Q1. On pages 55 and 56 of the textbook there is a proof for the correctness of the program `avg`. The author used the invariant

$$Inv(\texttt{i}, \texttt{sum})\colon 0 \leq \texttt{i} < len(A) \wedge \texttt{sum} = \sum_{k=0}^{\texttt{i}-1} A[k].$$

As you can see, the invariant is a predicate in two variables: `i`, `sum`. These two variables are used in the program `avg`, but neither is the variable on which the induction proof is based. The author is using simple induction, but it is not very clear what the induction variable is (also the induction predicate itself is ambiguous). We want to make sure you understand what is going on there by having you re-write the proof by yourself in a style similar to the one we use in lectures. Here is the predicate you should be proving:

$$Q(j) : \text{At the beginning of the } j^{th} \text{ iteration}, \texttt{sum} = \sum_{k=0}^{\texttt{i}-1} A[k].$$

Remark 1: By the program's design, the variables `i`, `sum` may change with each iteration (in other words, both are functions of j). This is why it might be more appropriate to write

$$Q(j) : \texttt{sum}_j = \sum_{k=0}^{\texttt{i}_j-1} A[k] \quad \text{or} \quad Q(j) : \texttt{sum}(j) = \sum_{k=0}^{\texttt{i}(j)-1} A[k].$$

where $\texttt{i}_j$ (or $\texttt{i}(j)$) means the value of program variable `i` at the beginning of the j^{th} iteration (the same with `sum`). That said, we believe that the version above Remark 1 is the best to work with as long as one understands that `i`, `sum` are iteration dependent.

Remark 2: Using j as an index in Remark 1 has a different meaning from that which is intended by the author. The author is using indices to differentiate between the values of `i`, `sum` at the beginning of an (arbitrary) iteration and their values after the iteration is run. The end goal is to prove $Q(len(A))$. A proof by induction will show that

$$\forall j \in \{1, \ldots, len(A)\}, Q(j).$$

Your proof must follow the style used in lectures.

Proof. It will be shown using simple induction that $\forall j \in \{1, ..., \texttt{len}(A)\}, Q(j)$.

Base Case. Let $j = 1$. At the beginning of the first iteration of the loop, $i = 0$ and $sum = 0$. Notice that in the expression $\sum_{k=0}^{-1} A[k]$, the lower bound is greater than the upper bound. This results in an empty sum that evaluates to 0. Therefore the base case holds.

Induction Hypothesis. Suppose that for some $j \in \{1, ..., \texttt{len}(A)\}$, $Q(j)$ is true.

Induction Step. i increases by 1 at the end of each iteration, so at the beginning of the jth iteration, $i = j - 1 < \texttt{len}(A)$, which means that the loop body is entered. From the induction hypothesis,

$$sum = \sum_{k=0}^{j-2} A[k]$$

Running through the loop, on line 9, $A[j-1]$ is added to *sum*. After this line is executed, the new value of sum becomes

$$sum = \sum_{k=0}^{j-2} A[k] + A[j-1] = \sum_{k=0}^{j-1} A[k]$$

Then, line 10 executes and the value of i is incremented and becomes j. After the line is executed, the program returns to the top of loop, signifying the beginning of the $j+1$th iteration. *sum* remains unchanged since the execution of line 9, and is equal to

$$\sum_{k=0}^{j-1} A[k] = \sum_{k=0}^{i-1} A[k]$$

as desired.

By the principle of simple induction, at the beginning of the jth iteration, $Q(j)$ holds for any $j \in \{1, 2, ..., \texttt{len}(A), \texttt{len}(A)+1\}$.

At the beginning of the $\texttt{len}(A)+1$th iteration, $i = \texttt{len}(A)$, so the loop terminates. The program returns $sum/\texttt{len}(A)$ on line 11, which is exactly the average of the numbers in A.

□

Q2. –Following up on the Q1– We mentioned this in class, but it is good to remind you again of the fact that, proving $\forall j \in \{1, \ldots, len(A)\}, Q(j)$ is equivalent to proving $\forall n \in \mathbb{N}, Q'(n)$ where

$$Q'(n) : 0 \leq n < len(A) \Rightarrow Q(n+1).$$

Explain the equivalence.

Proof. It will be proven that the two statements imply each other, that is,

$$\forall j \in \{1, ..., \texttt{len}(A)\}, Q(j) \iff \forall n \in \mathbb{N}, 0 \leq n < \texttt{len}(A) \implies Q(n+1)$$

To prove the forward direction, suppose that $Q(j)$ holds true for all $j \in \{1, ..., \texttt{len}(A)\}$. Let n be a natural number such that $0 \leq n < \texttt{len}(A)$. It follows that $n \in \{0, 1, ..., \texttt{len}(A) - 1\}$, which implies that $n + 1 \in \{1, ..., \texttt{len}(A)\}$. By the original assumption, $Q(n + 1)$ is true.

