EECS 376: Foundations of Computer Science **Due 8:00pm, May 16**
University of Michigan, Spring 2024

Homework 1

(accepted until 9:59 pm, no credit after)

This homework has 10 questions, for a total of 100 points and 10 extra-credit points.
Unless otherwise stated, each question requires *clear*, *logically correct*, and *sufficient* justification to convince the reader.
For bonus/extra-credit questions, we will provide very limited guidance in office hours and on Piazza, and we do not guarantee anything about the difficulty of these questions.
We strongly encourage you to typeset your solutions in LaTeX.
If you collaborated with someone, you must state their name(s). You must *write your own solution* for all problems and *may not use any other student's write-up.*

(0 pts) 0. **Before you start; before you submit.**

(a) Carefully read Handout 1 before starting this assignment, and apply it to the solutions you submit.

(b) If applicable, state the name(s) and uniqname(s) of your collaborator(s).

Solution: None.

1. **Practice with asymptotics ("big-Oh, big-Omega, big-Theta").**

For the following pairs of functions, state with justification whether or not each of the following hold: $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, $f(n) = \Theta(g(n))$.

You can find the definitions of asymptotic notations (O, Ω, Θ) in Handout 0.

(5 pts) (a) $f(n) = n^3 + 2n + 8$, $g(n) = 4n^3$.

Solution:
1. $f(n) = O(g(n))$.
Consider $c = 1$ and $n_0 = 2$, then $cg(n) - f(n) = n(3n^2 - 2) - 8$. When $n = n_0$, $cg(n) - f(n) = 20 - 8 = 12$. Since $cg(n) - f(n)$ increases as n grows when $3n^2 - 2 > 0$, $f(n) \le cg(n)$ when $n \ge n_0$. Therefore $f(n) = O(g(n))$.

2. $f(n) = \Omega(g(n))$.
Consider $c = \frac{1}{4}$ and $n_0 = 1$, then $f(n) - cg(n) = n^3 + 2n + 8 - n^3 = 2n + 8$. When $n >= 1$, $f(n) - cg(n) > 0$, $f(n) > cg(n)$. Therefore $f(n) = \Omega(g(n))$

3. $f(n) = \Theta(g(n))$.
Since $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, $f(n) = \Theta(g(n))$ by definition.

(5 pts) (b) $f(n) = (2 + (-1)^n)n^2 + 1$, $g(n) = n^2$.

Solution:
1. $f(n) = O(g(n))$.
Consider $c = 4$ and $n_0 = 1$, $f(x) - cg(x) = (-1)^n n^2 - 2n^2 + 1 \le -n^2 + 1$. So when $n \ge 1$, $-n^2 + 1 \le 0$, $f(n) \le cg(n)$. Therefore $f(n) = O(g(n))$.

2. $f(n) = \Omega(g(n))$.

Consider $c = 1$ and $n_0 = 1$, $f(x) - cg(x) = (-1)^n n^2 + n^2 + 1 \geq 1$. So $f(n) \geq cg(n)$. Therefore $f(n) = \Omega(g(n))$.

3. $f(n) = \Theta(g(n))$.
Since $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, $f(n) = \Theta(g(n))$ by definition.

(5 pts) (c) $f(n) = \log_2(n^{10})$, $g(n) = \log_{10}(n^2)$.

Solution: $f(x) = 10\log_2(n)$, $g(x) = 2\log_{10}(n) = 2\frac{\log_2(n)}{\log_2(10)} = \frac{2}{\log_2(10)}\log_2(n)$
Therefore $f(n) = \Omega(g(n))$, $f(n) = O(g(n))$ and $f(n) = \Theta(g(n))$ since we can choose $c = 5\log_2(10)$ to let $f(x) = cg(x)$.

(5 pts) (d) $f(n) = 2^{10n}$, $g(n) = n^{10} \cdot 10^{2n}$.

Solution: $f(x) = 2^{10n} = (2^{10})^n = 1024^n$, $g(x) = n^{10} \cdot 10^{2n} = n^{10} \cdot (10^2)^n = n^{10} \cdot 100^n$.

