CSC236 Fall 2018
Assignment #2: induction
due November 2nd, 3 p.m.

The aim of this assignment is to give you some practice with proving facts about recurrences, with the time complexity of recursive algorithms, and proving the correctness of algorithms.

Your assignment must be **typed** to produce a PDF document **a2.pdf** (hand-written submissions are not acceptable). Also submit Python source for two functions in sorting.py. You may work on the assignment in groups of 1 or 2, and submit a single assignment for the entire group on MarkUs

1. Define $\mathcal{T}$ as the smallest such such that:

   (a) Symbol $* \in \mathcal{T}$.

   (b) If $t_1, t_2 \in \mathcal{T}$, then $(t_1 t_2) \in \mathcal{T}$

   Some examples of elements of $\mathcal{T}$ are $*$, $(**)$, and $((**)*)$.

   (a) How many elements of $\mathcal{T}$ have (a) 0 left parentheses (b) 1 left parentheses (c) 2 left parentheses, (d) 3 left parentheses, and (d) 4 left parentheses?

   sample solution: There is (a) one element of $\mathcal{T}$ with zero left parentheses: $*$, (b) one element of $\mathcal{T}$ with one left parentheses: $(**)$, (c) two elements of $\mathcal{T}$ with two left parentheses: $(*(**))$, and $((**)*)$, (d) five elements of $\mathcal{T}$ with three left parentheses, since we can enclose with parentheses each element from (a) concatenated with each element from (c) [yielding 2], each element from (c) with each element from (a) [yielding another 2], and each element from (b) with each element from (b) [yielding another 1].
   (d) There are fourteen elements of $\mathcal{T}$ with four left parentheses, counted as follows: enclose with outer left and right parentheses each element with zero left parentheses concatenated with each element with three left parentheses [5 elements]; then each element with one concatenated with each element with two left parentheses [yielding 2 more]; then each element with two concatenated with each element with one left parentheses [2 more]; finally each element with three concatenated with each element with zero left parentheses [5 more], for a total of 14.

   (b) Devise a recurrence $c(n)$ such that $c(n)$ is the number of different elements of $\mathcal{T}$ with n left parentheses. Explain why your $c(n)$ is correct.

   The only element of $\mathcal{T}$ with zero left parentheses is $*$, so $c(0) = 1$. Every other "size" of elements in $\mathcal{T}$ is formed by an enclosing pair of parentheses (so one left parentheses there) around a pair of concatenated element contributing their own left parentheses. For a given n we count the number of ways two concatenated elements can contribute $n - 1$ left parentheses, which is the product of the number of elements with i left parentheses in the left-most element, with the number of elements with $n-i-1$ elements in the right-most element. Putting these together gives:

$$c(n) = \begin{cases} 1 & \text{if } n = 0 \\ \sum_{i=0}^{i=n-1} c(i)c(n-i-1) & \end{cases}$$

2. Devise a function $p(n)$ such that $p(n)$ is the number of different ways to create postage of n cents using 3-, 4-, and 5-cent stamps.

   (a) Carefully explain why your $p(n)$ is correct.

   sample solution: I build up a recurrent $p(n)$ in three steps:

   i. First I define a pair of functions. Function $t(n)$ calculates how many solutions there are for postage n with only 3-cent stamps. This is either exactly 1, if $n \equiv 0 \pmod 3$, or 0 if n is not a multiple of 3. Similarly function $f(n)$ calculates how many solutions are are for postage n with only 4-cent stamps.

   $$t(n) = \begin{cases} 1 & \text{if } n \equiv 0 \pmod 3 \\ 0 & \text{if } n \not\equiv 0 \pmod 3 \end{cases} \qquad f(n) = \begin{cases} 1 & \text{if } n \equiv 0 \pmod 4 \\ 0 & \text{if } n \not\equiv 0 \pmod 4 \end{cases}$$

   ii. Second, a pair of functions that calculate **the same thing in different ways.** $ft(n)$ calculates how many solutions there are for postage n with only 3- and 4-cent stamps. It does this by partitioning solutions into two sets: those that use only 3-cent stamps, and those that use at least one 4-cent stamp, which is the same as adding a 4-cent stamp to each solution for making postage of $n-4$ cents using only 3- and 4- cent stamps. Mirror image $tf(n)$ calculates the same thing by partitioning solutions into two different sets: those that use only 4-cent stamps, and those that use at least one 3-cent stamps, equivalent to adding a 3-cent stamp to each solution for making postage of $n-3$ using 3- and 4-cent stamps:

   $$ft(n) = \begin{cases} 0 & \text{if } n < 0 \\ t(n) + ft(n-4) & \text{if } n >= 0 \end{cases} \qquad tf(n) = \begin{cases} 0 & \text{if } n < 0 \\ f(n) + tf(n-3) & \text{if } n >= 0 \end{cases}$$

   iii. Third, function $p(n)$ calculates how many solutions there are for postage n using 3-, 4-, or 5-cent stamps. It does this by partitioning the solutions into two sets: those that use only 3- and 4-cent stamps, and those that use at least one 5-cent stamps, the latter being equivalent to adding one 5-cent stamp to each solution for making postage of $n-5$ using 3-, 4-, and 5-cent stamps.

