Tim Cheeseman
CS 260-003
Summer 2014
Assignment 2 Sample Solution

1.13)

a) $17 \le c \cdot 1$ with $n_0 = 0$, $c = 17$

b) $\frac{n(n-1)}{2} \le c \cdot n^2$ with $n_0 = 0$, $c = 1$

c) $10 \cdot n^2 \le c \cdot n^3$ with $n_0 = 10$, $c = 1$
$n^3 \le c \cdot n^3$ with $n_0 = 0$, $c = 1$

d) $\sum_{i=1}^{n} i^k \le \sum_{i=1}^{n} n^k = n^{k+1}$
$\therefore \sum_{i=1}^{n} i^k \le c \cdot n^{k+1}$ with $n_0 = 1$, $c = 1$
$\therefore \sum_{i=1}^{n} i^k \in O(n^{k+1})$

$\sum_{i=1}^{n} i^k \ge c \cdot \sum_{i=1}^{n} n^k = c \cdot n^{k+1}$ with $n_0 = 1$, $c = \frac{1}{k+1}$ ($\sum_{i=1}^{n} i^k \ge \int_{0}^{n} x^k dx = \frac{n^{k+1}}{k+1}$)
$\therefore \sum_{i=1}^{n} i^k \in \Omega(n^{k+1})$

e)

$p(n) = c_0 + c_1 \cdot n + c_2 \cdot n^2 + \ldots + c_k \cdot n^k$
$\le c_0 \cdot n^k + c_1 \cdot n^k + c_2 \cdot n^k + \ldots + c_k \cdot n^k$
$\le c \cdot k \cdot n^k$ with $c = \sum_{i=1}^{k} c_k$
$\le c \cdot n^k =$ with $n_0 = 1$, $c = k \cdot \sum_{i=1}^{k} c_k$
$\therefore p(n) \in O(n^k)$

$p(n) = c_0 + c_1 \cdot n + c_2 \cdot n^2 + \ldots + c_k \cdot n^k \ge c \cdot n^k$ with $n_0 = 1$, $c = c_k$
$\therefore p(n) \in \Omega(n^k)$

1.16)

- Lowest
- (h) $(1/3)^n$ (asymptotically approaches 0)
- (j) 17
- (d) log(log(n))
- (c) log(n)
- (e) $\log^2(n)$
- (b) $\sqrt{n}$
- (f) n/log(n)
- (g) $\sqrt{n} \cdot \log^2(n)$
- (a) n
- (i) $(3/2)^n$
- Highest

2.9)

In the case of an array implementation of a list, calling DELETE(p, L) moves whatever is in position p+1 into p. We then immediately call p := NEXT(p, L) which brings us to position p+1, and if what is now in position p is equal to x, we will have skipped a match and it will not be removed from the list.

In the case of a pointer implementation of a list (linked list), calling DELETE(p, L) deletes the cell after the cell pointed to by p. We then immediately call p := NEXT(p, L) which results in p pointing to the cell that was after the one just deleted, which means we have skipped over that cell. If its data is equal to x, we will have skipped a match and it will not be removed from the list.

In both cases, the fix is to put the call to p := NEXT(p, L) in an else block so that it executes only when an element is *not* deleted, as deleting an element essentially advances the position already.

2.11)

Outermost Loop	Executed for p = 1, …, n	n iterations Add 1 if including predicate checks

Middle Loop	Executed for:
	q = 1, …, n (n iterations)
	q = 2, …, n (n - 1 iterations)
	…
	q = n (1 iteration)
		$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ iterations
		Add n if including predicate checks

Innermost Loop	Executed for:
	r = 1 (1 iteration)
	r = 1, 2 (2 iterations)
	…
	r = 1, …, n (n iterations)
	This entire list is repeated for each of the n iterations of the outer loop.
		$n \sum_{i=1}^{n} i = \frac{n^2(n+1)}{2}$ iterations
		Add n if including predicate checks

FIRST	Called once at beginning = 1 Called once for every iteration of middle loop = $\frac{n(n+1)}{2}$ **Total** = $\frac{n(n+1)}{2}$ +1
NEXT	Called once per iteration of the middle loop = $\frac{n(n+1)}{2}$ Called once per iteration of the innermost loop = $\frac{n^2(n+1)}{2}$ **Total** = $\frac{n^3+2n^2+n}{2}$
END	Called once per check of the outer loop = n + 1 Called once per check of the middle loop = $\frac{n(n+1)}{2}$ + n **Total** = $\frac{n(n+1)}{2}$ + 2·n + 1 = $\frac{n^2 +5n+2}{2}$

Implementation)

list_concat(A, B):

Assume n = len(A), m = len(B)

T(n, m) = n + 1 = O(n)

list_concat_copy(A, B):

Assume n = len(A), m = len(B)

T(n, m) = n + m = O(n + m)

When n = m, it's O(2⋅n) = O(n)

Implementation	**Pros**	**Cons**
C = list_concat(A, B)	● Faster than list_concat_copy ● Uses less memory	● Risky; changes to B could leak cells or affect C ● list_concat(A, A) creates cycle in list, though we could guard against that
C = list_concat_copy(A, B)	● Safer; changes to A and B have no effect on C	● Slower than list_concat ● Uses more memory