Solutions to Quiz/Homework Assignment #1
CSE326

January 17, 2000

1. $\sum_{i=0}^{\infty} x^i = 1/(1-x)$ is true if and only if:

   (a) $-1 < x < 1$
   (b) $x \neq 1$
   (c) $0 < x < 1$
   (d) $x > 1$
   (e) $x = 1/2$

   The correct answer is (a). I will first show that $-1 < x < 1 \Rightarrow \sum_{i=0}^{\infty} x^i = 1/(1-x)$ (the *if* direction), and then I will show that these are the only xs that work (the *only if* direction). The book gives a reasonable demonstration that the formula holds for $0 < x < 1$, so I will take that as a given. When $x = 0$ we have:

$$\sum_{i=0}^{\infty} 0^i = 0^0 + 0^1 + 0^2 + 0^3 \cdots$$

   By definition, we know that $0a = 0$ for any number a, so all terms but the first are zero. The first term is just the empty product (which can also be written $\prod_{\emptyset}$)—a product of zero numbers—which is defined to be the multiplicative identity, 1 (similarly the empty sum, $\sum_{\emptyset}$, is defined to be the additive identity, 0). Thus,

$$\sum_{i=0}^{\infty} 0^i = 1 = \frac{1}{1-0}$$

   Now spose $-1 < x < 0$. Then

$$\sum_{i=0}^{\infty} x^i = \sum_{i=0}^{\infty} (x^{2i} + x^{2i+1}) = \sum_{i=0}^{\infty} (1+x)x^{2i} = (1+x)\sum_{i=0}^{\infty} x^{2i} = \frac{1+x}{1-x^2} = \frac{1}{1-x}$$

   It remains to show that the formula is false for values of x not in the interval $(-1, 1)$. Spose $x > 1$. Then all terms in the series are positive and non-decreasing, thus the series cannot converge to the finite (and negative) value $1/(1-x)$. If $x < -1$ then the series can be rewritten as $(1+x)\sum_{i=0}^{\infty} x^{2i}$. Each term is positive and non-decreasing, and the quantity $(1+x)$ is finite and non-zero, so again the series cannot converge to the finite value $1/(1-x)$. Now we have two special cases: $x = \pm 1$. If $x = 1$ then clearly the series diverges to ∞. If we defined $1/0 = \infty$, then the formula would be true in this case, but we don't. As Steve said in class, ∞ and "undefined" are not the same. If $x = -1$ then the partial sums of the series alternate between 1 and 0. So the *average* value is $1/2 = 1/(1-(-1))$, but the series doesn't converge so the formula doesn't hold.

2. *Note: this question was removed from the quiz and homework.*

   Which of the following formulas has the **smallest** value?

   (a) $(\sum_{i=1}^{5} i)(\sum_{i=1}^{5} i)$
   (b) $\sum_{i=0}^{100}(-5)^i$
   (c) $\sum_{n=0}^{5}(\sum_{i=0}^{n} 2^i)$
   (d) $\sum_{i=0}^{10} 2^3$

   Just FYI:

   (a) Using the rule that $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$, this is:
   $\frac{5(5+1)}{2}^2 = 15^2 = 225$
   (b) Let's just get a feel for this one. -5^{100} is the largest term in this series. It's equal to 5^{100} since $(-1)^n = 1$ for even n. That's a huge number. It gets a chunk of $-(5^{99})$ taken out of it, but that's still $4 * 5^{99}$, and the rest of the series just adds more. So this is very big.
   (c) Using the rule that $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$, this becomes
   $\sum_{n=0}^{5}(2^{n+1} - 1) = 2\sum_{n=0}^{5} 2^n + \sum_{n=0}^{5} 1$
   Using the same rule again, we get:
   $2(2^{5+1}) + 5 = 128 + 5 = 133$
   (d) $\sum_{i=0}^{10} 2^3 = 2^3 \sum_{i=0}^{10} 1 = 2^3 * 11 = 88$

   So, letter (e) is the smallest.

