the partial schedule achieved by using it. Other answers are possible.

Problem 1-2: (10 points) Suppose that the function \text{Bar}(n) requires time in \Theta(n). What is the worst case complexity of the following algorithm? Use either Big-Oh or Big-Theta notation.

Procedure 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).
**Problem 1-3:** (10 points) What is the worst case complexity of the following algorithm? Use either Big-Oh or Big-Theta notation.

ProcedureFoo(n);
Begin;
    Fori := n downto 1 Do
        If i > 5
            Then
                Forj := 1 to i Do
                    Writeln("Yowza!");
    End;
End;

Answer:
For sufficiently large n, the number of Writelnswill be
\[
\sum_{j=1}^{n} j - \sum_{j=1}^{5} j = \left(\frac{n}{2}\right) - 15 \in \Theta(n^2)
\]

**Problem 1-4:** (10 points) What is the worst case complexity of the following algorithm? Use either Big-Oh or Big-Theta notation.

ProcedureFoo(n);
Begin
    Sum := 0;
    While n > 1 Do
        Begin;
            Sum := Sum + 1;
            n := n Div 2;
        End;
    Fori := 1 to Sum Do
        Writeln(".");
End;

Answer:
The while loop executes $\Theta(\log n)$ times, and the for loop executes for exactly as many iterations as the while loop (since Sum counts the number of times through the former and controls the latter). Thus, this algorithm is in $\Theta(\log n)$.

**Problem 1-5:** (10 points) What is the worst case complexity of the following algorithm? Use either Big-Oh or Big-Theta notation.

```plaintext
Procedure Foo(x);
Begin
    If x = 1 Then Return 1;
    Sum := 0;
    For i := 1 to x Do
        Sum := Sum + i;
    Return Sum + Foo(x-1);
End;
```

**Answer:**

Let $t_n$ be the time which this algorithm requires on input $n$. Then $t_n = t_{n-1} + n + a$ where $t_{n-1}$ is the time for the recursive invocation of $\text{Foo}$, $n$ is the time for the loop, and $a$ is the constant overhead for things like initialization. To make the non-linear part more obvious, this is equal to

$$t_n - t_{n-1} = 1^n (n + a)$$

The characteristic equation for this is

$$(x - 1)^3$$

So the closed form is

$$c_1 + c_2 n + c_3 n^2$$

for some constants $c_1, c_2, c_3$. This is in $\Theta(n^2)$.

**Problem 1-6:** (10 points) Suppose algorithm $\text{Foo}()$ requires $s_n$ bytes of storage for inputs of size $n$, and that

$$s_n = 2s_{n-1} + 3s_{n-2}$$

How much storage does $\text{Foo}()$ require, asymptotically (in closed form)?
Answer:
The characteristic equation for this recurrence relation is

\[(x - 3)(x + 1)\]

so the closed form solution is

\[c_1 3^n + c_2\]

for some constants \(c_1\) and \(c_2\). This is in \(\Theta(3^n)\).

**Problem 1-7:** (10 points) Suppose \(\text{Foo}(n)\) requires time in \(O(\log n)\) in the worst case, but any sequence of \(n\) executions of \(\text{Foo}(n)\) requires time in \(O(n)\). Can this happen? What is the amortized complexity of each call to \(\text{Foo}(n)\) in a sequence of \(n\) calls?

**Answer:**
Yes, it can happen. In fact, it did with the example of a binary counter and with the insertion algorithm for binomial heaps.

The amortized time is \(O(1)\) for each call in the sequence.

**Problem 1-8:** (10 points) **Insertion Sort** is \(O(n^2)\) average complexity, and \(O(n^2)\) in the worst case. But it is \(O(n)\) in the best case, when the input is sorted.

Suppose you have to sort a file with about a million entries, every night. This file is modified about a hundred times each day. Would you use **Insertion Sort**? Why or why not?

**Answer:**
Perhaps. With only .01 percent of the entries changing in a day, the dataset will be very nearly sorted at the end of the day. That means that both average case and worst case analysis are misleading. **Insertion Sort** approaches best case behavior under these conditions. That is, it will require very close to \(O(10^6)\) units of time. Hence, it would be a good algorithm in this case.

In its favor: it is easy to code, hard to get wrong, and pretty fast in this case.

On the other hand, there are \(O(n \log n)\) algorithms available. For this particular problem, that means they will need about \(O(6 \cdot 10^5)\) units of time, which is very close to that **Insertion Sort** would need. In the future, the number of changes may increase, in which case **Insertion Sort** would be a bad idea. I might make sense to switch to an asymptotically faster algorithm to ward off this possibility.
Besides, there are adaptive \( O(n \log n) \) algorithms, though they are typically harder to code correctly. These would require very close to \( O(n) \) time in this particular instance, though the multiplicative constant would be higher than with Insertion Sort.

Problem 1-9: (10 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
\[
\lim_{n \to \infty} \frac{n^4}{n^2} = \infty
\]

Problem 1-10: (10 points) Suppose you must choose between algorithms \( A \) and \( B \), which use time in \( O(n^2) \) and \( O(n^{1.5}) \), respectively, in the worst case. What additional information, if any, do you need in order to select one of these two algorithms?

Answer:
Many answers are possible. For example, you might want to know if the worst case is overly pessimistic. It may be that the input instances are such that the worst case behavior will not be observed in practice. In this case, you would need to know the “typical” behavior of the algorithms. Depending on the actual input instances, and the way the algorithms will be used, average or best case or (if possible) amortized analysis may be more appropriate.
You might also want to know the hidden constants in these big-Oh complexities. For example, $A$ might be preferable if $B$ actually requires $t(n) = 2^{50} n^{1.59}$ units of time on input instances of time $n$.

You might also want to know that the algorithms are correct. Their efficiency is irrelevant if they are wrong.

You might consider how easy the algorithms are to maintain, particularly if they are likely to be modified or if they are part of a commercial package.

The upper bound may not be tight—how about some lower bounds?