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{n^{10} \cdot 100^n}{1024^n} = \lim_{n\to\infty} (\frac{25}{256})^n \cdot n^{10} = \lim_{n\to\infty} \frac{n^{10}}{(\frac{256}{25})^n} = 0$$

by the L'Hôpital's Rule.
Therefore $f(n) \neq O(g(n))$, $f(n) = \Omega(g(n))$.
And $f(n) \neq \Theta(g(n))$ since not both expressions are true.

(5 pts) 2. **Comparing asymptotic running times.**

Suppose that Algorithm X has running time $T_X(n) = \Theta(n)$, Algorithm Y has running time $T_Y(n) = \Theta(n^2)$, and Algorithm Z has running time $T_Z(n) = O(n^2 \log n)$.

As usual, all running times are stated in terms of the *worst case* for inputs of size n. That is, $T_X(n)$ is the *maximum* number of steps for which X runs, taken over all inputs of size n (and similarly for T_Y, T_Z). Choose all claims that are necessarily true.

- ☐ On every input, X runs faster than Y.
- ☐ For all large enough n, X runs faster than Y on every input of size n.
- ■ **For all large enough n, there is an input of size n for which X runs faster than Y.**
- ■ **For all large enough n, there is an input of size n for which Y runs faster than Z.**

3. **EECS 376 lover.**

Consider the following algorithm:

```
1: function EECS376Lover($n, k$)        ▷ $n$ is a positive integer, and $k \in \{1, 2, \dots, n\}$
2:     for $i = 1, 2, \dots, k$ do
3:         for $j = 1, 2, \dots, n - k$ do
4:             Print("I love EECS 376!")
```

(1 pt) (a) Identify the value of k that induces the **most** `"I love EECS 376!"` printed by the algorithm.

(2 pts) (b) Based on your answer in (a), give the **tightest correct asymptotic** (big-O) bound, as a function of n, on the number of `"I love EECS 376!"` printed by the algorithm.

(2 pts) (c) Is the algorithm *efficient*, i.e., runs in at most polynomial time with respect to the input size? Briefly explain your answer.

Solution:

(a) Fix n, for any $1 \le k \le n$, the number of `"I love EECS 376!"` printed by the algorithm is $k(n-k) = -(k-\frac{n}{2})^2 + \frac{n^2}{4}$.
Therefore the maximum is reached when k is closest to $\frac{n}{2}$.
So the value of k that induces the most `"I love EECS 376!"` printed by the algorithm is $k = \lfloor \frac{n}{2} \rfloor$ and $k = \lceil \frac{n}{2} \rceil$. (If n is even, it is just $\frac{n}{2}$; if n is odd, then it is $\frac{n \pm 1}{2}$)

(b) Based on the answer in (a), the tightest correct asymptotic bound is $O(n^2)$ since the number of `"I love EECS 376!"` printed $= k(n-k) = -(k-\frac{n}{2})^2 + \frac{n^2}{4} \le \frac{n^2}{4}$

(c) The algorithm is efficient since ts running time is within $O(n^2)$, which is polynomial time with respect to the input size.

(10 pts) 4. **Power of induction.**

Consider the following algorithm to compute a^b, where a and b are some integers.

```
function Pow($a, b$)
    if $b = 0$ then
        return 1
    if $b$ is even then
        return $(\text{Pow}(a, b/2))^2$
    else
        return $a \cdot (\text{Pow}(a, (b-1)/2))^2$
```

Prove that the algorithm is correct by induction.

Solution: Proof:
Let a, b be an arbitrary integer(b is positive).
We prove the correctness of the algorithm by induction on b.
Base case: $b = 0$. The algorithm returns $1 = a^0$, which is correct.

Inductive step: Assume that the algorithm is correct for all $b \le k$, where $k \ge 0$.
Now we show that the algorithm is correct for $b = k+1$.
There are only two cases to consider:
Case 1: $k+1$ is even, i.e. $k+1 = 2m$ for some integer m.

