CRN 22094 CIS 621 Winter 1997

Midterm Exam Solutions

---

1. This loop computes the smallest Fibonacci number not smaller than the parameter c. We can prove it using the following invariant:

   For some i, $a = F(i)$ and $b = F(i-1) < c$.

   Indeed, assuming $c > 1$, this invariant is established by the initial assignments ($i = 1$). Execution of the body of the loop when $a < c$ assigns values $F(i)+F(i-1)$ to a and $F(i)+F(i-1)-F(i-1) = F(i) < c$ to b, maintaining the invariant. Upon termination, $b < c$ and $a \geq c$ are two consecutive Fibonacci numbers, proving partial correctness. Termination is proved by considering termination function $t = c - a$, which decreases after execution of the loop body, and implies fasity of the loop condition when $t \leq 0 \equiv \not a < c$.

2. Maintainig an auxiliary FIFO queue of sizes of merged files allows for a constant time selection of two smallest file sizes after each execution of the body of the greedy loop: "select two smallest files, merge them, delete the sizes and insert into the queue the size of the resulting file". The minimum size selection process requires comparing the leftmost element of the sequence of remaining original file sizes with the element at the front of the queue. The loop invariant supporting this is "every original record is in a file represented in the original list or the auxiliary queue, the latter in sorted order, $q_1 \leq q_2 \leq \ldots \leq q_k$, where q_k is not smaller than the sum of two smallest items in the remaining sequence and the queue combined, and the number of merges is less than n." The invariant is established initially, maintained because the next element on the queue is at least as large as the penultimate one, q_k, and smaller that the sum of two smallest sizes remaining. Upon exit, all records are in one file, the total number of files decreasing by one with every execution of the body of the loop. This number, minus one is the termination function.

3. Sorting twice takes $\mathcal{O}(n \log n)$ time. Combining solutions of subproblems takes linear time. The time complexity $T(n)$ is thus subject to recurence $T(n) = 2T(\frac{n}{2}) + n$ which has solution $\Theta(n \log n)$.

4. We have to prove a lemma that the median element m of all elements is always between the median elements in the arrays X and Y. m is also the midian of the subset from which an equal number of elements known to be not greater than m and those known to be not smaller than m are removed. Once we know this, our algorithm will compare (in constant time) the middle elements of both arrays, $a = X[n/2]$ and $b = Y[n/2]$ and recursively call itself for the pertinent subarrays of size $n/2$ each, unless $a = b$. The base case of $n = 1$ and the problem with integer division by 2 are trivial. The complexity $T(n)$ of the original problem is subject of the recursion $T(n) = 1 + T(n/2)$, with the desired solution of order $\log n$. The lemma follows from the fact that none of the n smallest elements of X and Y is greater than a or b, *i.e.*, the median $m \leq \max(a, b)$ and similarly $m \geq \min(a, b)$.

5. The cost of discovering failure is equal to the depth of the parent node of the corresponding empty subtree. Thus the cost function must add q_{i-1} and q_i to p_i in computing the weighted contribution of a_i. This involves using q_i as the weight of an empty subtree. If we define $c_{i,j}$ to include nodes $a_i, \ldots, a_{j-1}$, then we need $c_{i,i} = q_{i-1}$, redefining the weight function, $w_{i,i} = q_{i-1}$ (which reflects the above comment about the subproblem a_i: $w_{i,i+1} = q_{i-1} + q_i + p_i$).

6. The stated property guarantees the monotonicity of root positions for solutions at a given level: $r_{i,j} \leq r_{i+1,j+1}$. To solve all problems at level $l = j - i$ we need to try out particular rooting k at most twice ($k = r_{i,j}$ twice, for $c_{i-1,j}$ and for $c_{i,j+1}$, and once for all the other values of k.) Since each "try out" involves constant time work, that means linear time per level, or $\mathcal{O}(n^2)$ time complexity of the modified algorithm.

7. The generating function $A(x)$ of the sequence fulfills the recurrence $A(x) - a_0 - a_1 x = x(A(x) - a_0) - 2x^2 A(x)$. This gives $A(x) = \frac{1-x}{1-x-2x^2}$ which has a simple fraction expansion as $\frac{p}{1+x} + \frac{q}{1-2x}$. Comparing coefficients of x in the functional equation $1 - x = p(1 - 2x) + q(1 + x)$ (or substituting $x = -1$ and $x = \frac{1}{2}$ in it), we find $p = \frac{2}{3}$ and $q = \frac{1}{3}$. Since the fractions correspond to sequences $(-1)^n$ and 2^n, respectively, their linear combination gives the general term in our sequence:
$a_n = \frac{1}{3}(2(-1)^n + 2^n)$.

8. We observe that the sequence $s(m,n) = 1 + T(m,n)$ has the same recurrence as the binomial coefficients: $s(m,n) = s(m-1,n)+s(m,n-1)$, except for the initial conditions $s(0,n) = s(m,0) = 1$. One can see that these are the m^{th} numbers from the n^{th} row of Pascal triangle, and thus $s(m,n) = \binom{m+n}{m}$.

   An alternative (more elegant) solution involves a sequence of generating functions. Defining the generating function $S_m(x) = \sum_{n\geq 0} s(m,n)x^n$ and summing the sides of the recurrence multiplied by x^n, we get:

$$\begin{aligned} \sum_{n>0} s(m,n)x^n &= \sum_{n>0} s(m-1,n)x^n + \sum_{n>0} s(m,n-1)x^n \\ S_m(x) - s(m,0) &= S_{m-1}(x) - s(m-1,0) + xS_m(x) \end{aligned}$$

   which gives the equation

$$S_m(x) = \frac{1}{1-x} S_{m-1}(x) = \frac{1}{1-x}^{2} S_{m-2}(x) = \dots$$

   By definition, $S_0(x) = 1$, and thus $S_m(x) = (\frac{1}{1-x})^{m+1} = \sum_{n\geq 0} \binom{-m-1}{n}(-1)^n x^n$. Since $\binom{-r}{k} = \binom{r+k-1}{k}$, the coefficients are $s(m,n) = \binom{m+n}{n}$.

   Using Stirling approximation of $n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$, we get the asymptotic growth rate

$$f(n) = T(n,n) = s(n,n) - 1 \in \mathcal{O}(\binom{2n}{n}) = \mathcal{O}(\frac{(2n)!}{(n!)^2}) = \mathcal{O}(2^{2n}).$$