CIS507: Design & Analysis of Algorithms
Pre-midterm Review, Spring 2014

1 Orders of growth, recurrences

For each pair of functions below, indicate whether $f(n) = \mathcal{O}(g(n))$, $f(n) = \Omega(g(n))$, or $f(n) = \Theta(g(n))$. Assume $n \geq 1$, $f(n) \geq 1$ and $g(n) \geq 1$.

1. $f(n) = (n+5)^2$ versus $g(n) = n^2$
2. $f(n) = n!$ versus $g(n) = n^n$
3. $f(n) = 2^n$ versus $g(n) = 2^{n^2}$
4. $f(n) = 2^{2^n}$ versus $g(n) = 2^{n^2}$
5. $f(n) = (\sqrt{2})^{\lg n}$ versus $g(n) = 2\sqrt{n}$. (lg is base 2 logarithm)
6. $f(n) = \binom{n}{n/2}$ versus $g(n) = 2^n$.

ANSWER:

1. $f(n) = \Theta(g(n))$. Observe that $f(n) = (n+5)^2 = n^2 + 10n + 25$. Now:
   - Choose $c_1 = 1$, then $n^2 + 10n + 25 \geq n^2$ for all $n \geq 1$. Therefore $f(n) = \Omega(g(n))$.
   - Choose $c_2 = 10$, then $n^2 + 10n + 25 \leq 10n^2$ for all $n \geq 10$. Therefore $f(n) = \mathcal{O}(g(n))$.
2. $f(n) = \mathcal{O}(g(n))$. Because $n! = n(n-1)(n-2)\ldots 1 \leq n^n$.
3. $f(n) = \mathcal{O}(g(n))$. Taking logarithm of both, we compare n with n^2.
4. $f(n) = \Omega(g(n))$. Taking logarithm of both, we compare 2^n versus n^2.
5. $f(n) = (\sqrt{2})^{\lg n} = n^{\lg \sqrt{2}} = n^{\lg 2^{1/2}} = n^{\frac{1}{2}\lg 2} = n^{\frac{1}{2}} = \sqrt{n}$, so $f(n) = \Theta(g(n))$.
6. Using Stirling's approximation: $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$, we obtain
$$f(n) = \frac{n!}{((n/2)!)^2} \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \cdot \frac{1}{\pi n\left(\frac{n}{2e}\right)^n} = \frac{1}{\sqrt{\pi n/2}} \cdot 2^n,$$
so $f(n) = o(g(n)$.

2 Sorting and hashing

For each of the following, circle the correct answer.

1. Suppose the running time of an algorithm follows the recurrence $T(n) = T(\frac{n}{4}) + T(\frac{n}{2}) + \Theta(n)$. What is the asymptotic running time?
   (i) $\Theta(n)$ (ii) $\Theta(n^{0.75})$ (iii) $\Theta(\log n)$
   (iv) $\Theta(n^p)$, where p satisfies:
   $(\frac{1}{2})^p + (\frac{1}{4})^p = 1$

2. Suppose that we use an open-addressed hash table of size m to store $n \leq m/2$ items. Then under the assumption of uniform hashing, the probability that the ith insertion requires strictly more than k probes is at most
   (i) $\Theta(1/n^2)$ (ii) 2^{-n} (iii) $\Theta(1/k^2)$ (iv) 2^{-k}

3. Suppose instead of dividing in half at each step of the mergesort, you divide into thirds, sort each third, and finally combine all of them using a three way merge. What is the overall running time of this algorithm? (*Hint:* Note that the merge step can still be implemented in $\mathcal{O}(n)$ time.)
   (i) $\mathcal{O}(n^3)$ (ii) $\mathcal{O}(n^2 \lg n)$ (iii) $\mathcal{O}(n \lg n)$ (iv) $\mathcal{O}(n^3 \lg n)$

4. Suppose you are given k sorted arrays, each with n elements, and you want to combine them into a single array of kn elements. Consider the following approach. Using the merge subroutine, you merge the first 2 arrays, then merge the 3rd given array with this merged version of the first two arrays, and so on until you merge in the final (kth) input array. What is the time taken for this strategy, as a function of k and n?
   (i) $\Theta(n^2)$ (ii) $\Theta(nk^2)$ (iii) $\Theta(kn^2)$ (iv) $\Theta(kn)$

5. Consider a sequence of operations on a dynamic data structure, where an insert requires an actual cost of 2, while delete requires an actual cost of 5. In a sequence of 100 insertions and 200 deletions, the amortized cost of an operation (which can be either insert or delete) is
   (i) 7 (ii) 4 (iii) 3.5 (iv) 3

ANSWER:

1. (i) $\Theta(n)$. This follows by the recursion tree method since $\frac{1}{4} + \frac{1}{2} < 1$.

