Prerequisite Review Problems

These selected questions should help you review the material covered by this course's prerequisites. Note that they do not represent the entirety of the prerequisite material.

1. (CSC 202) Give a recursive function that finds the index of a value in a Python list, or determines that such an index does not exist. For example, given:

   `[2, -1, 9, 8, 5, -3, 0, 8]`

   ...and the value `5`, your function should return `4`, whereas given the value `6`, it should return `None`.

```
def index(lst, value, idx = 0):
    if idx == len(lst):
        return None
    elif lst[idx] == value:
        return idx
    else:
        return index(lst, value, idx + 1)
```

2. (CSC 202) Consider the following functions:

```
def f(n):
    i = 1
    while i < n:
        i = i * 2

    return i
```

```
def g(n):
    for j in range(n):
        for k in range(n):
            f(n)

    return 0
```

   What are the Big-O estimates of these functions' running times?

   The loop in `f` runs as many times as it is possible to multiply 1 by 2 until reaching n, which is approximately $\log_2 n$. The nested loops in `g` run n^2 times, calling `f` on each iteration. Thus, `f` has complexity $O(\log n)$ and `g` has complexity $O(n^2 \log n)$

3. (CSC 202) Give the Big-O estimate of each of the following operations' running time.

   (a) Adding to the middle of an array list

   Adding to the middle of an array list requires shifting all elements after that point, and thus has complexity $O(n)$.

   (b) Removing from the beginning of a linked list

   Removing from the beginning of a linked list requires only accessing the head node, and thus has complexity $O(1)$.

   (c) Pushing to an array stack

   Pushing to an array stack is adding to the end of an array list, and thus has amortized complexity $O(1)$, assuming that capacity increases geometrically.

   (d) Rehashing every key-value pair in a hash table

   Rehashing is simply re-inserting every key-value pair, and thus has average-case complexity $O(n)$. There is theoretically a chance (though very small, given a decent hash function) that every key collides under the new capacity, which would then produce the worst-case complexity $O(n^2)$.

4. (CSC 202) Consider the following list:

0	1	2	3	4	5	6	7
`19`	`0`	`76`	`-1`	`12`	`30`	`5`	`-8`

Suppose that this list has been implemented using arrays, such that it supports fast random access.

(a) Given the above unsorted list, show the steps of an insertion sort to sort it in ascending order.

1. Start at index 1*:* $[19, 0, 76, -1, 12, 30, 5, -8]$

2. Insert 0 *before* 19*:* $[0, 19, 76, -1, 12, 30, 5, -8]$

3. Insert 76 *after* 19*:* $[0, 19, 76, -1, 12, 30, 5, -8]$

4. Insert -1 *before* 0*:* $[-1, 0, 19, 76, 12, 30, 5, -8]$

5. Insert 12 *between* $0, 19$*:* $[-1, 0, 12, 19, 76, 30, 5, -8]$

6. Insert 30 *between* $19, 76$*:* $[-1, 0, 12, 19, 30, 76, 5, -8]$

7. Insert 5 *between* $0, 12$*:* $[-1, 0, 5, 12, 19, 30, 76, -8]$

8. Insert -8 *before* -1*:* $[-8, -1, 0, 5, 12, 19, 30, 76]$

(b) Given the resulting sorted list, show the steps of a binary search to find the index of `0` within it.

1. Set the midpoint to 3*.*

2. Since $5 > 0$*, set high to* 2*.*

3. Set the midpoint to 1*.*

4. Since $-1 < 0$*, set low to* 2*.*

5. Set the midpoint to 2*.*

6. Since $0 = 0$*, return* 2*.*

5. (CSC 202) Consider the following integers, in no particular order:

$$8,\ 13,\ -6,\ 10,\ 211,\ 91,\ 12,\ \text{and}\ 5$$

(a) Give an order in which to insert these integers into a binary search tree, resulting in the most balanced tree possible.

In general, the median key should to be inserted first, so that each of its subtrees will contain roughly half of the remaining keys. One possibility is $12,\ 5,\ 91,\ -6,\ 8,\ 13,\ 211,\ 10$*. Note that it is not possible to construct a perfect binary tree, since the number of keys is not of the form* $2^n - 1$*.*

(b) Give an order in which to insert these integers into a binary search tree, resulting in the least balanced tree possible.

In general, the minimum key should to be inserted first, so that one of its subtrees will be empty. One possibility is $-6,\ 5,\ 8,\ 10,\ 12,\ 13,\ 91,\ 211$*. Note that this could also be done in descending order with the maximum key, provided the insertion order is consistent.*

6. (CSC 248) Consider the following statement:

   Let n be an integer. If $3n+2$ is even, then n is even.