3. Which of the following formulas has the **greatest** value?

   (a) $\log_{10} 1024$
   (b) $(\log_{13} 343)/(\log_{13} 7)$
   (c) $\log_2(1/128)$
   (d) $\log_8 256$
   (e) $(\log_a a^2) + (2\log_a \sqrt{a})$

   (a) Logarithms are monotonically increasing, hence $\log_{10} 1024 > \log_{10} 1000 = 3$.
   (b) Using the rule $\log_A B = \frac{\log_C B}{\log_C A}$, we get:
   $\frac{\log_{13} 343}{\log_{13} 7} = \log_7 343 = \log_7 7^3 = 3$.
   (c) $\log_2(1/128) = \log_2 2^{-7} = -7$.
   (d) As for (b), $\log_8 256 = \frac{\log_2 256}{\log_2 8} = \frac{\log_2 2^8}{\log_2 2^3} = 8/3$.
   (e) $(\log_a a^2) + (2\log_a \sqrt{a}) = 2 + 2(1/2) = 3$.

   So, letter (a) is the greatest.

4. Which of the following is not equal to $a^3 + 4ab^2$?

   (a1) $a(a^2 + \frac{6ba}{3ab^0}^2)$
   (a2) $a(a^2 + \frac{6ba}{3(ab)^0}^2)$
   (b) $\frac{a^4b}{ab} + a(2b)^2$

(c) $((2^{1+a+b})(2^{1+a-b})a^2b^2 + a^4)/(a4^a)$

(d) $4\frac{((2ab)^2+a^4)}{4a}$

(e) none of these

Note: I am including two versions of (a) in the solutions to reflect the original (a1) and updated (a2) versions of the homework. As written on the quiz, (a) was *not* a correct answer. The five people who answered (a) should pay special attention to this solution because, due to an error on our part, they were not required to redo this problem on their homework.

(a1) $a(a^2 + \frac{6ba}{3ab^0}^2) = a(a^2 + \frac{6ba}{3a}^2) = a(a^2 + (2b)^2) = a(a^2 + 4b^2) = a^3 + 4ab^2$

(a2) $a(a^2 + \frac{6ba}{3(ab)^0}^2) = a(a^2 + \frac{6ba}{3}^2) = a(a^2 + (2ab)^2) = a(a^2 + 4a^2b^2) = a^3 + 4a^3b^2$

(b) $\frac{a^4b}{ab} + a(2b)^2 = a^3 + a(4b^2) = a^3 + 4ab^2$

(c) I actually intended this to be written $((2^{1+a+b})(2^{1+a-b})(a^2b^2 + \frac{1}{4}a^4))/(a4^a)$, in which case the answer would have been (e), but instead (c) evaluates to:

$$\frac{(2^{1+a+b})(2^{1+a-b})a^2b^2 + a^4}{a4^a} = \frac{2^{2(a+1)}a^2b^2 + a^4}{a4^a} = \frac{4^{a+1}a^2b^2 + a^4}{a4^a} = \frac{(a4^a)(4ab^2) + a^4}{a4^a} = 4ab^2 + \frac{a^3}{4^a}$$

(d) $4\frac{((2ab)^2+a^4)}{4a} = \frac{(4a^2b^2+a^4)}{a} = \frac{a(4ab^2+a^3)}{a} = a^3 + 4ab^2$

Choices (a1), (b) and (d) evaluate to $a^3 + 4ab^2$ while (a2) and (c) do not. Hence (a2) and (c) are the two correct answers.

5. *Note, this question has two correct answers. If you got either right on the quiz, you were correct. Otherwise, find and justify both for the homework.*

   Which of the following does **not** equal 1?

