Comprehensive Exam: Algorithms and Concrete Mathematics Autumn 2004

This is a one hour closed-book exam and the point total for all questions is 60.

In questions that ask you to provide an algorithm, please explain the algorithm in words and diagrams, no need to write code or pseudo code. Also, for any algorithm, state and prove its running time. No credit will be given for exponential-time algorithms. Polynomial but slow algorithms will get some partial credit. Amount of credit will depend on how much slower they are compared to what is achievable using the knowledge in the reading list.

For full credit, the answers should be short and precise. Long and convoluted answers will not get full credit even if they are correct.

1. **[14 pts]** Please answer "true" or "false" to each one of the following questions. Correct answers will give you **(2 pts)** each while wrong answers will reduce your score by **(2 pts)** each.

a
- $2n = O(n^2)$

  Answer: TRUE

b
- $n! = o(n^n)$

  Answer: TRUE

c
- $(\log n)^{10} = \Omega(\sqrt[10]{n})$

  Answer: FALSE

d
- Any graph with n nodes and m edges has a spanning tree with $n-1$ edges.

  Answer: FALSE, graph can be disconnected

e
- Assume that T is a minimum cost spanning tree for a weighted graph $G = (V, E)$ with positive weights $w : E \to R^+$. Will T be still a minimum cost spanning tree if instead of w we use $\log w$?

  Answer: TRUE, relative order of weights does not change

f
- You are given a graph $G = (V, E)$ with positive lengths on edges and a shortest path P from $v \in V$ to $u \in V$. Next, lengths are transformed by uniformly adding 1 to the length of each edge. Will P be still a shortest path from v to u ?

  Answer: FALSE

g
- Let A be an array of n distinct numbers. The goal is to rearrange the numbers so that $\forall i$ where $0 < i < n/2$ we have $A[i] < A[2i]$. It is true that this operation requires $\Omega(n \log n)$ time in the comparison model?.

  Answer: FALSE

2. **[15 pts]** Let u and v be two n-bit numbers, where for simplicity n is a power of 2. The traditional multiplication algorithm requires $\Theta(n^2)$ operations. A divide-and conquer based algorithm splits the numbers into two equal parts, computing the product as

$$uv = (a2^{n/2} + b)(c2^{n/2} + d) = ac2^n + (ad + bc)2^{n/2} + bd$$

Here, a and c are the higher order bits, and b and d are the lower order bits of u and v respectively. Multiplications are done recursively, except multiplication by 2^i for integer i, which is a shift and we assume takes $\Theta(i)$ time. We also assume that addition/subtraction of i-bit numbers takes $\Theta(i)$.

- **[6 pts]** Write a recurrence for the running time of this algorithm as stated. Solve the recurrence and determine the running time.

  Answer: $T(n) = 4T(n/2) + \Theta(n), T(1) = 1$. This means that $T(n) = \Theta(n^2)$, no improvement over brute-force approach.

- **[9 pts]** Note that $ad + bc$ can be computed as $(a + b)(c + d) - ac - bd$. Why is this advantageous? Write and solve a recurrence for the running time of modified algorithm.

  Answer: There is one less multiplication per iteration, resulting in $T(n) = 3T(n/2) + \Theta(n), T(1) = 1$. This means that $T(n) = \Theta(n^{\log_2 3}) = o(n^2)$.

  Note that, strictly speaking, some of the multiplications are of $n+1$-bit numbers and not n-bit numbers. Our calculations are still correct since to multiply two $n+1$-bit numbers, one can execute a single multiplication of two n-bit numbers plus $\Theta(n)$ extra work.

3. **[16 pts]** Given a set of items $x_1, x_2, \ldots, x_n$ where item x_i has a non-negative weight $w(x_i)$, the goal is to find a subset of items with maximum total weight under the constraint that if item x_i is chosen, then you are forbidden to chose either x_{i+1} or x_{i-1} (forbidden to choose the "neighbors" of x_i). For example, you can choose x_5, x_8, and x_{10}, but you are not allowed to choose x_5, x_6, x_9, x_{11}. Design an efficient algorithm to solve this problem, prove correctness and analyze running time. For simplicity, your algorithm should just produce the value of the maximum total weight and does not need to list the items that you are choosing to achieve this weight.