   Recall that any proof by contraposition can also be done by contradiction.

   (a) Using a proof by contraposition, prove the above statement.

   Proof:

   - *Let n be an integer and suppose, for contraposition, that n is odd. By definition, $n = 2k+1$, where k is an integer.*
   - *Then $3n+2 = 6k+3+2 = 2(3k+2)+1$, which is odd. Thus, if n is odd, then $3n+2$ is odd.*
   - *Therefore, by contraposition, if $3n+2$ is even, then n is even.* $\square$

   (b) Using a proof by contradiction, prove the above statement.

   Proof:

   - *Let n be an integer and suppose, for contradiction, that $3n+2$ is even and n is odd. By definition, $n = 2k+1$, where k is an integer.*
   - *Then $3n+2 = 6k+3+2 = 2(3k+2)+1$, which is odd. This contradicts the assumption that $3n+2$ was even.*
   - *Therefore, by contradiction, if $3n+2$ is even, then n is even.* $\square$

7. (CSC 248) Consider the following statement:

   Let x and y be integers. If x and y have opposite parity, then $5x+5y$ is odd.

   Recall that the phrase "without loss of generality" asserts that all other cases are trivially similar.

   (a) Without using the notion of "without loss of generality", prove the above statement.

   Proof:

   - *Let x and y be integers of opposite parity. Two possibilities exist:*
     - *If x is even and y is odd, by definition, $x = 2k$ and $y = 2l+1$, where k and l are integers. Then $5x+5y = 10k+10l+5 = 2(5k+5l+2)+1$, which is odd.*
     - *If x is odd and y is even, by definition, $x = 2k+1$ and $y = 2l$, where k and l are integers. Then $5x+5y = 10k+5+10l = 2(5k+2+5l)+1$, which is odd.*
   - *In all possible cases, $5x+5y$ is odd.*
   - *Therefore, if x and y are integers of opposite parity, then $5x+5y$ is an odd integer.* $\square$

   (b) Using the notion of "without loss of generality", prove the above statement.

   Proof:

   - *Let x and y be integers of opposite parity. Without loss of generality, suppose x is even and y is odd. By the definition, $x = 2k$ and $y = 2l+1$, where k and l are integers.*
   - *Then $5x+5y = 10k+10l+5 = 2(5k+5l+2)+1$, which is odd.*
   - *Therefore, if x and y are integers of opposite parity, then $5x+5y$ is an odd integer.* $\square$

8. (CSC 248) Consider the following statement:

   There exist rational numbers x and y such that x^y is irrational.

   Recall that existence can be proven by constructing a satisfying example, if one can be found.

   (a) Prove the above statement.

   Proof:

   - *Consider the rational numbers* $x = 2$ *and* $y = {}^1/_2$. *Then* $x^y = \sqrt{2}$, *which is irrational.*
   - *Therefore, there exist rational numbers* x *and* y *such that* x^y *is irrational.* $\square$

   (b) Is your proof constructive or non-constructive?

   Constructive

9. (CSC 248) The *Cartesian product* of two sets is defined as follows:

   Definition: The Cartesian product of sets A and B, denoted $A \times B$, is the set $\{(x, y) \mid x \in A \wedge y \in B\}$.

   Prove the following statement:

   Let A, B, C, and D be sets. If $A \subseteq C$ and $B \subseteq D$, then $A \times B \subseteq C \times D$.

   Proof:

   - *Let* A, B, *and* C *be sets such that* $A \subseteq C$ *and* $B \subseteq D$. *Further, let* $(x, y) \in A \times B$. *By definition,* $x \in A$ *and* $y \in B$.
   - *Since* $A \subseteq C$, $x \in C$, *and since* $B \subseteq D$, $y \in D$. *Thus,* $(x, y) \in C \times D$.
   - *Therefore, if* $A \subseteq C$ *and* $B \subseteq D$, *then* $A \times B \subseteq C \times D$. $\square$

10. (CSC 248) The *internal nodes* and *leaves* of a full binary tree are defined recursively as follows:

    Basis Step: A full binary tree consisting of a single node has $i = 0$ internal nodes and $l = 1$ leaves.

    Recursive Step: If T_1 and T_2 are disjoint full binary trees with i_1 and i_2 internal nodes and l_1 and l_2 leaves, respectively, then the full binary tree T consisting of a root r with T_1 and T_2 as its subtrees has $i = i_1 + i_2 + 1$ internal nodes and $l = l_1 + l_2$ leaves.

    Prove the following statement:

    Let T be a full binary tree. If T has i internal nodes and l leaves, then $l = i + 1$.