Then the algorithm returns $(\text{Pow}(a,(k+1)/2))^2 = (\text{Pow}(a,m))^2 = a^m \cdot a^m = a^{2m} = a^{k+1}$, which is correct.

Case 2: $k+1$ is odd, i.e. $k+1 = 2m+1$ for some integer m. So $k = 2m$.
Then the algorithm returns $a \cdot (\text{Pow}(a,(k+1-1)/2))^2 = a \cdot (\text{Pow}(a,2m/2))^2 = a \cdot (\text{Pow}(a,m))^2 = a \cdot a^m \cdot a^m = a^{2m+1} = a^{k+1}$, which is correct.
Since Both cases are correct, we have proved the induction step.

Therefore, by induction, the algorithm is correct for all $b \geq 0$.

5. **Pigeons and the Contra-.**

Recall the Pigeonhole Principle, which states "If n pigeons are placed into m pigeonholes, where $n > m$, then at least one pigeonhole contains more than one pigeon."

Prove the Pigeonhole Principle using

(5 pts) (a) proof by contradiction, and

(5 pts) (b) proof by contrapositive.

Solution:

(a) Proof by contradiction:
Suppose for sake of contradiction that n pigeons are placed into m pigeonholes, (where $n > m$) with no pigeonhole contains more than one pigeon.
Then each pigeonhole contains at most one pigeon.
Since there are m pigeonholes, there can at most be m pigeons, so the total number of pigeons $n \leq m$, which contradicts the assumption that $n > m$.
Therefore the Pigeonhole Principle is true.

(b) Proof by contrapositive:
The contrapositive of the statement is: For the n pigeons placed into m pigeonholes, if no pigeonhole contains more than one pigeon, then $n \leq m$.
Suppose that $n > m$. Then there are more pigeons than pigeonholes.
Since no pigeonhole contains more than one pigeon, the number of pigeons in every hole is less or equal to 1.
Since there are m holes, the total number of pigeons $n \leq m$.
Then we have proved the contrapositive of the statement. So the Pigeonhole Principle is true.

(15 pts) 6. **Potential method.**

Alice is playing a `FactorFinding` game with herself. The integer factorization is not exciting enough so she decides to consider another number system.

Alice thinks about the "complex integers", i.e. $a + bi$ where a, b are integers and $i^2 = -1$. For example,

- addition: $(1 + 2i) + (3 + 4i) = 4 + 6i$;
- multiplication: $(1 + 2i)(3 + 4i) = 3 - 8 + (6 + 4)i = -5 + 10i$

She then plays the game as follows. Alice starts with an arbitrary "complex integer" $a_1 + b_1 i$ and $k = 1$. Alice tries to find $a_{k+1} + b_{k+1} i$ that divides $a_k + b_k i$ which means there exist two integers c and d such that $(a_{k+1} + b_{k+1} i)(c + di) = a_k + b_k i$. If $|c| + |d| \le 1$, then the game terminates. Otherwise, she increments k and repeats this step.

Note: The initial complex number $a_1 + b_1 i \neq 0$; that is, at least one of a_i and b_i must be non-zero.

Question: Can Alice continue the game forever? If yes, give an example infinite run. If not, prove it by the potential-function method.

Hint: Try the potential function $\Phi(a + bi) = a^2 + b^2$.

Solution: Alice cannot continue the game forever.
Using the potential function $\Phi(a+bi) = a^2+b^2$, we can prove that the game will terminate.

First we claim that: for all complex number $x = a_1 + b_1 i, y = a_2 + b_2 i, z = a_3 + b_3 i$, if $xy = z$, then $|x| \cdot |y| = |z|$.
Proof: $|x| \cdot |y| = \sqrt{a_1^2 + b_1^2} \cdot \sqrt{a_2^2 + b_2^2} = \sqrt{(a_1^2 + b_1^2)(a_2^2 + b_2^2)} = \sqrt{a_1^2 a_2^2 + b_1^2 b_2^2 + a_1^2 b_2^2 + a_2^2 b_1^2}$
$= \sqrt{(a_1 a_2 - b_1 b_2)^2 + (a_1 b_2 + a_2 b_1)^2} = \sqrt{a_3^2 + b_3^2} = |z|$