Answer: Let W_i denote the maximum weight that we can collect by choosing only among items numbered 1 through i. Clearly, $W_1 = x_1$. To compute W_{i+1} we need to consider two cases: either x_{i+1} participates in the optimum solution when the problem is restricted to items 1 through $i+1$, or not. If it does not, then the optimum solution is the same as optimum for problem when input is restricted to items 1 through i, i.e. in this case $W_{i+1} = W_i$. Alternatively, if x_{i+1} participates, then x_i cannot participate. We claim that $W_{i+1} = W_{i-1} + w(x_{i+1})$ in this case. To prove this claim, assume that there is an optimum for the restricted problem (items 1 through $i+1$) that includes $i+1$ and that is that produces weight larger than $W_{i+1} = W_{i-1} + w(x_{i+1})$. Since x_{i+1} participates, x_i does not, and thus removing x_{i+1} from this solution we get a legal solution for the problem restricted to items 1 through $i-1$ with weight higher than W_{i-1}, which contradicts our assumption.

To summarize, $W_{i+1} = \max\{W_i, W_{i-1} + w(x_{i+1})\}$. This takes constant time per iteration. With n iterations, the total is $\Theta(n)$. Observe that one does not need an extra array of n elements to hold all the computed W_i values since at each iteration it is sufficient if we store only the latest two values, W_i and W_{i-1}.

4. **[15 pts]** Consider a section of highway with numbered exits. You are given information about n cars that you suspect of speeding. Car i will enter the highway at point s_i and exits at $t_i > s_i$. The goal is to place minimum number of photo-radar units along the highway to ensure that you will catch (photograph) all of the n speeding cars. (Assume that once the photo-radar is placed, you cannot move it.)

   We say that paths of two cars i and j do not intersect (disjoint) if $t_i < s_j$ or $t_j < s_i$. Observe that if there is a set of k of cars that have disjoint paths, then you will need at least k photo-radars.

   Let k^* be the size of the maximum-size set of cars with disjoint paths. The observation above implies that you will need at least k^* photo-radars. Prove that in fact k^* photo-radars is sufficient, i.e. there exists a placement of k^* photo-radars that will photograph all of the cars.

   Answer: Car i corresponds to a closed interval (s_i, t_i). We will prove the claim by giving an algorithm that finds k disjoint intervals and k places for photo-radars such that each interval has at least one photo-radar in it. By definition, $k^* \geq k$. Moreover, since k photo-radars are sufficient, $k^* \leq k$. Thus, $k^* = k$, proving the claim.

   The algorithm starts by ordering all intervals in increasing order by t_i. Pick the first interval in this order (s_{i_1}, t_{i_1}), mark it, and delete it from the set. Place photo-radar at t_{i_1} and delete all intervals that intersect with (s_{i_1}, t_{i_1}). Repeat until all intervals are deleted. Result is k marked intervals and k photo-radar placements.

   First we claim that, by construction, all k chosen intervals are disjoint. Next we claim that the placed photo-radars will photograph all cars, i.e. for every interval there is a photo-radar that is placed at one of the points in the interval. Observe that each iteration results in placement of a photo-radar and deletion of some intervals. We claim that all deleted intervals in an iteration are "touched" by the photo-radar placed in this iteration. This is trivially true for the interval marked in this iteration. Consider an interval (s_j, t_j) deleted during iteration q. By construction, $t_j > t_{i_q}$. The only way the photo-radar at t_i does not touch it is if $s_j > t_{i_q}$. But in this case we will not delete this interval.

   One can implement this algorithm to run efficiently, but this is irrelevant to the question, since all we needed was a proof of the claim.