Conversely, suppose that for all natural n, if $0 \leq n < \texttt{len}(A)$ then $Q(n + 1)$ is true. Let $j \in \{1, ..., \texttt{len}(A)\}$. Then $j - 1 \in \{0, ..., \texttt{len}(A) - 1\}$. It is clear that $j - 1$ is a natural number and that $0 \leq j < \texttt{len}(A)$. By the assumption in the beginning, $Q(j)$ is true.

Thus both statements are equivalent to each other.

□

Q3. Solve questions 6 to 10 on pages 64 to 66 of the textbook. There is a typo in Q8 line 12 (it should be `c = 1` instead of `c == 1`). Proofs must follow the lecture's format and level of clarity. Only **one** of the five questions will be selected for marking. A serious attempt of all the mentioned questions is a necessary condition for receiving more than 0 points.

Question 6.

1. Give a loop invariant that characterizes the values of a and y.

   Proof. Define the loop invariant to be

   $$Inv(x, y) : a = x + \frac{y(y+1)}{2} - 55$$

   It will be proven that at the beginning of the ith iteration of the loop, $Inv(x, y)$ is true.

   Base Case. At the beginning of the first iteration, $a = x$ and $y = 10$. Thus

   $$x + \frac{y(y+1)}{2} - 55 = x - \frac{10(11)}{2} - 55 = x - 55 + 55 = x = a$$

   The base case has been proven to be true.

   Induction Hypothesis. Now suppose that for some positive natural j, $Inv(a, y)$ holds true at the beginning of the jth iteration. y is initially 10 and is decremmented at the end of every iteration, so at the beginning of the jth iteration, $y = 10 - (j-1) = 11 - j$. Then the loop invariant states that

   $$a = x + \frac{(11-j)(12-j)}{2} - 55$$

   Induction Step. If $a \leq 0$, the loop terminates and never reaches the beginning of the $j+1$th iteration. Otherwise, on line 6 in the jth iteration, a is subtracted by $y = 11 - j$. Using the induction hypothesis, the value of a is now

   $$a = x + \frac{(11-j)(12-j)}{2} - 55 - (11-j) = x + \left(\frac{(11-j)(12-j)}{2} - (11-j)\right)$$

   $$= x + \frac{(11-j)(12-j-2)}{2} - 55 = x + \frac{(10-j)(11-j)}{2} - 55$$

   Finally, on line 7, the value of y is decremented by 1 and becomes $y = 10 - j$. The program then arrives at the beginning of the $j+1$th iteration. Indeed, it is true that

   $$a = x + \frac{(10-j)(11-j)}{2} - 55 = x + \frac{y(y+1)}{2} - 55$$

   Thus $Inv(a, y)$ is true at the beginning of the $j+1$th iteration. Verifying the validity of the loop invariant.

   □

2. Show that sometimes this code fails to terminate

Proof. Let $x = 56$. It wil be shown using simple induction that at the beginning of every iteration, $a > 0$, which implies that the loop can never terminate.

Consider the beginning of an arbitrary ith iteration. It is known that $y = 11 - j$. From the loop invariant found in part (a),

$$a = x + \frac{y(y+1)}{2} - 55 = 56 + \frac{(11-j)(12-j)}{2} - 55 = 1 + \frac{(11-j)(12-j)}{2}$$

Note that $\frac{(11-j)(12-j)}{2} < 0$ if and only if $11 < j$ and $j < 12$ at the same time, which is impossible for $j \in \mathbb{N}$. Thus $\frac{(11-j)(12-j)}{2} \geq 0$ so

$$a = 1 + \frac{(11-j)(12-j)}{2} \geq 1 > 0$$

Therefore, $a > 0$ at the beginning of all iterations, which means that the loop never terminates.

□

Question 7. State the pre- and post-conditions that the algorithms must satisfy, then prove that each algorithm is correct.

(a) *Proof.* Pre-condition: a is a non-zero number and b is a natural number.

Post-condition: Returns the value of a^b.

The correctness of the program will be proved using complete induction on b. For the entirety of the proof, a is fixed to be an arbitrary non-zero number.

Base Case. When $b = 0$, the method returns 1, which is exactly a^b.

Induction Hypothesis. Suppose that the program returns the correct value of a^b for all $b \leq n$, for some $n \in \mathbb{N}$.