   $$p(n) = \begin{cases} 0 & \text{if } n < 0 \\ 1 & \text{if } n = 0 \\ tf(n) + p(n-5) & \text{if } n > 0 \end{cases}$$

   another sample solution: Use the inclusion-exclusion principle to count all solutions that use at least one 3-cent stamp, plus those that use at least one 4-cent stamp, plus those that use at least one 5-cent stamp — these solutions correspond to adding a 3-cent, 4-cent, or 5-cent stamp to the postage for $n-3$, $n-4$, or $n-5$, respectively. Then subtract the over-counted solutions that use both a 3- and a 4-cent stamps, both a 3- and a 5-cent stamp, and both a 4- and a 5-cent stamp. Finally, add back in the undercounted solutions that use a 3-, a 4-, and a

5-cent stamp. Be sure to deal with negative postage having 0 solutions.

$$p(n) = \begin{cases} 0 & \text{if } n < 0 \\ 1 & \text{if } n = 0 \\ p(n-3) + p(n-4) + p(n-5) & \\ \quad -p(n-7) - p(n-8) - p(n-9) + p(n-12) & \text{if } n > 0 \end{cases}$$

(b) Prove that $p(n)$ is monotonic nondecreasing on $\mathbb{N}^+$

sample solution: Define the predicate:

$$P(n) : p(n) - p(n-1) \geq 0$$

I will prove by by complete induction that $\forall n \in \mathbb{N} - \{0,1\}, P(n)$. I will prove this for each independent definition of $p(n)$ above:

using the second definition of $p(n)$: This uses the definition:

$$p(n) = \begin{cases} 0 & \text{if } n < 0 \\ 1 & \text{if } n = 0 \\ p(n-3) + p(n-4) + p(n-5) & \\ \quad -p(n-7) - p(n-8) - p(n-9) + p(n-12) & \text{if } n > 0 \end{cases}$$

inductive step: Let n be an arbitrary natural number no smaller than 2. Assume $\bigwedge_{i=2}^{i=n-1} P(i)$. I will show that $P(n)$ follows, that is $p(n) - p(n-1) \geq 0$.

case $2 \leq n < 14$: It is straightforward, although a bit labourious, to verify $P(2) \wedge P(3) \wedge \cdots \wedge P(13)$.

case $n \geq 14$: Subtracting $p(n) - p(n-1)$ yields:

$$\begin{aligned} p(n) - p(n-1) &= p(n-3) + p(n-4) + p(n-5) \\ &\qquad - p(n-7) - p(n-8) - p(n-9) + p(n-12) \qquad \# \text{ by definition, } n > 0 \\ &\quad -p(n-4) - p(n-5) - p(n-6) \\ &\qquad + p(n-8) + p(n-9) + p(n-10) - p(n-13) \qquad \# \text{ by definition, } n-1 > 0 \\ &= p(n-3) - p(n-6) - p(n-7) + p(n-10) + p(n-12) - p(n-13) \\ &\qquad \# \text{ subtract equivalent terms} \\ &\geq p(n-3) - p(n-6) - p(n-7) + p(n-10) \qquad \# \text{ by IH, } 2 \leq n-12 < n \\ &\quad \# \text{ this is postage for } n-3 \text{ that uses no 3- or 4-cent stamps} \\ &\geq 0 \quad \blacksquare \end{aligned}$$

using first definition of $p(n)$: This uses the definition of p in terms of t, f and p itself.

inductive step: Let n be an arbitrary natural number no smaller than 2. Assume $\bigwedge_{i=2}^{i=n-1} P(i)$. I will show that $P(n)$ follows, that is $p(n) - p(n-1) \geq 0$.

case $2 \leq n < 17$: It is straightforward, although a bit labourious, to verify $P(2) \wedge P(3) \wedge \cdots \wedge P(16)$.

case $n \geq 17$: Recall the definitions of $t(n), f(n), tf(n), ft(n)$, and $p(n)$.

