due Wednesday, 10 September 2014, at 5:00 PM

[Total points: 42]

1. [worth 3 points] *Counting exact numbers of primitive operations.*

The algorithm selection sort is described below:

```
procedure SELECTION-SORT($A, n$) is
    for $k \leftarrow 1$ to $n-1$ do
        SELECT($A, k, n$)
    end do
end SELECTION-SORT

procedure SELECT($A, k, n$) is
▷ Precondition: $\mathbf{A[1] \leq \ldots \leq A[k-1]}$ and, for $\mathbf{k \leq j \leq n, A[k-1] \leq A[j]}$
    $min \leftarrow A[k]$; $minPosition \leftarrow k$
    ▷ Invariant: for $\mathbf{k \leq j \leq i-1}, min = \mathbf{A}[minPosition] \leq \mathbf{A[j]}$
    for $i \leftarrow k+1$ to $n$ do
        if $A[i] < min$ then
            $min \leftarrow A[i]$; $minPosition \leftarrow i$
        endif
    end do
    $A[k] \leftrightarrow A[minPosition]$
▷ Post-condition: $\mathbf{A[1] \leq \ldots \leq A[k]}$ and, for $\mathbf{k+1 \leq j \leq n, A[k-1] \leq A[j]}$
end SELECT
```

Give exact values for each of the following measures related to selection sort (worst case, as a function of n, the number of items being sorted): $C(n)$ = number of *key* comparisons;[1]; $oc(n)$ = number of other comparisons[2]; and $A(n)$ = number of assignments (not counting the implicit ones during procedure calls).

2. [worth 6 points] *Practice with recursion, i.e., the concept of reducing (an instance of) a problem to a smaller instance of the same problem.*

For each of the following problems describe a process by which an instance of the problem can be reduced to a smaller one. Your description can be a recursive function (in pseudocode with easily understandable primitive operations), a recursive mathematical definition, or a sufficiently precise English description, as long as the smaller instance and how it is manipulated to obtain a solution to the original are clear. An example of an English description for insertion sort might be:

> When sorting a list L, create a smaller list $L' = L - x$, i.e., L' is L with some element x removed.
> Sort L' and insert x into the result.

This would be followed up with a description of the insertion operation. Note that in order for insertion sort to be correct, x can be any element. The implementations of insertion sort choose either the first or the last for convenience and efficiency.

(a) sorting a list using selection sort (see above).

(b) determining whether an item x appears in a list L (items may be anything, not necessarily numbers or something else that can be sorted).

(c) finding the smallest number in a list of numbers L; by definition the smallest number in an empty list is ∞.

2 points for each part

[1] A key comparison compares (the keys of) two of the items being sorted.

[2] "**for** i $\leftarrow$ 1 **to** n **do** $\langle body \rangle$ **end do**" is the same as "i $\leftarrow$ 1; **while** i $\leq$ n **do** $\langle body \rangle$; i $\leftarrow$ i+1 **end do**"

3. [worth 7 points] *Understanding asymptotic notation*

For each of the following statements, prove that it is true or give a counterexample to prove that it is false. If you give a counterexample, you still have to prove that your example is, indeed, a counterexample.

(a). If $f(n)$ is $\mathrm{O}(g(n))$ then $g(n)$ is $\mathrm{O}(f(n))$.

(b). If $f(n)$ is $\Theta(g(n))$, then $\lg f(n)$ is $\Theta(\lg g(n))$.

(c). If $f(n)$ is $\Theta(g(n))$, then $2^{f(n)}$ is $\Theta(2^{g(n)})$.

(d). If $f(n)$ is $\mathrm{O}(g(n))$ then $g(n)$ is $\Omega(f(n))$.

(e). $f(n)$ is $\Theta(f(n/2))$.

(f). If $g(n)$ is $\mathrm{o}(f(n))$ then $f(n) + g(n)$ is $\Theta(f(n))$

(g). $f(n) + g(n)$ is $\Theta(\max(f(n), g(n)))$.

1 point for each part

4. [worth 6 points] *Bounding summations.*

Give Θ bounds for each of the following summations.

(a). $$\sum_{i=2}^{n} i^2 \lg\lg i$$

(b). $$\sum_{i=1}^{n} i^2 2^i$$

(c). $$\sum_{i=0}^{d} a^i n^i$$ Let a be any real number $\neq 0$, and $d \geq 0$ be an integer constant.