    Proof:

    - Basis Step:
      - *Let* $i = 0$. *Then* T *has just one node, which is a leaf, thus,* $l = 0 + 1 = 1$.
    - Inductive Hypothesis:
      - *Suppose that, for all* $0 \leq i \leq k$, *if* T *has* i *internal nodes, then it has* $l = i + 1$ *leaves.*
    - Inductive Step:
      - *Let* $i = k + 1$. *Since* $k \geq 0$, $i \geq 1$. *Then* T *consists of a root* r *with two non-empty subtrees,* T_1 *and* T_2, *with* i_1 *and* i_2 *internal nodes, respectively.*
      - *Note that* $i = k + 1 = i_1 + i_2 + 1$, *thus,* $i_1 \leq k$ *and* $i_2 \leq k$. *By the hypothesis,* T_1 *and* T_2 *contain* $l_1 = i_1 + 1$ *and* $l_2 = i_2 + 1$ *leaves, respectively. Thus,* $l = l_1 + l_2 = i_1 + 1 + i_2 + 1 = i + 1$.
    - *Therefore, by the Principle of Mathematical Induction, if a full binary tree* T *has* i *internal nodes and* l *leaves, then* $l = i + 1$. $\square$

11. (CSC 248) Solve each of the following recurrence relations:
    (a) $a_n = a_{n-1} - n$, where $a_0 = 4$ $a_n = 4 - \frac{n(n+1)}{2}$
    (b) $a_n = 2a_{n-1} - 3$, where $a_0 = -1$ $a_n = 3 - 2^{n+2}$

12. (CSC 248) Give the smallest Big-O estimate for each of the following functions:
    (a) $f(n) = (n! + 2^n)\left(n^3 + \log_2\left(n^2+1\right)\right)$ $O\left(n! \cdot n^3\right)$
    (b) $f(n) = \left(n^3 + n^2 \log_2 n\right)(\log_2 n + 1) + (17 \log_2 n + 19)\left(n^3 + 2\right)$ $O\left(n^3 \log n\right)$

13. (CSC 202 and CSC 248) Suppose that the complexity of an algorithm is given by a function $T(n)$.
    (a) What would it mean for $T(n)$ to be $O(1)$ — can an algorithm have complexity $O(1)$?

    A function that is $O(1)$ is non-increasing, bound above by a constant. Algorithms that take constant time have complexity $O(1)$.

    (b) What would it mean for $T(n)$ to be $\Omega(1)$ — can an algorithm have complexity $\Omega(1)$?

    A function that is $\Omega(1)$ is non-decreasing, bound below by a constant. All algorithms, in practice, have complexity $\Omega(1)$; they can't get indefinitely faster as their inputs grow indefinitely larger.

    (c) What would it mean for $T(n)$ to be $\Theta(1)$ — can an algorithm have complexity $\Theta(1)$?

    A function that is $\Theta(1)$ is neither increasing nor decreasing, bound both above and below by constants. Algorithms that take constant time have complexity $\Theta(1)$.

14. (CSC 248) The English alphabet contains 26 lowercase letters.
    (a) How many strings of lowercase English letters have length 4?

    For each of the 4 letters in the string, there are 26 possible options: 26^4

    (b) How many strings of 4 lowercase English letters *do not* contain the letter '`x`'?

    For each of the 4 letters in the string, there are now only 25 possible options: 25^4

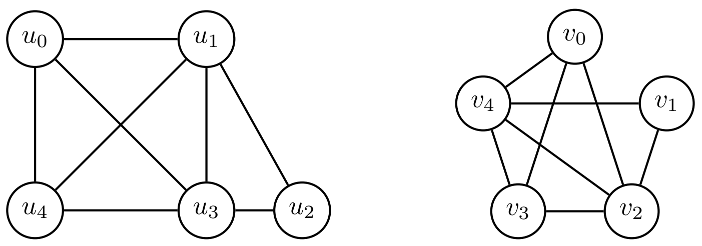
    (c) How many strings of 4 lowercase English letters *do* contain the letter '`x`'?

    Every string either contains '`x`' or does not contain '`x`': $26^4 - 25^4$

15. (CSC 248) Two graphs are *isomorphic* if and only if they have the same "shape", defined as follows:

    Definition: Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if and only if there exists a bijection $f : V_1 \to V_2$ such that $(u, v) \in E_1$ if and only if $(f(u), f(v)) \in E_2$.

    Isomorphic graphs have the same underlying structure, and the *isomorphism* f "renames" the vertices of G_1 to match G_2. Give an isomorphism for the following graphs:



One possible isomorphism is $\{(u_0, v_0), (u_1, v_2), (u_2, v_1), (u_3, v_4), (u_4, v_3)\}$.