$$\begin{aligned}
p(n) - p(n-1) &= ft(n) + p(n-5) - ft(n-1) - p(n-6) \\
&\qquad \text{\# use definition of } p(n), p(n-1) \\
&= ft(n) + ft(n-5) + ft(n-10) + p(n-15) \\
&\qquad -ft(n-1) - ft(n-6) - ft(n-11) - p(n-16) \\
&\qquad \text{\# use definition of } p \text{ four more times} \\
&\geq ft(n) + ft(n-5) + ft(n-10) \\
&\qquad -ft(n-1) - ft(n-6) - ft(n-11) \\
&\qquad \text{\# by IH, } 2 \leq n-15 < n \Rightarrow p(n-15) - p(n-16) \geq 0 \\
&= ft(n) + ft(n-5) + ft(n-10) \\
&\qquad -tf(n-1) - tf(n-6) - tf(n-11) \qquad \text{\#} tf(n) = ft(n) \\
&= t(n) + ft(n-4) + t(n-5) + ft(n-9) + t(10) + ft(n-14) \\
&\qquad -f(n-1) - tf(n-4) - f(n-6) - tf(n-9) \\
&\qquad -f(n-11) - tf(n-14) \\
&\qquad \text{\# use definitions of } ft(n), ft(n-5), ft(n-10) \\
&\qquad \text{\# use definitions of } tf(n-1), tf(n-6), tf(n-11) \\
&= t(n) + t(n-5) + t(10) - (f(n-1) + f(n-6) + f(n-11)) \\
&\qquad \text{\# } ft(n-4) = tf(n-4), \text{ etcetera} \\
&= t(n) + t(n-2) + t(n-1) - (f(n-1) + f(n-2) + f(n-3)) \\
&\qquad \text{\# } -5 \equiv -2 \bmod 3, -10 \equiv -1 \bmod 3 \\
&\qquad \text{\#} -6 \equiv -2 \bmod 4, -11 \equiv -3 \bmod 4 \\
&\geq 0 \quad \blacksquare \\
&\qquad \text{\# } t(n) + t(n-2) + t(n-1) = 1 \\
&\qquad \text{\# exactly one of } n, n-1, \text{ or } n-2 \text{ is a multiple of 3} \\
&\qquad \text{\# } f(n-1) + f(n-2) + f(n-3) \leq 1 \\
&\qquad \text{\# at most one of } n-1, n-2, n-3 \text{ is a multiple of 4}
\end{aligned}$$

3. Consider the recurrence T that we derived for the worst-case time complexity of `recBinSearch`:

$$T(n) = \begin{cases} c' & \text{if } n = 1 \\ 1 + T(\lceil n/2 \rceil) & \text{if } n > 1 \end{cases}$$

(a) Emulate Lemma 3.6 from the course notes to prove that T is nondecreasing.

sample solution: Define:

$$P(n) : \bigwedge_{m=1}^{m=n} T(m) \leq T(n)$$

I will prove $\forall n \in \mathbb{N}^+, P(n)$ using complete induction. I will assume, without proof, that if n is a natural number greater than 1, then $n > \lceil n/2 \rceil \geq 1$.

inductive step: Let $n \in \mathbb{N}^+$. Assume $\wedge_{i=1}^{i=n-1} P(i)$. I will show that $P(n)$ follows.

base case $n < 3$: $T(1) = c'$, which is no smaller than the value of T on any smaller positive natural number — since there are no smaller positive natural numbers — so $P(1)$ holds. By the definition $T(2) = 1 + T(1) \geq T(1)$, so $P(2)$ holds.

case $n \geq 3$: Since $n > 2$ we know that $n > n-1 \geq 1$ so by our IH we know that $P(n-1)$ holds and (by transitivity of $\leq$) we need only show that $T(n-1) \leq T(n)$. Since $n-1 > 1$ we have

$$\begin{aligned} T(n-1) &= 1 + T(\lceil (n-1)/2 \rceil) && \text{\# by definition, } n-1 > 1 \\ &\leq 1 + T(\lceil n/2 \rceil) && \text{\# by IH, } 1 \leq \lceil (n-1)/2 \rceil \leq \lceil n/2 \rceil < n \\ &= T(n) \quad \blacksquare \end{aligned}$$

(b) Use simple induction on k to prove that $\forall k, n \in \mathbb{N}, n = 2^k \Rightarrow T(n) = \lg(n) + c'$.

sample solution: Define $P(k) : T(2^k) = k + c'$. Prove $\forall k \in \mathbb{N}, P(k)$ by simple induction.

base case: $T(2^0) = T(1) = c' = 0 + c'$, so $P(0)$ holds.

inductive step: Let $k \in \mathbb{N}$. Assume $P(k)$. I will show that $P(k+1)$ follows:

$$\begin{aligned} T(2^{k+1}) &= 1 + T(2^k) && \text{\# by definition, } 2^{k+1} > 1 \\ &= 1 + k + c' && \text{\# by IH} \\ &= k + 1 + c' && \blacksquare \end{aligned}$$

(c) Combine the previous two steps to prove that $T \in \Theta(\lg)$. Do **not** use induction.

big-Oh: First I prove that there exists $d \in \mathbb{R}^+$ and $B \in \mathbb{N}$ such that for all $n \in \mathbb{N}, n \geq B \Rightarrow T(n) \leq \lg(n)$. As a convenience define:

$$n^* = 2^{\lceil \lg n \rceil}, \text{ for } n \in \mathbb{N}$$

Notice that $n^*/2 < n \leq n^* < 2n$.
Let $d = 2 + c'$. Then $d \in \mathbb{R}^+$. Let $B = 2$. Then $B \in \mathbb{N}$. Let n be an arbitrary natural number no smaller than B.

$$\begin{aligned} T(n) &\leq T(n^*) && \text{\# } T \text{ nondecreasing and } n \leq n^* \\ &= \lg(n^*) + c' && \text{\# previous result} \\ &\leq \lg(2n) + c' = \lg(n) + 1 + c' && \text{\# lg nondecreasing and } n^* < 2n \\ &\leq \lg(n) + \lg(n)(1 + c') = (2 + c')\lg(n) && \text{\# since } n \geq B = 2 \\ &= d\lg(n) \quad \blacksquare && \text{\# } d = 2 + c' \end{aligned}$$

So $T \in \mathcal{O}(\lg)$.