   (a) 5 mod 2

   (b) (98179387171 − 19873956961) mod 10

   (c) $(4879165163 * 91991931877)$ mod 10

   (d) $(1987917135 + 19871938574)$ mod 2

   (e) (1 mod 1999) + 1

   Note: The expressions ought to have been parenthesized for clarity, as shown above. The proper way to think about "mod" (from a theory standpoint) is as a mode or environment, rather than as a function. Thus in parts (a) and (d) we are dealing with the ring of integers modulo 2 (which just has two elements), in parts (b) and (c) we are dealing with the ring of integers modulo 10 (10 elements), and in part (e) we are dealing with the ring of integers modulo 1999 (you guessed it—1999 elements). In the solutions below I will use $\equiv_M$ to denote equivalence modulo M.

   (a) $5 = 2 \cdot 2 + 1$, hence $5 \equiv_2 1$

   (b) $98179387171 \equiv_{10} 1$, and $19873956961 \equiv_{10} 1$, so $(98179387171 - 19873956961) \equiv_{10} (1 - 1) \equiv_{10} 0$

   (c) $4879165163 \equiv_{10} 3$, and $91991931977 \equiv_{10} 7$, so $(4879165163 * 91991931877) \equiv_{10} (3 * 7) \equiv_{10} 21 \equiv_{10} 1$

   (d) $1987917135 \equiv_2 1$, and $19871938574 \equiv_2 0$, so $(1987917135 + 19871938574) \equiv_2 (1+0) \equiv_2 1$

(e) $(1 + 1) \equiv_{1999} 2$

Thus (b) and (e) are the two choices which are not equivalent to 1.

6. What is the best way to implement a queue?

   (a) A linked list, because the memory usage is proportional to the number of elements in the queue.
   (b) A circular array, because no memory is wasted with next (or previous) pointers.
   (c) A linked list, because it doesn't limit how many elements you can have in the queue.
   (d) A circular array; it's faster because you don't have to deference next or previous pointers.
   (e) It depends.

   I don't think this one really needs much of a solution. The answer, of course, is (e). Any one of the other choices may be a perfectly valid reason to choose one implementation over the other *depending* on the situation. As an example, if you know ahead of time how many elements will be in your queue and you are concerned about speed, then it's probably best to use a circular array. If, however, you don't know ahead of time how many elements you will have, or if your memory is too fragmented to hold a large array, then it's probably best to use a linked list.

7. *Note, this question has* no *correct answer on the quiz. You should find and justify the correct answer for homework.*

   The following pseudocode uses the functions C, F, and G.

```
if (C()) {
  for i = 1 to n
    for j = 1 to n
      F()
} else {
  F()
  G()
}
```

   In terms of the runtime of C, F, and G ($T(C)$, $T(F)$, and $T(G)$, respectively) the asymptotic bound on the runtime of this code must be equivalent to which of the following?

   There are a few reasonable answers to this. I will use a conservative upper bound based on the sum of the time to evaluate the conditional, the time to evaluate one branch, and the time to evaluate the other branch. The actual running code *cannot* do more work than this, so it is a valid upper bound.

   I'll set up a pair of sums to find out how long the loop takes in the first branch, and add the times for F and G (since they occur in sequence) for the second branch.

   Time for conditional $= T(C)$

   Time for first branch $= \sum_{i=1}^{n} \sum_{j=1}^{n} T(F) = n^2 T(F)$

   Time for second branch $= T(F) + T(G)$

   All told then, this is $O(T(C) + T(F) + T(G) + n^2 T(F))$, but $T(F)$ is $o(n^2 T(F))$, so we can drop $T(F)$ (but may not drop $n^2 T(F)$). The final time then is $O(T(C) + T(G) + n^2 T(F))$.

   Note that any of C, F, and G may read input or access global variables; so, their runtime is *not* necessarily constant.