Then let $a_k + b_k i$ be the complex number at the k-th step, and let $c+di$ be the next possible factor.
Case 1: $|c| + |d| \le 1$. Then the game terminates at $k+1$-th step.
Case 2: $|c| + |d| > 1$. Since c, d are integers, $|c| + |d| \ge 2$, so $|c| \ge 1$ and $|d| \ge 1$.
Then the norm of $c + di = \sqrt{c^2 + d^2} = \sqrt{|c|^2 + |d|^2} \ge \sqrt{2}$, therefore $c^2 + d^2 \ge 2$.
So by our first claim, $\Phi(a_k + b_k i) = (c^2 + d^2) \cdot \Phi(a_{k+1} + b_{k+1} i) \ge 2\Phi(a_{k+1} + b_{k+1} i)$.

Then $\Phi(a_{k+1} + b_{k+1} i) \le \frac{1}{2}\Phi(a_k + b_k i)$ for any step k.
Suppose $\Phi(a_1 + b_1 i) = a$ for some arbitrary positive integer a. Then $\Phi(a_k + b_k i) \le \frac{1}{2^k} a$ for any step k. by the property of integer(real numbers), there must exist some $k_0 \in \mathbb{Z}$ such that $2^{k_0} \le a < 2^{k_0+1}$. So consider $k = k_0 + 1$, then we have $\Phi(a_k + b_k i) \le 1$.
Therefore the game will terminate.

7. **Master theorem.**

Consider the recurrence
$$T(n) = \sqrt{n} \cdot T(\sqrt{n}) + O(n).$$

(5 pts) (a) Explain why the master theorem cannot be applied *directly* to give a closed form for $T(n)$.

Solution: Because the master theorem requires $T(n) = aT(\frac{n}{b}) + O(n^d)$ where a is a constant and the size of subproblems is divided linearly. But here the size of subproblems is divided by $\sqrt{n}$ and the coefficient of $T(\sqrt{n})$ is not a constant.

(5 pts) (b) Define $S(n) = T(n)/n$. Using substitution, write a recurrence for $S(n)$.

Solution: Since $T(n) = \sqrt{n} \cdot T(\sqrt{n}) + O(n)$ and $S(n) = \frac{T(n)}{n} = \frac{\sqrt{n} \cdot T(\sqrt{n}) + O(n)}{n} = \frac{T(\sqrt{n})}{\sqrt{n}} + \frac{O(n)}{n} = S(\sqrt{n}) + O(1)$.
So $S(n) = S(\sqrt{n}) + O(1)$.

(5 pts) (c) Let $n = 2^m$ and define $R(m) = S(2^m) = S(n)$. Using this substitution, write a recurrence for $R(m)$. **Hint:** You may need to use that $\sqrt{n} = \sqrt{2^m} = 2^{m/2}$

Solution: $R(m) = S(2^m) = S(n) = S(\sqrt{n}) + O(1) = S(2^{m/2}) + O(1) = R(m/2) + O(1)$.
Therefore, $R(m) = R(m/2) + O(1)$.

(5 pts) (d) Use the Master Theorem and the above recurrence to get an asymptotic expression for $R(m)$, then use it to get asymptotic expressions for $S(n)$ and finally $T(n)$.

Solution: For $R(n) = R(n/2) + O(1)$, we have $k = 1, b = 2, d = 1$. Since $b^d = 2^1 = 2 > 1$, we have $R(n) = O(n)$.
And then, since $R(m) = S(2^n)$, we have $S(n) = R(\log_2 n) = O(\log n)$ and $T(n) = nS(n) = O(n \log n)$.

(15 pts) 8. **Divide and conquer algorithm.**

The matrix H_t is a $n \times n$ matrix where $n = 2^t$. It is defined recursively, where $H_0 = (1)$, and in general,

$$H_{t+1} = \begin{pmatrix} H_t & H_t \\ H_t & -H_t \end{pmatrix}$$

For example,

$$H_2 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

Given a vector $x \in \mathbb{Z}^n$, there is a trivial algorithm that uses $O(n^2)$ operations to compute the matrix-vector product $H_t x$, by first computing the matrix H_t and then computing its product with x. Describe an algorithm that computes $H_t x$ in $O(n \cdot t) = O(n \log n)$ operations.