Ω: I will prove that there exists $d \in \mathbb{R}^+$ and $B \in \mathbb{N}$ such that $\forall n \in \mathbb{N}$, $n \geq B \Rightarrow T(n) \geq d\lg(n)$. For convenience, use the definition of n^* above.
Let $d = 1/2$. Then $d \in \mathbb{R}^+$. Let $B = 4$. Then $B \in \mathbb{N}$. Let n be an arbitrary natural number no smaller than B.

$$\begin{aligned} T(n) &\geq T(n^*/2) && \text{\# } T \text{ nondecreasingg and } n > n^*/2 \\ &= \lg(n^*/2) + c' && \text{\# previous result} \\ &\geq \lg(n/2) + c' = \lg(n) - 1 + c' && \text{\# } n^*/2 \geq n/2 \\ &= \lg(n)/2 + \lg(n)/2 - 1 + c' \geq \lg(n)/2 + c' && \text{\# } n \geq B = 4 \Rightarrow \lg(n)/2 \geq 1 \\ &\geq \lg(n)/2 = d\lg(n) \quad \blacksquare && \text{\# } d = 1/2 \end{aligned}$$

4. Suppose you take the trouble to legally change your name to a string from the alphabet $\{A, C, T, G\}$, for example $TAGAC$ might make a fine name. Then you (discreetly) collect DNA samples from your

friends, e.g. nail clippings, hair, used coffee cups. Your idea is to count the number of times your name occurs as a subsequence of their DNA (you may need some help sequencing that), and then invoice them for that many licenses of your intellectual property. If your friend's DNA contained the string $ATAGGACCA$ they'd owe you for at least four licenses!

Clearly you'll need some computational help counting sequences. Read over the code below and either (a) prove that its precondition plus execution implies its postcondition, or (b) provide a counterexample that shows it is incorrect.

```
def count_subsequences(s1: str, s2: str,
                       i: int, j: int) -> int:
    """ Return the number of times s1[: i] occurs as a
    subsequence of s2[: j].

    Precondition: 0 <= i <= len(s1), 0 <= j <= len(s2)

    >>> count_subsequences("", "Danny", 0, 5)
    1
    >>> count_subsequences("Danny", "", 5, 0)
    0
    >>> count_subsequences("AAA", "AAAAA", 3, 5)
    10

    Postcondition: returns number of times s1[: i] occurs as a
    subsequence of s2[: j]
    """
    if i == 0:
        return 1
    elif i > j:
        return 0
    elif s1[i-1] != s2[j-1]:
        return count_subsequences(s1, s2, i, j-1)
    else:
        return (count_subsequences(s1, s2, i, j-1)
                + count_subsequences(s1, s2, i-1, j-1))
```

You may also find the above code, plus an efficient **memoized** version, in sequences.py

sample solution: I prove count_subsequences correct in terms of its precondition and postcondition using simple induction on j.[1] Let s1, s2 be strings. Let $P(j)$: "$\forall i \in \mathbb{N}$ if $0 \leq i \leq \text{len}(\text{s1}), 0 \leq j \leq \text{len}(\text{s2})$, then count_subsequences(s_1, s_2, i, j) returns the number of times s1[: i] occurs as a subsequence of s2[:]." Notice that I quantify i **within** the predicate, since I will want to use more than one fixed instance of i in the inductive step.

base case: Let $i \in \mathbb{N}$. There are two cases to consider, depending on whether i is zero or positive.

case $i = 0$: Then s1[:i] is the empty string, which occurs exactly once as a subsequence of any string, which is what the code returns, so $P(0)$ holds.

case $i > 0$: Then s1[:i] is a string of positive length and cannot occur as a subsequence of the empty string, and the code returns 0, so $P(0)$ holds.

inductive step: Let $j \in \mathbb{N}$. Assume $P(j)$. I will show that $P(j+1)$ follows. Let $i \in \mathbb{N}$. There are several cases to consider:

case $i = 0$: Again s1[:i] is an empty string, and it occurs exactly once as a subsequence of any string, which is what the code returns.

[1]There are other workable options. Also note that, since there are no loops nor function calls other than recursive sub-calls, termination is equivalent to the recursive sub-calls returning.

case $i > j + 1$: Then s1[:i] is longer than s2[:$j + 1$], so it cannot occur as a subsequence, and the code returns 0.

case $i \leq j + 1$ **and s1[:**$i - 1$**] != s2[:**j**]**: In this case the only occurrences of s1[:i] as a subsequence of s2[:$j + 1$] must occur without the final character of s1[:i] matching the final character of s2[:$j + 1$], thus they must be subsequences of s2[:j], which is what count_subsequences(s1, s2, i, j) returns, according to the IH.

case $i \leq j + 1$ **and s1[:**$i - 1$**] = s2[:**j**]**: occurrences of s1[:i] as a subsequence of s2[:$j + 1$] can be partitioned into those that agree on the last character, and those that do not. The number of subsequences where the last characters do not agree is the same as the previous case, and the number where they do agree is equivalent to the number of times s1[:$i - 1$] occurs as a subsequence s2[:j]. The sum of the number of subsequences in these two partitions is the number of subsequences we seek, and this is what count_subsequences(s1, s2, i, j) + count_subsequences(s1, s2, i-1, j) returns, according to the IH.

