Problem Set 1

MIT students: This problem set is due in lecture on *Day 6*.

Reading: Chapters 1–4, excluding §4.4; §28.2; §30.1.

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered by the exercises.

Mark the top of each sheet with your name, the course number, the problem number, your recitation section, the date, and the names of any students with whom you collaborated.

MIT students: Each problem should be done on a separate sheet (or sheets) of three-hole punched paper.

You will often be called upon to "give an algorithm" to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of your essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudocode.
2. At least one worked example or diagram to show more precisely how your algorithm works.
3. A proof (or indication) of the correctness of the algorithm.
4. An analysis of the running time of the algorithm.

Remember, your goal is to communicate. Graders will be instructed to take off points for convoluted and obtuse descriptions.

Exercise 1-1. Do Exercise 2.3-5 on page 37 in CLRS.

Exercise 1-2. Do Exercise 2.3-7 on page 37 in CLRS.

Exercise 1-3. Do Exercise 3.1-1 on page 50 in CLRS.

Exercise 1-4. Do Exercise 4.1-6 on page 67 in CLRS.

Exercise 1-5. Rank the following functions by order of growth; that is, find an arrangement $g_1, g_2, \ldots, g_{30}$ of the functions satisfying $g_1 = \Omega(g_2)$, $g_2 = \Omega(g_3)$, …, $g_{29} = \Omega(g_{30})$. Partition your list into equivalence classes such that $f(n)$ and $g(n)$ are in the same class if and only if $f(n) = \Theta(g(n))$.

$\lg(\lg^* n)$	$(\sqrt{2})^{\lg n}$	n^2	$n!$	e^n	$\lg^*(n^n)$
3^n	n^3	$\lg^2 n$	$\lg(n!)$	$n^{2+\sin n}$	$n^{1/\lg n}$
1	$\lg^*(\lg n)$	$n \cdot 2^n$	$n^{\lg \lg n}$	$\ln n$	$\ln \ln n$
$3^{\lg n}$	$(\lg n)^{\lg n}$	2^n	$n \lg n$	$\sum_{k=1}^{n} \frac{1}{k}$	$\prod_{k=2}^{n} \left(1 - \frac{1}{k}\right)$

Problem 1-1. Asymptotic notation for multivariate functions

The generalization of asymptotic notation from one variable to multiple variables is surprisingly tricky. One proper generalization of O-notation for two variables is the following:

Definition 1

$$O(g(m,n)) = \{f(m,n) : \text{there exist positive constants } m_0,\ n_0, \text{ and } c \text{ such that } 0 \le f(m,n) \le cg(m,n) \text{ for all } m \ge m_0 \text{ or } n \ge n_0\}\ .$$

Consider the following alternative definition:

Definition 2

$$O'(g(m,n)) = \{f(m,n) : \text{there exist positive constants } m_0,\ n_0, \text{ and } c \text{ such that } 0 \le f(m,n) \le cg(m,n) \text{ for all } m \ge m_0 \text{ and } n \ge n_0\}\ .$$

(a) Explain why Definition 2 is a “bogus” definition. That is, what anomalies does the definition of O' permit that are counterintuitive? You may find it helpful to illustrate your answer with a diagram of relevant regions of the $m \times n$ plane.

Remarkably, famous computer scientists have used Definition 2 without being aware of its deficiencies. Nevertheless, their theorems and analyses carry over to Definition 1, because the functions they analyzed satisfy two key properties.

The first property is “monotonicity”:

Definition 3 A two-variable function $f(m,n)$ is ***monotonically increasing*** if

$$f(m,n) \leq f(m+1,n)$$

and

$$f(m,n) \leq f(m,n+1)$$

for all nonnegative m and n.

(b) Explain this definition in plain English.

The second property is more complicated:

Definition 4 A two-variable function $g(m,n)$ is ***multiplicatively separable*** if there exist a constant $L \geq 0$ and two one-variable functions, $a(m)$ and $b(n)$, such that whenever $m \geq 0$, $n \geq 0$, and $g(m,n) \geq L$, we have

$$g(m,n) \leq a(m) \cdot g(m-1,n)$$

and

$$g(m,n) \leq b(n) \cdot g(m,n-1)\ .$$

Intuitively, increasing one argument of a multiplicatively separable function g increases the value of g by at most a multiplicative factor which can be bounded in terms of that argument itself, independent of the other argument.