8. Given that $T_1(N) \in \theta(f(N))$ and $T_2(N) \in \theta(f(N))$ which of the following is **not** true?

   (a) $T_1(N) + T_2(N) \in O(f(N))$
   (b) $T_1(N) - T_2(N) \in o(f(N))$
   (c) $T_1(N)/T_2(N) \in O(1)$
   (d) $T_1(N) \in O(T_2(N))$
   (e) $T_1(N) \in \Omega(T_2(N))$

   $T(n) \in \theta(f(n))$ if and only if $T(n) \in O(f(n))$ and $T(n) \in \Omega(f(n))$ based on the definition of θ. So, $T_1(N) \in O(f(N))$ and $T_1(N) \in \Omega(f(N))$ and $T_2(N) \in O(f(N))$ and $T_2(N) \in \Omega(f(N))$. There must be some constant c_1 such that $T_1(N) \leq c_1 f(N)$ for sufficiently large N and another (not necessarily equal) constant c_2 such that $T_2(N) \leq c_2 f(N)$ for sufficiently large N. (And the same holds with constants d_1 and d_2 for $\geq$.) Now, on to the options:

   (a) We can just add c_1 and c_2 and prove that the new function must be $\leq (c_1 + c_2)f(N)$ for sufficiently large N.
   (b) This is false, and I'll prove it by counterexample. Let $f(N) = n^2$, let $T_1(N) = 5n^2$, and let $T_2(N) = 2n^2$. Clearly, these satisfy the requirements of the problem. However, $T_1(N) - T_2(N) = 3n^2 \in \theta(n^2)$ and by definition this is *not* $\in o(n^2)$.
   Notice, however, that if the two functions are equal (or even just have the same coefficient for their high order term) their difference *is* $\in o(f(N))$.
   (c) The largest this fraction can get for sufficiently large N is $\frac{c_1 f(N)}{d_2 f(N)} = c_1/d_2$ (upper bound of numerator over lower bound of denominator). This is a constant, so the fraction is bounded above by constant multiples of 1. Therefore, by definition, it is $\in O(1)$.
   (d) For sufficiently large N:
   $T_2(N) \geq d_2 f(N)$ (since $T_2(N) \in \Omega(f(N))$). So, $\frac{c_1}{d_2}T_2(N) \geq c_1 f(N)$. But, $c_1 f(N) \geq T_1(N)$ (since $T_1(N) \in O(f(N))$). So, $\frac{c_1}{d_2}T_2(N) \geq T_1(N)$, and, by definition, $T_1(N) \in O(T_2(N))$.
   (e) By analogous reasoning to (d), this is true.

9. Which of the choices below is a correct upper bound on the running time of this code?

```
sum = 0;
for (i = 0; i < n; i++)
  for (j = 0; j < i; j++)
    sum++;
```

   (a) $O(1)$
   (b) $\omega(n^2)$
   (c) $o(n^2)$
   (d) $O(n^2 \log n)$
   (e) none of these

   The "sum++" line takes constant time, and the loops set up the following formula:

$$
\begin{aligned}
\sum_{i=0}^{n}\sum_{j=0}^{i}\theta(1) &= \theta(1)\sum_{i=0}^{n}\sum_{j=0}^{i}1 \\
&= \theta(1)\sum_{i=0}^{n}(i+1) \\
&= \theta(1)(\sum_{i=0}^{n}i+\sum_{i=0}^{n}1) \\
&= \theta(1)(\frac{n(n+1)}{2}+n+1) \\
&= \theta(1)n^2/2+3\theta(1)n/2+\theta(1)
\end{aligned}
$$

Dropping lower order terms and coefficients, this is $\theta(n^2)$. Now, on to the answers:

(a) $O(1)$ is an upper bound, but $\theta(n^2) \not\subset O(1)$ (it's faster than n^2 on the race sheet).

(b) $\omega(n^2)$ is not an upper bound.

(c) $o(n^2)$ *is* an upper bound, but, by definition, a function is only $o(f(n))$ if it is *not* $\theta(f(n))$. So this is wrong.

(d) $O(n^2 \log n)$ is an upper bound which is larger than n^2. But, that's OK. It just means this is a loose upper bound.

10. Under which conditions would you want to store a reference to each row and column header in each element of a cross list?

    (a) Time is cheap, memory is precious.

    (b) Memory is precious, but the actual information stored at each element is small.

    (c) Time is precious, memory is cheap, but the actual information stored at each element is large.