So $P(j + 1)$ follows in all possible cases. ■

5. Read over unimplemented function shade_sort below:

```
def shade_sort(colour_list: List[str]) -> None:
    """ Put colour_list in order "b" < "g" < "r".

    precondition: colour_list is a List[str] from {"b", "g", "r"}

    >>> list_ = ["r", "b", "g"]
    >>> shade_sort(list_)
    >>> list_ == ["b", "g", "r"]
    True

    postcondition: colour_list has same strings as before, ordered "b" < "g" < "r"
    """
    # TODO: initialize blue, green, red to be consistent with loop invariants.
    # Hint: blue, green may increase while red decreases.
    #
    # loop invariants:
    #
    # 0 <= blue <= green <= red <= len(colour_list)
    # colour_list[0 : green] + colour_list[red :] same colours as before
    # and all([c == "b" for c in colour_list[0 : blue]])
    # and all([c == "g" for c in colour_list[blue : green]])
    # and all([c == "r" for c in colour_list[red :]])
    #
    # TODO: implement loop using invariants above!
```

implement: Use the invariants as a guide to implement function shade_sort.

sample solution: Below is an implementation. The loop invariants suggest that the Python slices are incrementally increased to contain the ”b”, ”g”, and ”r” in the correct order.

```
# TODO: initialize blue, green, red to be consistent with loop invariants.
# Hint: blue, green may increase while red decreases.
blue, green, red = 0, 0, len(colour_list)
# loop invariants:
#
# 0 <= blue <= green <= red <= len(colour_list)
# colour_list[0 : green] + colour_list[red :] same colours as before
# and all([c == "b" for c in colour_list[0 : blue]])
# and all([c == "g" for c in colour_list[blue : green]])
# and all([c == "r" for c in colour_list[red :]])
#
```

```
        # TODO: implement loop using invariants above!
        while green < red:
            if colour_list[green] == "r" and colour_list[red - 1] == "r":
                red -= 1
            elif colour_list[green] == "r" and colour_list[red - 1] == "g":
                colour_list[green], colour_list[red - 1] = (colour_list[red - 1],
                                                            colour_list[green])
                green, red = green + 1, red - 1
            elif colour_list[green] == "r" and colour_list[red - 1] == "b":
                colour_list[green], colour_list[red - 1] = (colour_list[red - 1],
                                                            colour_list[green])
                red -= 1
                colour_list[blue], colour_list[green] = colour_list[green], colour_list[blue]
                blue, green = blue + 1, green + 1
            elif colour_list[green] == "b":
                colour_list[blue], colour_list[green] = colour_list[green], colour_list[blue]
                blue, green = blue + 1, green + 1
            else: # colour_list[green] == "g"
                green += 1
```

prove: Use those same invariants to prove that your function shade_sort is correct: first partial correctness, then termination.

sample solution: I first prove partial correctness, that is the precondition, plus execution, assuming termination, imply the postcondition.

First I define loop invariant, where blue_i is the value of blue after the ith iteration, green_i is the value of green after the ith iteration, and red_i is the value of red after the ith iteration.[2]

$P(i)$: After the ith iteration of the loop (if it occurs):

```
(a) red_i - green_i \in \Z
(b) 0 <= blue_i <= green_i <= red_i <= len(colour_list)
(c) colour_list[0 : green_i] + colour_list[red_i :] same colours
      as before, possibly permuted
(d) all([c == "b" for c in colour_list[0 : blue_i]])
(e) all([c == "g" for c in colour_list[blue : green_i]])
(f) all([c == "r" for c in colour_list[red_i :]])
(g) colour_list[green_i: red_i] is unchanged
```

Prove $\forall i \in \mathbb{N}, P(i)$ using simple induction. Let $n = \text{len(colour\_list)}$.

base case $i = 0$: In this case the loop has not iterated even once, so by the initialization code

(a) $\text{green}_0 = 0$, an integer, and $\text{red}_0 = n$, also an integer.

(b) $\text{blue}_0 = \text{green}_0 = 0 \leq \text{red}_0 = \text{n}$, verifying this invariant.

(c) No assignments to list elements have been made, which verifies this invariant.

(d) $\text{colour\_list}[0{:}\text{blue}_i]$ is empty, so this invariant is vacuously true.

(e) $\text{colour\_list}[0{:}\text{green}_0]$ is empty, so this invariant is vacuously true.

(f) $\text{colour\_list}[\text{red}_i{:}]$ is empty, so this invariant is vacuously true.