2. (iv) 2^{-k}. This follows from the proof of Theorem 11.6 in the text book (with $\alpha = 1/2$).

3. (iii) $\mathcal{O}(n \lg n)$. There is still a logarithmic number of levels, and the overall amount of work at each level is still linear.

4. (ii) $\Theta(nk^2)$. First merge costs $2n$, second costs $3n$, etc. Total is $2n + 3n + \cdots + kn = n(2 + 3 + \cdots + k) = n(\frac{k(k-1)}{2} - 1) = \Theta(nk^2)$

5. (ii) 4. The amortized cost is $(2 \times 100 + 5 \times 200)/300 = 4$.

3 Random Coloring

You are given a graph $G = (V, E)$ with nodes (vertices) V and edges E between them. You have 3 colored pens, and you are asked to color all nodes.

For every pair of nodes a and b such that $\{a, b\} \in E$, the edge $e = \{a, b\}$ is *satisfied* if a and b have different colors. For each edge that is satisfied, you will be given \$1. Theoretically, your maximum reward is at most $\$|E|$.

You like money, but you are also lazy! So you decide to color each node randomly (i.e. for each node, choose a color uniformly at random, independent of the colors of other nodes). Compute how much you expect to make as a function of $|E|$. Show your full derivation.

Hint: Use indicator random variables over edges.

ANSWER:

Define indicator random variable X_e for each edge $e = \{a, b\}$, which is 1 if a and b have different colors.

$$X_e = \begin{cases} 1 & \text{if nodes } a \text{ and } b \text{ have different colors} \\ 0 & \text{otherwise} \end{cases}$$

Then, for any edge, there are 9 ways to color its two ends, each of which appears with the same probability, and 3 of them will deduct your reward. So we expect $\frac{6}{9} = \frac{2}{3}$ to be satisfied on average:

$$E[X_e] = Pr\{e \text{ is satisfied}\} = \frac{6}{9} = \frac{2}{3}$$

Let Y be a random variable denoting the number of satisfied edges. We have:

$$Y = \sum_{e \in E} X_e$$

Take expectation of both sides:

$$E[Y] = E\left[\sum_{e \in E} X_e\right]$$

By linearity of expectation:

$$E[Y] = \sum_{e \in E} E\big[X_e\big] = \sum_{e \in E} \frac{2}{3} = \frac{2}{3}|E|$$

4 Maximum independent set of intervals

You are given a set of events to schedule; no two events can be scheduled at the same time. Each event you schedule will earn you some amount of money. Design an efficient algorithm that finds a schedule that maximizes your earnings.

Hint: represent each event as an interval.

ANSWER: Let the given set of events be represented as intervals $I_1, \ldots, I_n$, where $I_i = [b_i, e_i]$, with weight w_i. Then we are asked to find a maximum-weight collection of disjoint intervals.

We will use dynamic programming. Clearly all interesting events happen at points in time that are defined by a start or an end of an interval. Sort this set of endpoints $\{b_1, e_1, \ldots, b_n, e_n\}$ into an array X of size $2n$ (for simplicity we assume all points are distinct; the algorithm can be easily modified to deal with the general case).

Define MWIS[i] to be the maximum weight subset of events that can be scheduled in the period from the start to time $X[i]$. Observe (by a cut and paste argument) that the optimal substructure property holds.

We initialize MWIS[0] := 0. Then define

$$\text{MWIS}[i] := \begin{cases} \text{MWIS}[i-1] & \text{if no interval ends at } X[i] \\ \max\{\text{MWIS}[i-1], \text{MWIS}[X^{-1}(b_j)] + w_j\} & \text{if interval } I_j \text{ ends at } X[i], \end{cases}$$

where $X^{-1}(b_j)$ is the index in X of b_j. Taking the max above corresponds to the choice between not taking the interval I_j into the solution, or taking it: if we take it, then the solution increases by w_j, but we can only combine it with the intervals up to b_j.

By storing (during construction of X) for each point $X[i]$ the interval that ends at it (if any), and for each interval I_j the index i such that $X[i] = b_j$, we can insure that the total running time of the procedure is $O(n \log n)$ (needed for sorting the endpoints). To find an optimal set of intervals, we store, in another table, the indices of the intervals achieving the maximum above, and traverse this table backward.