    (d) none of the above

    Pointers to the column and row headers save time traversing through the row and column lists to find those headers, but they take extra space in each node. However, if the data is large to start with, the pointers add very little extra memory requirement to each node. Therefore, an excellent time to use pointers to the headers is when time is precious (because they speed up the cross list), memory is cheap (because they require storage), but the actual information stored at each element is large (because then the memory hit — which is cheap anyway — is also relatively small).

11. What is a tight upper bound on the running time of this code?

```
for (i = 1; i * i <= n; i++)
  if (n % i = 0)
    cout << i << ' ';
cout << endl;
```

    (a) $O(\sqrt{n})$

    (b) $O(1)$

(c) $\Omega(n)$
(d) $O(n^2)$
(e) none of the above

The loop repeats as long as $i * i \leq n$ or equivalently $i^2 \leq n$ or $i \leq \sqrt{n}$. The conditional inside each loop takes constant time whether it executes or not. So, the code takes:
$\sum_{i=1}^{\sqrt{n}} \theta(1) \in \theta(\sqrt{n})$
So, only $O(\sqrt{n})$ is a tight upper bound on the code.

12. Which choice below is a tight asymptotic bound for a function with the following runtime characteristics?

$T(n) = T(n-1) + n$

$T(1) = 1$

(a) $\Omega(n)$
(b) $\theta(n)$
(c) $O(n^3)$
(d) $\theta(n^2)$
(e) none of the above

Note: A number of people answered that since we add n things, and each of them is about n, $T(n) = n \cdot n \in \Theta(n^2)$. This is handwaving, and in the future you will need to be a bit more careful to get full marks on a question. Actually (as we'll show), the sum is $1 + 2 + 3 + \ldots + n$, and it's not obvious that you can think of this as $n \cdot n$, particular for a lower bound.
Note 2: A question like this is asking for a proof, so arguing by eliminating the choices that the questions gives is not really sufficient. Also, many people who did this had very vague justifications (e.g. it's "clearly" much greater than $\Theta(n)$).
Here's a solution.

$$\begin{aligned} T(n) &= T(n-1) + n \\ &= n + T(n-2) + (n-1) \qquad \text{by recursive substitution} \\ &= n + (n-1) + T(n-3) + n - 2 \end{aligned}$$

So, after k recursive substitutions of the formula, we get

$$T(n) = \left(\sum_{i=n+1-k}^{n} i \right) + T(n-k)$$

And after $n-1$ recursive substitutions (keeping in mind that $T(1) = 1$),

$$T(n) = \left(\sum_{i=1}^{n} i \right)$$

Now, $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ (there's no need to justify this formula, since we just learned it as a fact in class).

Thus, we have

$$
\begin{aligned}
T(n) &= \frac{n(n+1)}{2} \\
&= \frac{1}{2}n^2 + \frac{1}{2}n \\
&\in \Theta(n^2)
\end{aligned}
$$

So, (d) is the correct answer. You could get away with a bit less of the english text, and introducing k is not necessary as long as you demonstrate the pattern before going to the $\sum$ formula.

13. Given the following templated class:

```
template <class T>
class Templ
public:
  T min (T x, T y) {
    if (x < y) return x;
    else return y;
  }
};
```

What can we say about class T?

(a) It has a less than test function, 'operator $<$'.

(b) It has an equality testing function, 'operator $==$'.

(c) It has a less than test function, 'less'.

(d) All of the above.

(e) None of the above.

(a) is correct. If we're comparing X with Y using operator ¡, then the class T must have that operator defined (technically, as one student pointed out, it could have operator ¡ defined as a friend function.).

We never used == or called a 'less' method, so we can't say that it necessarily has those functions defined.

14. What is the flaw in the following inductive proof?

Theorem All dogs are the same color.

Proof Proof by induction. Formally, we wish to prove that, for any set of n dogs, all dogs are the same color.

Base case: $n = 1$ Clearly, if we have a set of only one dog, the dog is the same color as itself.