(g) No assignments to list elements have been made, which verifies this invariant..

So $P(0)$ follows.

inductive step: Let $i \in \mathbb{N}$. Assume $P(i)$. I will show that $P(i+1)$ follows.

If there is an $(i+1)th$ iteration then $\text{green}_i < \text{red}_i$, according to the loop condition. There are five clauses in the **if** condition to consider:

case colour_list[green$_i$] == "r" == colour_list[red$_i$ - 1]: By the code in this clause:

[2] I number them and add subscripts, for convenience. Also, I add a sixth invariant which turns out to be useful.

(a) $\text{red}_{i+1} = \text{red}_i - 1$ and $\text{green}_{i+1} = \text{green}_i$, and by $P(i)$ $\text{red}_i - \text{green}_i$ is an integer, so $\text{red}_{i+1} - \text{green}_{i+1} = \text{red}_i - \text{green}_i - 1$ is also an integer, since integers are closed under subtraction.

(b) $\text{blue}_{i+1} = \text{blue}_i$, $\text{green}_{i+1} = \text{green}_i$, and $\text{red}_{i+1} = \text{red}_i - 1$. Combining this with the IH and the fact (above) that $\text{green}_i < \text{red}_i$, we have: $0 \leq \text{blue}_{i+1} \leq \text{green}_{i+1} \leq \text{red}_{i+1} < \text{red}_i \leq n$.

(c) By IH(c), and the fact that no assignments were made to colour_list, so the colours remain the same as before.

(d) colour_list[0:blue_{i+1}] = colour_list[0:blue_i], so this follows from IH(d).

(e) colour_list[blue_{i+1} :green_{i+1}] = colour_list[blue_i:green_i], so this follows from IH(e).

(f) By IH(f), and colour_list[$\text{red}_i - 1$] = "r" = colour_list[red_{i+1}], this follows.

(g) By IH(g), and the fact that no assignments were made to colour_list, this follows.

case colour_list[green_i] == "r" and colour_list[$\text{red}_i - 1$] == "g": By the code in this clause:

(a) $\text{red}_{i+1} = \text{red}_i - 1$ and $\text{green}_{i+1} = \text{green}_i + 1$, and by IH(a) $\text{red}_i - \text{green}_i$ is an integer, so $\text{red}_{i+1} - \text{green}_{i+1} = \text{red}_i - \text{green}_i - 2$ is also an integer, since integers are closed under subtraction.

(b) $\text{blue}_{i+1} = \text{blue}_i$, $\text{green}_{i+1} = \text{green}_i + 1$, and $\text{red}_{i+1} = \text{red}_i - 1$. We also know both that $\text{green}_i < \text{red}_i$ (loop condition) and $\text{green}_i \neq \text{red}_i - 1$ (case condition), so $\text{green}_i < \text{red}_{i+1}$, and thus $\text{green}_i + 1 \leq \text{red}_{i+1}$. Combining these with IH(b)

$$0 \leq \text{blue}_i = \text{blue}_{i+1} \leq \text{green}_i \leq \text{green}_{i+1} \leq \text{red}_{i+1} = \text{red}_i - 1 \leq \text{red}_i \leq n$$

(c) By IH(c), IH(g), and the fact that colour_list[green_i] was swapped with colour_list[$\text{red}_i - 1$], this follows.

(d) colour_list[0:blue_{i+1}] = colour_list[0:blue_i], so this follows from IH(d).

(e) By IH(e), and the fact that colour_list[$\text{green}_{i+1} - 1$] = "g", this follows.

(f) By IH(f), and the fact that colour_list[red_{i+1}] = "r", this follows.

(g) By IH(g), and the fact that there were no assignments to colour_list[green_{i+1}:red_{i+1}], this follows.

case colour_list[green_i] == "r" and colour_list[$\text{red}_i - 1$] == "b": By the code in this clause:

(a) $\text{red}_{i+1} = \text{red}_i - 1$ and $\text{green}_{i+1} = \text{green}_i + 1$, and by IH(a) $\text{red}_i - \text{green}_i$ is an integer, so $\text{red}_{i+1} - \text{green}_{i+1} = \text{red}_i - \text{green}_i - 2$ is also an integer, since integers are closed under subtraction.

(b) $\text{blue}_{i+1} = \text{blue}_i + 1$, $\text{green}_{i+1} = \text{green}_i + 1$, and $\text{red}_{i+1} = \text{red}_i - 1$. We also know both that $\text{green}_i < \text{red}_i$ (loop condition) and $\text{green}_i \neq \text{red}_i - 1$ (case condition), so $\text{green}_i < \text{red}_{i+1}$, and thus $\text{green}_i + 1 \leq \text{red}_{i+1}$. Combining these with IH(b)

$$0 \leq \text{blue}_i < \text{blue}_i + 1 = \text{blue}_{i+1} \leq \text{green}_i + 1 = \text{green}_{i+1} \leq \text{red}_i - 1 = \text{red}_{i+1} < \text{red}_i \leq n$$

(c) By IH(c), IH(g), and the fact that the only assignments to colour_list were that colour_list[green_i] was swapped with colour_list[$\text{red}_i - 1$], and that then colour_list[green_i] was swapped with colour_list[blue_i], this follows.

