CS 170 Algorithms
Spring 2015 Christos Papadimitriou and Prasad Raghavendra **HW 1**

1. (5 pts.) Problem 1

I understand the course policies.

2. (15 pts.) Problem 2

(a) $f = \Omega(g)$; both are polynomials and $1.5 > 1.3$.

(b) $f = \Theta(g)$; $2^{n-1} = \frac{1}{2}2^n$, so f and g are within a factor 2 of each other.

(c) $f = \Omega(g)$; $n^{1.3\log n} > n^{\log n}$, and $n^{\log n}$ grows faster than any polynomial in n.

(d) $f = \Omega(g)$; $\lim_{n\to\infty} \frac{3^n}{n2^n} = \lim_{n\to\infty} \frac{\left(\frac{3}{2}\right)^n}{n} = \infty$, so f grows much faster than g.

(e) $f = O(g)$; $\log(n)$ to any constant power grows much more slowly than n^c for any constant $c > 0$.

(f) $f = \Omega(g)$; $n = 2^{\log n}$ and $(\log n)^{\log\log n} = 2^{(\log\log n)^2}$, so f grows faster than g since $\log n$ grows faster than $(\log\log n)^2$.

(g) $f = O(g)$; $\frac{2^n}{n!} = \prod_{i=1}^{n} \frac{2}{i} < 2\left(\frac{2}{3}\right)^{n-2}$ which goes to zero as n increases.

(h) $f = O(g)$; $\log(e^n) = n\log(e) = O(n)$, so $f(n)$ grows more slowly than $g(n) = n\log n$.

(i) $f = \Theta(g)$; In both f and g the linear term n dominates the other term, so both f and g grow as $\Theta(n)$.

(j) $f = \Theta(g)$; The term $5n$ dominates $\sqrt{n}$ in f, and n dominates $\log n$ in g, so again both grow as $\Theta(n)$.

3. (20 pts.) Problem 3

This problem is about finding the contiguous sub-array or subsequence of maximum sum. The proposed algorithm divides the array in two halves and computes first the subsequence C of maximum sum that crosses the mid point. This is done in $O(n)$ time by checking every sequence starting and ending at the midpoint, and concatenating together the one of maximum sum that ends at the midpoint with the one of the maximum sum that starts at midpoint. Then the algorithm recursively computes the maximum sum subsequence in each of the two halves, call them L and R, and outputs the length of the longest among L, C and R. (Note that with some extra bookkeeping we can output also the starting and ending indeces of the the maximum sum subsequence.)

To bound the running time we use the Master Theorem. Using the notation from the book, the running time is given by $T(n) = 2T(n/2) + O(n)$, since the problem is reduce to two smaller sub-problems each of size $n/2$ (hence $a = 2$ and $b = 2$), and it takes and additional $O(n)$ steps to produce the output (hence $d = 1$). The Master Theorem implies that the running time is $O(n\log n)$ since $d = \log_b a$.

4. (20 pts.) Problem 4 For each j in $\{1,\ldots,n\}$ define the hash function $h_j(x) = j$ and consider the set of hash functions $H = \{h_j \mid j \in \{1,\ldots,n\}\}$. Note that to choose a uniform random hash function $h \in H$ we can simply choose a uniform random $j \in \{1,\ldots,n\}$ and return the hash function h_j. Thus for a fixed data item x we have

$$\Pr[h(x) = i] = \Pr[h_j(x) = i] = \Pr[j = i] = \frac{1}{n}$$

where the first probability is for a uniform random $h \in H$, and the second two are for a uniform random $j \in \{1,\ldots,n\}$. Hence H satisfies the alternative property given.

However, note that no matter which h_j we randomly select, every one of the m data items gets hashed to the bucket j. So, for every random choice of $h \in H$ there are always m elements colliding in some bucket. Thus, the average number of collisions in some bucket is m.

5. (20 pts.) Problem 5

(a) We show something slightly more general. Namely we will prove for any $i \in \mathbb{N}$, $\sum_{k=1}^{n} k^i = \Theta(n^{i+1})$; this problem is just the $i = 2$ case. Note that since $k \leq n$ every term in the sum is at most n, so

$$\sum_{k=1}^{n} k^i = 1^i + \cdots + n^i \leq n^i + \cdots + n^i = \sum_{k=1}^{n} n^i = n^{i+1}.$$

We do something similar for the lower bound. The only additional idea is that we only look at the second half of the sum. The smallest element in the second half of the sum correspond to $k = n/2$ (assuming without loss of generality that n is even). Then,

$$\sum_{k=1}^{n} k^i \geq \sum_{k=n/2}^{n} k^i \geq \sum_{k=n/2}^{n} (n/2)^i = 2^{-i-1} n^{i+1}.$$

Note. Alternatively, this problem can also be solved by remembering that $\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$.

(b) The main idea for both upper and lower bound is that for any k there exist i such that $\frac{1}{2^{i+1}} \leq \frac{1}{k} \leq \frac{1}{2^i}$. Note that for several k's the value of i is the same, so we can replace several $1/k$'s by the same inverse power of 2 to obtain a more manageable sum.