Induction step We assume that the property is true for some n, and show that it is true for $n+1$. We select two sets of n dogs A and B such that all of the $n+1$ dogs occurs in at least one of the sets A or B, And there is one dog that appears in both sets (call this dog "Rover"). Since A and B are of size n, we know from the Induction Hypothesis that the dogs in each set are the same color. Moreover, since Rover is in both sets, both sets must be the same color. So, the property holds for $n+1$ dogs. QED.

(a) The base case does not cover sets with no dogs ($n = 0$), so we can't conclude that all dogs are the same color for any number of dogs.

(b) We did not mention what happens for $n > 1$, so we have not proven the property for any number of dogs.

(c) The given Induction Step does not work when $n = 1$, so the property does not hold for larger n.

(d) None of the above.

A number of people selected choice (b). The proof doesn't need to explicitly *mention* what happens for $n > 1$, because the induction step, in combination with the base case, takes care of that. So, implicitly, this proof *did* mention $n > 1$. It just had flawed reasoning.

The flaw is that the induction step does not work when $n = 1$. Consider what would happen with the set with $n+1 = 2$ dogs when $n = 1$. According to the induction step, there is a dog, Rover, who is in both sets. With $n = 1$, the induction step uses sets of 1 dog each, for sets A and B.

If Rover is in both sets, then there's only one dog total (same dog in both sets). If the two sets have different dogs, then Rover doesn't have a place.

So, it's choice (c).

15. In C, strings are implemented as arrays. They could be implemented using linked lists, specifically a linked list of characters. In the string "Data Structures are supercool", the head of the list would be the letter 'D', the next node would be 'a', the next node would be 't', etc.

    We could use a singly linked list or a doubly linked list. For which of the following operations, would you want to use a doubly-linked list (assuming the program performed this operation frequently)? Assume that only big-O time is significant.

    (a) Find character (e.g., does some string contain the letter 'A'?)

    (b) Count length of string (e.g., given "Data", the length is 4)

    (c) Reverse string (e.g., given "Data", output "ataD")

    (d) Find previous (i.e., given a pointer to a character, find the previous character)

    (e) Concatenate strings (e.g., given "Data" and "Structures", output "DataStructures")

    note: To get full credit on an answer like this in the future, you'll need to show why the other choices aren't correct. That is, you have to demonstrate that you understand not only why Find previous is difficult for singly linked lists, but also why all of the other choices are just as easy.

    - Find character. Clearly, we can just follow next pointers through the list, looking for our character. So both singly and doubly linked lists are $\Theta(n)$ worst case for this.
    - Count length. Like find character, we can follow next pointers, counting the number of characters in the string. So both singly and doubly linked lists are $\Theta(n)$ for this.

- Reverse string. A straightforward way to implement this is to push everything onto a stack, and then pop it off. (This does mean that you'll need to use a doubly-linked list or an array in order to implement the stack, but dynamically creating a doubly-linked list still preserves the linear time bound).

  Alternately, you can go through the list reversing next pointers, as the following (C++ style) pseudocode shows. Note that you could copy to a new singly linked list beforehand in linear time, if you were concerned about clobbering the input.

  ```
  p=listHead
  If p==NULL
    Return
  End If
  p2=p->nextPtr
  p->nextPtr=NULL // p will become the tail of the reversed list
  While p2!=NULL
    p3=p2->nextPtr
    p2->nextPtr=p
    p=p2
    p2=p3
  End While
  listHead=p
  ```

  So, it's $\Theta(n)$ time for doubly and singly linked lists.

- Find previous. Correct answer. For a doubly-linked list, you can follow the prev pointer in $\Theta(1)$. However, for a singly linked list, you have no choice but to walk the whole list, in $\Theta(n)$ worst case (and probably average case).

- Concatenate strings. The prev pointers of the doubly-linked list don't help. You create a new list, then walk the first input list using its next pointers, copying its nodes to the end of the new list (you'll need the tail pointer of the new list for this, but only temporarily). Then you copy the nodes from the second input list to the end of the new list. This is $\Theta(n)$ for either kind of linked list.