(d) By IH(d), the fact that $\text{blue}_{i+1} = \text{blue}_i + 1$, and the fact that colour_list[$\text{blue}_{i+1} - 1$] = "b", this follows.

(e) By IH(e), the fact that $\text{blue}_{i+1} = \text{blue}_i + 1$, the fact that $\text{green}_{i+1} = \text{green}_i + 1$, and the fact that colour_list[$\text{green}_{i+1} - 1$] = "g", this follows.

(f) By IH(f), and the fact that colour_list[red_{i+1}] = "r", this follows.

(g) By IH(g), and the fact that there were no assignments to colour_list[green_{i+1}:red_{i+1}], this follows.

case colour_list[green_i] = "b": By the code in this clause:

(a) $\text{green}_{i+1} = \text{green}_i + 1$, and $\text{red}_{i+1} = \text{red}_i$, and by IH(a) $\text{red}_i - \text{green}_i$ is an integer, so $\text{red}_{i+1} - \text{green}_{i+1} = \text{red}_i - \text{green}_i - 1$ is also an integer, since integers are closed under subtraction.

(b) $\text{blue}_{i+1} = \text{blue}_i + 1$, $\text{green}_{i+1} = \text{green}_i + 1$, and $\text{red}_{i+1} = \text{red}_i$. We also know (loop condition) that $\text{green}_i < \text{red}_i$, so $\text{green}_i + 1 \leq \text{red}_i = \text{red}_{i+1}$. Combining these with IH(b):

$$0 \leq \text{blue}_i < \text{blue}_i + 1 = \text{blue}_{i+1} \leq \text{green}_i + 1 = \text{green}_{i+1} \leq \text{red}_{i+1} = \text{red}_i \leq n$$

(c) By IH(c), IH(g), and the fact that the only assignments to colour_list were that colour_list[blue_i] was swapped with colour_list[green_i], this follows.

(d) By IH(d), and the fact that colour_list[$\text{blue}_{i+1} - 1$] = "b", this follows.

(e) By IH(e), and the fact that colour_list[$\text{green}_{i+1} - 1$] = "g", this follows.

(f) By IH(f), and the fact that $\text{red}_i = \text{red}_{i+1}$, this follows.

(g) By IH(g), and the fact that there were no assignments to colour_list[green_{i+1}:red_{i+1}], this follows.

case colour_list[green_i] = "g": By the code in this clause:

(a) $\text{green}_{i+1} = \text{green}_i + 1$, and $\text{red}_{i+1} = \text{red}_i$, and by IH(a) $\text{red}_i - \text{green}_i$ is an integer, so $\text{red}_{i+1} - \text{green}_{i+1} = \text{red}_i - \text{green}_i - 1$ is also an integer, since integers are closed under subtraction.

(b) $\text{blue}_{i+1} = \text{blue}_i$, $\text{green}_{i+1} = \text{green}_i + 1$, and $\text{red}_{i+1} = \text{red}_i$. Also, by the loop condition, $\text{green}_i < \text{red}_i$, so $\text{green}_{i+1} = \text{green}_i + 1 \leq \text{red}_i = \text{red}_{i+1}$. Combining these with IH(b):

$$0 \leq \text{blue}_i = \text{blue}_{i+1} \leq \text{green}_i + 1 = \text{green}_{i+1} \leq \text{red}_{i+1} = \text{red}_i - 1 < \text{red}_i \leq n$$

(c) By IH(c), IH(g), and the fact that there were no assignment to colour_list, this follows.

(d) By IH(d), and the fact that $\text{blue}_{i+1} = \text{blue}_i$, this follows.

(e) By IH(e), and the fact that colour_list[green_i] = "g", this follows.

(f) By IH(f), and the fact that $\text{red}_{i+1} = \text{red}_i$, this follows.

(g) By IH(g), and the fact that there were no assignments to colour_list, this follows.

partial correctness: Assume the precondition, and that the loop terminates at, say, iteration f. This leads to two conclusions:

(a) $\text{green}_f \geq \text{red}_f$ # by the loop condition.

(b) $\text{green}_f \leq \text{red}_f$ # by $P(f)$ invariant 1

$\Rightarrow \text{green}_f = \text{red}_f$.

In addition, by $P(f)$ invariant 2 we know that colour_list[0: green_f] + colour_list[red_f:] have the same colours as before. Since $\text{green}_f = \text{red}_f$, this means that colour_list has the same colours as before. $P(f)$ invariants 3–5 ensure that the three, now consecutive, slices colour_list[0: blue_f], colour_list[blue_f: green_f], and colour_list[green_f:] contain, respectively, only "b", "g", "r", and since these slices comprise all of colour_list, these strings, in order, make up the entire list. ■

termination: It is enough to show that $\text{red}_i - \text{green}_i$ is a strictly decreasing sequence of natural numbers. We know the elements of the sequence are natural numbers since, by our loop invariant, for every $i \in \mathbb{N}$, we know that at the end of iteration i, if it occurs, $\text{red}_i - \text{green}_i$ is an integer (invariant (a)), and that $\text{red}_i - \text{green}_i$ is non-negative (invariant (b)). It remains to show that it is strictly decreasing.