Your bound will involve d. Also, if d is odd, assume that $a > 0$.

2 points each part

5. [worth 10 points] *Solving recurrences.*

For each of the following recurrences, do one of two things. If the Master Theorem applies, give the Θ and the appropriate case. If not, solve the recurrence using any method you like, as long as it yields a rigorous proof that is easy for us to follow. In either case, assume that $T(n)$ is some constant c_0 when $n < s$ (you can choose s appropriately if the Master Theorem does not apply). The term "Master Theorem" here refers to the one in the textbook *not* the more general version found in, say, Wikipedia.

(a). $T(n) = 2T(n/2) + n \lg\lg n$

(b). $T(n) = 5T(n/4) + n \lg n$

(c). $T(n) = 2T(n/2) + n \lg^2 n$

(d). $T(n) = 2T(n/2) + \frac{n^2}{\lg^2 n}$

(e). $T(n) = 4T(n/2) + \frac{n^2}{\lg^2 n}$

(f). $T(n) = 27T(n/3) + n^3$

(g). $T(n) = 6T(n/2) + n^3$

10 points total, not evenly distributed.

6. [worth 5 points] *Applying analysis of divide and conquer to a concrete problem.*

The dominant operations in Strassen's matrix multiplication algorithm, particularly if the numbers being manipulated are multiple precision (take up more than one machine word), are scalar multiplications and scalar additions. Recurrences for these two operations are as follows.

$$\begin{array}{rcll} M(n) & = & 7M(n/2) & \text{note: no scalar multiplications are done before or after any recursive calls} \\ M(1) & = & 1 & \\ & & \text{for scalar multiplications} & \\ A(n) & = & 7A(n/2) + 10(n/2)^2 & \text{there are 10 additions of } n/2 \times n/2 \text{ matrices} \\ A(1) & = & 0 & \\ & & \text{for scalar additions} & \end{array}$$

Strassen's algorithm, even if you ignore bookkeeping overhead, is not a good choice for small matrices.

Your job is to figure out the exact value of n at which it makes sense to stop recursing and use the ordinary matrix multiplication algorithm instead. The number of scalar operations for the ordinary algorithm are n^3 multiplications and n^3 additions (actually there are $n^3 - n^2$ but let's not make things too complicated). To simplify your calculations, assume that multiplication takes twice as long as addition.

You should express the answer as simply as possible (you may want to assume s is a power of 2 to start with and come up with a more exact value later; this is not exactly kosher, but it works). *Do not convert logarithms into numbers, i.e., say* $\lg 7$ *instead of* 2.80735492206 *or* 2.81.

Rather than come up with a closed form solution for s you can leave it as finding the root of a polynomial (one of the terms will be a non-integer power). You should then use a sophisticated calculator, MatLab, or any program that works to interpolate (using, e.g., binary search or Newton's method) an exact value for s. The result should be rounded up to the next power of 2.

3 points for coming up with the right recurrences with a base case that is not 1 and evaluating them; 2 points for the remaining details.

For up to 10 points extra credit, do an experiment to determine the best value for n at which to stop the recursion. You can get up to 3 points for a detailed description of how such an experiment should be conducted. Submit your description, a writeup of your experiment, and your code with instructions for compiling and running it on a Unix-based system to the h1ex locker. It should be a zip archive with the name h1ex_uid.zip, where uid is your unity login id.

7. [worth 5 points] *Demonstrating abstract problem solving ability needed in CSC 505; also an interesting example of divide-and-conquer.*

You are interested in analyzing some hard-to-obtain data from two separate databases. Each database contains n numerical values – so there are $2n$ values in all – and you can assume that no two values are the same. You'd like to determine the median of this set of $2n$ values, which we will define here as the n-th smallest value.

However, the only way you can access these values is through *queries* to the databases. In a single query, you can specify a value k to one of the databases, and the chosen database will return the k-th smallest value that it contains. Since queries are expensive, you would like to compute the median using as few queries as possible.

Give an algorithm that finds the median value using at most $\mathrm{O}(\lg n)$ queries. Make your description as succint as possible while still being precise; as Einstein would say "as simple as possible but no simpler."

3 points for a correct algorithm (full credit only if description is simple); 2 points for correct analysis of your algorithm (if either your algorithm is correct or it has an analysis that is as interesting as the correct algorithm)