(c) For each of the following functions $g(m,n)$, argue that g is multiplicatively separable.

i. $g(m,n) = m + n$

ii. $g(m,n) = m^2 n$

iii. $g(m,n) = 2^m + 2^n$

v. $g(m,n) = 2^{m+n}$

vi. $g(m,n) = 2^{2^m+2^n}$

(d) Prove that the following two functions are not multiplicatively separable:

i. $g(m,n) = 2^{m \cdot n}$

ii. $g(m,n) = m^n$

Suppose that f is monotonically increasing and g is multiplicatively separable. Suppose further that $f(m,n) = O'(g(m,n))$, that is, there exist positive constants m_0, n_0, and c such that $0 \leq f(m,n) \leq cg(m,n)$ for all $m \geq m_0$ and $n \geq n_0$.

(e) Prove that there exists a constant $r \geq 0$ such that

$$f(m,n) \leq r \cdot g\left(\max(m,m_0), \max(n,n_0)\right)$$

for all nonnegative m and n.

(f) Prove that there exists a constant $s \geq 0$ such that

$$g\left(\max(m, m_0), \max(n, n_0)\right) \leq s \cdot g(m, n)$$

for all nonnegative m and n.

(g) Conclude that $f(m,n) = O(g(m,n))$.

(h) (*Extra credit.*) Give a proper generalization of Ω to two variables. Justify your definition.

Problem 1-2. Tree Traversal

The following pseudocode is a standard recursive tree-traversal algorithm for counting the number of nodes in a tree R. The initial call is $\text{Count-Nodes}(root[R])$.

```
COUNT-NODES($x$)
1  if $x = \text{NIL}$
2     then return $0$
3     else return $1 + \text{Count-Nodes}(\textit{left}[x])$
                  $+ \text{Count-Nodes}(\textit{right}[x])$
```

Define *size*(x) to be the number of nodes in the subtree rooted at node $x \in R$, and let $T(x)$ denote the worst-case running time of $\text{Count-Nodes}(x)$.

(a) Give a recurrence for $T(x)$ in terms of *left*(x) and *right*(x).

(b) Use the substitution method to prove that $T(x) = O(\textit{size}(x))$.

A common compiler optimization of this code, called ***tail recursion***, is to replace one of the recursive calls with a loop, resulting in the following pseudocode:

```
COUNT-NODES-TAIL($x$)
1  $s \leftarrow 0$
2  while $x \neq \text{NIL}$
3        do $s \leftarrow s + 1 + \text{Count-Nodes-Tail}(\textit{left}[x])$
4           $x \leftarrow \textit{right}[x]$
5  return $s$
```

Let *right*$^i[x]$ denote the ith right descendant of x, that is,

$$\textit{right}^{\,i}[x] = \begin{cases} x & \text{if } i = 0 \,, \\ \textit{right}[\textit{right}^{\,i-1}[x]] & \text{if } i > 0 \,. \end{cases}$$

Consider the loop invariant

$$s = k + \sum_{i=0}^{k-1} \text{Count-Nodes-Tail}(\textit{left}[\textit{right}^{\,i}[x]]), \tag{1}$$

where $k \geq 0$ is the number of times the **while** loop (lines 2–4) in COUNT-NODES-TAIL has been executed.

(c) Prove that if Equation (1) holds for k, then it holds for $k+1$.

Let $K(x)$ be the smallest positive integer for which $\textit{right}^{K(x)}[x] = \text{NIL}$.

(d) Prove that COUNT-NODES-TAIL returns

$$K(x) + \sum_{i=0}^{K(x)-1} \text{COUNT-NODES-TAIL}(\textit{left}[\textit{right}^{\,i}[x]]) \ .$$

(e) Prove by induction that COUNT-NODES-TAIL(x) correctly computes $\textit{size}(x)$.

Problem 1-3. Polynomial multiplication

If we have two linear polynomials $ax+b$ and $cx+d$, we can multiply them using the four coefficient multiplications

$$\begin{aligned} m_1 &= a \cdot c\ , \\ m_2 &= a \cdot d\ , \\ m_3 &= b \cdot c\ , \\ m_4 &= b \cdot d \end{aligned}$$

to form the polynomial

$$m_1x^2 + (m_2 + m_3)x + m_4\ .$$

(a) Give a divide-and-conquer algorithm for multiplying two polynomials of degree-bound n based on this formula.

(b) Give and solve a recurrence for the worst-case running time of your algorithm.

(c) Show how to multiply two linear polynomials $ax + b$ and $cx + d$ using only *three* coefficient multiplications.

(d) Give a divide-and-conquer algorithm for multiplying two polynomials of degree-bound n based on your formula from part (c).

(e) Give and solve a recurrence for the worst-case running time of your algorithm.