Assume there is an $(i+1)$th iteration. There are several cases to consider:

case colour_list[green$_i$] = "r" = colour_list[red$_i$ − 1]: $\text{red}_{i+1} = \text{red}_i - 1$ and $\text{green}_{i+1} = \text{green}_i$, so

$$\text{red}_{i+1} - \text{green}_{i+1} = \text{red}_i - 1 - \text{green}_i < \text{red}_i - \text{green}_i.$$

case colour_list[green$_i$] = "r" and colour_list[red$_i$ - 1] = "g": $\text{red}_{i+1} = \text{red}_i - 1$ and $\text{green}_{i+1} = \text{green}_i + 1$, so

$$\text{red}_{i+1} - \text{green}_{i+1} = \text{red}_i - 1 - (\text{green}_i + 1) < \text{red}_i - \text{green}_i.$$

case colour_list[green$_i$] = "r" and colour_list[red$_i$ - 1] = "b": $\text{red}_{i+1} = \text{red}_i - 1$ and $\text{green}_{i+1} = \text{green}_i + 1$, so

$$\text{red}_{i+1} - \text{green}_{i+1} = \text{red}_i - 1 - (\text{green}_i + 1) < \text{red}_i - \text{green}_i.$$

case colour_list[green$_i$] = "b": $\text{red}_{i+1} = \text{red}_i$ and $\text{green}_{i+1} = \text{green}_i + 1$, so

$$\text{red}_{i+1} - \text{green}_{i+1} = \text{red}_i - (\text{green}_i + 1) < \text{red}_i - \text{green}_i.$$

case colour_list[green$_i$] = "g": $\text{red}_{i+1} = \text{red}_i$ and $\text{green}_{i+1} = \text{green}_i + 1$, so

$$\text{red}_{i+1} - \text{green}_{i+1} = \text{red}_i - (\text{green}_i + 1) < \text{red}_i - \text{green}_i.$$

In every case the sequence strictly decreases. ■

extend: Modify the invariants of `shade_sort` so they specify function `four_shade_sort` which adds a fourth string "y" (meaning yellow) that must end up following the other three when `colour_list` is sorted. Then implement function `four_shade_sort`.

Submit both implementations (including invariants) as **sorting.py**, along with **a2.pdf**. You will find the above code in sorting.py

sample solution: Here is an implementation of four_shade_sort:

```
def shade_sort3(colour_list: List[str]) -> None:
    """ Put colour_list in order "b" < "g" < "r" < "y".

    precondition: colour_list is a List[str] from {"b", "g", "r", "y"}

    # examples omitted...

    postcondition: colour_list has same strings as before, ordered "b" < "g" < "r" < "y"
    """
    # TODO: initialize blue, green, red, yellow to be consistent with loop invariants.
    # Hint: blue, green may increase while red, yellow decrease
    blue, green, red, yellow = 0, 0, len(colour_list), len(colour_list)
    # loop invariants:
```

```
#
# 0 <= blue <= green <= red <= yellow <= len(colour_list)
# colour_list[0 : green] + colour_list[red :] same colours as before
# and all([c == "b" for c in colour_list[0 : blue]])
# and all([c == "g" for c in colour_list[blue : green]])
# and all([c == "r" for c in colour_list[red :]])
# and all([c == "y" for c in colour_list[yellow :]])
#
# TODO: implement loop using invariants above!
def swap(i1, i2) -> None:
    # swap colour_list elements at i1 and i2
    colour_list[i1], colour_list[i2] = colour_list[i2], colour_list[i1]

while green < red:
    if colour_list[green] == "g":
        green += 1
    elif colour_list[red - 1] == "r":
        red -= 1
    elif colour_list[green] == "b":
        swap(blue, green)
        blue, green = blue + 1, green + 1
    elif colour_list[red - 1] == "y":
        swap(red - 1, yellow - 1)
        red, yellow = red - 1, yellow - 1
    elif colour_list[green] == "r" and colour_list[red - 1] == "g":
        swap(green, red - 1)
        green, red = green + 1, red - 1
    elif colour_list[green] == "r" and colour_list[red - 1] == "b":
        swap(green, red - 1)
        swap(blue, green)
        blue, green, red = blue + 1, green + 1, red - 1
    elif colour_list[green] == "y" and colour_list[red - 1] == "g":
        swap(green, red - 1)
        swap(red - 1, yellow - 1)
        green, red, yellow = green + 1, red - 1, yellow - 1
    else:
        assert colour_list[green] == "y" and colour_list[red - 1] == "b"
        swap(green, red - 1)
        swap(blue, green)
        swap(red - 1, yellow - 1)
        blue, green, red, yellow = blue + 1, green + 1, red - 1, yellow - 1
```