Note: Recall how matrix vector multiplication works. If we have a $n \times n$ matrix A and a vector $x \in \mathbb{Z}^n$, their product is calculated by taking the dot product of x with each row in A. (That is, if the row is $[a_1, a_2, ..., a_n]$ and $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$, then the dot product is $a_1x_1 + a_2x_2 + ... + a_nx_n$).

For example, here we have a 2×2 matrix A and a vector $x \in \mathbb{Z}^2$.

$$Ax = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} ax_1 + bx_2 \\ cx_1 + dx_2 \end{bmatrix}$$

Moreover, instead of multiplying each individual row and column, we can also multiply matrices in blocks. For example, if we have a 4×4 matrix A and a vector $x \in \mathbb{Z}^4$, we can break A into 4 smaller matrices, say 2×2 matrices H, I, J, K. We can also break x into 2 vectors $\vec{x}_1$ and $\vec{x}_2$ each of size 2. This would give the following formulation of the product Ax.

$$Ax = \begin{bmatrix} H & I \\ J & K \end{bmatrix} \begin{bmatrix} \vec{x}_1 \\ \vec{x}_2 \end{bmatrix} = \begin{bmatrix} H\vec{x}_1 + I\vec{x}_2 \\ J\vec{x}_1 + K\vec{x}_2 \end{bmatrix}.$$

Solution:

```
Input: int $n$ and vector $\vec{x} \in \mathbb{Z}^n$, where $n = 2^t$ for some $t \in \mathbb{Z}_{\geq 0}$;
Output: vector $\vec{y} \in \mathbb{Z}^n$
function LINEARTRANS($n, \vec{x}$)
    if $n = 1$ then
        return x
    else
        $t = \log_2(n)$
        $\vec{x}_1 = \vec{x}[0 : \frac{n}{2} - 1]$
        $\vec{x}_2 = \vec{x}[\frac{n}{2} : n - 1]$
        $\vec{y}_1 =$ LINEARTRANS($\frac{n}{2}, \vec{x}_1$)
        $\vec{y}_2 =$ LINEARTRANS($\frac{n}{2}, \vec{x}_2$)
        $y[0 : \frac{n}{2} - 1] = \vec{y}_1 + \vec{y}_2$
        $y[\frac{n}{2} : n - 1] = \vec{y}_1 - \vec{y}_2$
        return y
```

This algorithm computes $H_t x$ in $O(n \log n)$ operations. Slicing $\vec{x}$ into $\vec{x}_1, \vec{x}_2$ and concatenating *vecy* by *vecy*$_1$ and $\vec{y}_2$ takes linear time (within $2n$), so the recurrence relation for the algorithm is $T(n) = 2T(n/2) + O(n)$, which is $O(n \log n)$ by the Master Theorem.

(10 EC pts) 9. **Extra credit:** *You are not required to do this question to receive full credit on this assignment.* To receive the bonus points, you must typeset this **entire** assignment in LaTeX, and draw a table with two columns that includes the *name* (e.g., "fraction") and an *example* of each of the following:

- fraction (using `\frac`),

- less than or equal to,
- union of two sets,
- summation using Sigma ($\sum$) notation,
- the set of real numbers ($\mathbb{R}$); write a mathematically correct statement that applies to *all* real numbers $x \in \mathbb{R}$.

Solution:

name	**example**
fraction	$\frac{1}{3} = \frac{2}{6}$
less than or equal to	$2ab \leq a^2 + b^2$
union of two sets	$\{1,2\} \bigcup \{3,4\} = \{1,2,3,4\}$
summation	$\sum_{i=1}^{10} i = 55$
the set of real numbers	$\mathbb{R}$. $\forall a \in \mathbb{R}$, $a \in \mathbb{Q}$ or $a \in \mathbb{R} \backslash \mathbb{Q}$.