More precisely, for the upper bound we replace each $1/k$ with $1/2^i$, where 2^i is the largest power of 2 smaller than or equal to $1/k$. For simplicity assume n is a power of 2, i.e. $n = 2^\ell$ for some $\ell \in \mathbb{N}$. (Note that this assumption has no effect on the order of the sum.) Therefore,

$$\begin{aligned}
\sum_{k=1}^{n} \frac{1}{k} &= 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \leq 1 + \frac{1}{2} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{8} + \cdots \\
&\leq 1 + \sum_{k=2}^{3} \tfrac{1}{2} + \sum_{k=4}^{7} \tfrac{1}{4} + \cdots + \sum_{k=n/2}^{n} \frac{1}{n/2} = 1 + 2\tfrac{1}{2} + 4\tfrac{1}{4} + \cdots + 2^{\ell-1} \frac{1}{2^{\ell-1}} = \ell = O(\log n).
\end{aligned}$$

We can do the same for the lower bound, except we replace $1/k$ with $1/2^i$, where 2^i is the smallest power of 2 larger than k.

Note. This problem can also be solved by integrating: $\sum_{k=1}^{n} \frac{1}{k} = \int_1^{n+1} \left\lceil \frac{1}{x} \right\rceil dx = \Theta(\log n)$.

(c) Observe that

$$n! = 1 * 2 * 3 \cdots * n \leq n * n * n \cdots * n \leq n^n$$

and assuming n is even (without loss of generality)

$$n! = 1 * 2 * 3 \cdots * n \geq n * (n-1) * (n-2) \cdots * (n - n/2) \geq \left(\frac{n}{2}\right)^{\frac{n}{2}+1}.$$

Hence $\left(\frac{n}{2}\right)^{\frac{n}{2}} \leq n! \leq n^n$. Then,

$$\frac{n}{2} \log\left(\frac{n}{2}\right) \leq \log(n!) \leq n \log n.$$

(d) By the formula for the sum of a partial geometric series, for $c \neq 1$: $S(k) := \sum_{i=0}^{k} c^i = \frac{1-c^{k+1}}{1-c}$. Thus,

- If $c > 1$, for sufficiently large k, $c^{k+1} > c^{k+1} - 1 > c^k$. Dividing this inequality by $c-1$ yields: $\frac{c}{c-1}c^k > S(k) > \frac{1}{c-1}c^k$. Thus, $S(k) = \Theta(c^k)$, since $\frac{1}{c-1}$ is constant.
- If $c = 1$, then every term in the sum is 1. Thus, $S(k) = k+1 = \Theta(k)$.
- If $c < 1$, then $\frac{1}{1-c} > \frac{1-c^{k+1}}{1-c} = S(k) > 1$. Thus, $S(k) = \Theta(1)$.

6. (20 pts.) Problem 6

As suggested in the hint we generalize the algorithm from class for computing the n-th Fibonacci number. Let $\vec{f}_n$ be the $1 \times k$ vector $(F_n, ..., F_{n-k+1})$. We would like to find a $k \times k$ matrix A such that $\vec{f}_{n+1} = A\vec{f}_n$. Then to compute F_n we just compute $\vec{f}_n$ and output its first coordinate.

Let $r_1, \cdots, r_k$ be the rows of A. We want $r_1 \cdot \vec{f}_n = F_{n+1}$, so r_1 is just the all 1's vector. For all other $2 \le i \le k$, we want $r_i \cdot \vec{f}_n = F_{n-i+2}$, so r_i is 1 in the $(i-1)$-th coordinate and 0 everywhere else. Hence,

$$A = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}.$$

Therefore, $\vec{f}_n = A\vec{f}_{n-1} = A^2\vec{f}_{n-2} = \cdots = A^{n-k}\vec{f}_k$. To compute A^{n-k}, when $n-k$ is even, we compute $A^{\frac{n-k}{2}}$ and then square it to obtain A^{n-k}. If $n-k$ is odd, we compute $A^{\lfloor \frac{n-k}{2} \rfloor}$, square it and multiply it one more time by A to obtain A^{n-k}.

If $T(n)$ is the number of $k \times k$ matrix multiplications performed by this algorithm to compute F_n, then $T(n) = T(n/2) + O(1)$. The Master Theorem with $a = 1$, $b = 2$ and $d = 0$ implies that $T(n) = O(\log n)$.

Rubric:

For any part of a problem with 10 points or more, use the following guideline for partial credit:

- All correct: 100%
- A few minor errors: 80%
- On right track, but many/major errors: 50%
- Had some ideas, but wasn't on right track: 20%
- No answer: 0%

Redemption You can redeem points on all problems this week.

Problem 1

Assign full credit if "I understand the course policies." is written.

Problem 2

Each part is 1.5 pts: 1pt for identifying right relation between f and g and 0.5 pts for any reasonable explanation.

Problem 3

Proposed solution 1: Identify recurrence for running time and apply Master Theorem.

- 6 pts for identifying correct recurrence.
- 4 pts for identifying that the Master Theorem is applicable in this setting.
- 10 pts for correct application of the Master Theorem.

Proposed solution 2: Identify recurrence for running time and apply a different method to bound close form.

- 6 pts for identifying correct recurrence.
- 14 pts for analyzing (correctly) the recurrence using a different method

Proposed solution 3: Bound the running time using a different method.

- Assign 20 pts if you are completely sure that your method is correct.

Problem 4

- 5 pts for coming up with a family that satisfies the alternative property.
- 15 pts for proving that the number of collisions is m on average.

Problem 5

Each part is 5 pts. There are many correct ways of proving each of these facts. Give yourself full credit if you are completely sure that your proof is correct. Otherwise, look at the proposed solutions and see if you can figure if there is something wrong.

Problem 6

- 5 pts for Identifying the right $k \times k$ matrix.
- 5 pts for the correct description of how to use the matrix to obtain the n-th number.
- 8 pts for the correct algorithm for computing A^n with only $O(\log n)$ matrix multiplications.
- 2pts for getting all the exponents, indexes and other details right. (5