Induction Step. When $b = n + 1$, $b \geq 1$, so line 3 is not reached. Consider two possible cases:

If b is even, line 5 and 6 will be executed. A new variable x is initialized to be equal to `exp_rec(a, b / 2)`. Since $\frac{b}{2}$ is natural and less than or equal to n, By the induction hypthesis, the value of x is $a^{\frac{b}{2}}$. The program returns `x * x`, which is equal to $a^{\frac{b}{2}} \cdot a^{\frac{b}{2}} = a^b$ on line 6.

If b is odd, line 8 and 9 will be executed. By a similar argument as the case above, a variable x is initialized to the value $a^{\frac{b-1}{2}}$ and `x * x * a` is returned, which has the value of $a^{\frac{b-1}{2}} \cdot a^{\frac{b-1}{2}} \cdot a = a^b$

Therefore, it can be concluded that the program satisfies the postconditions for all natural b.

□

(b) *Proof.* Let a be a non-zero number and b be a natural number.

Define the loop variant to be `exp`, which is initialized to be the natural number b. Every iteration, a floor division of 2 is applied to it, so it is always natural and decreasing. Thus the program will eventually terminate.

For a positive natural i, define the loop invariant

$$P(i) : \texttt{mult}^{\texttt{exp}} \cdot \texttt{ans} = a^b$$

It will be shown using induction on i that for $i \in \{1, 2, ..., \lfloor \log_2(b) \rfloor + 1\}$, $P(i)$ is true at the beginning of the ith iteration.

Base Case. Let $i = 1$. At the start of the very first iteration, `mult = a`, `ans = 1`, and `exp = b`, so it follows that

$$\texttt{mult}^{\texttt{exp}} \cdot \text{ans} = a^b \cdot 1 = a^b$$

Thus the base case holds.

Induction Hypothesis. Let $j \in \{1, 2, ..., \lfloor \log_2(b) \rfloor\}$. Suppose that at the beginning of the jth iteration, $P(j)$ is true. That is,

$$\texttt{mult}^{\texttt{exp}} \cdot \texttt{ans} = a^b$$

Induction Step. Let $\texttt{mult}_0, \texttt{exp}_0, \texttt{ans}_0$ be the value of $\texttt{mult}, \texttt{exp}, \texttt{ans}$ at the beginning of the jth iteration and let $\texttt{mult}_1, \texttt{exp}_1, \texttt{ans}_1$ denote the value of the variables at the beginning of the $j + 1$th iteration.

Now, two cases will be considered for the analysis of the loop execution.

If `exp` is odd, then line 7 will run and multiply `ans` by `mult`, which implies that $\texttt{ans}_1 = \texttt{ans}_0 \cdot \texttt{mult}_0$.

Line 8 will run and will multiply `mult` by itself, so $\texttt{mult}_1 = \texttt{mult}_0^2$.

As well, `exp` can be rewritten as $2n + 1$, for some natural n, so line 9 will result in $\texttt{exp}_1 = \texttt{exp}_0 // 2 = \frac{\texttt{exp}_0 - 1}{2}$

After line 9, the current iteration ends and the $j + 1$th iteration begins. Using algebriac manipulation and the induction hypothesis, $P(j + 1)$ can be verified to be true:

$$\texttt{mult}_1^{\texttt{exp}_1} \cdot \texttt{ans}_1 = (\texttt{mult}_0^2)^{\frac{\texttt{exp}_0 - 1}{2}} \cdot (\texttt{ans}_0 \cdot \texttt{mult}_0) = (\texttt{mult}_0)^{\texttt{exp}_0 - 1}(\texttt{mult}_0 \cdot \texttt{ans}_0)$$
$$= \texttt{mult}_0^{\texttt{exp}_0} \cdot \texttt{ans}_0 = a^b$$

Otherwise if `exp` is even, then line 7 will not run, so $\texttt{ans}_1 = \texttt{ans}_0$.

Finally, from line 8, it is seen that $\texttt{mult}_1 = \texttt{mult}_0^2$

`exp` can be written as $2k$ for some natural k, so line 9 produces $\texttt{exp}_1 = \texttt{exp}_0 // 2 = \frac{\texttt{exp}_0}{2}$.

The execution of line 9 denotes the end of the jth iteration and the start of the $j + 1$th iteraton. Similar to the previous case, it is seen that

$$\texttt{mult}_1^{\texttt{exp}_1} \cdot \texttt{ans}_1 = (\texttt{mult}_0^2)^{\frac{\texttt{exp}_0}{2}} \cdot (\texttt{ans}_0) = \texttt{mult}_0^{\texttt{exp}_0} \cdot \texttt{ans}_0 = a^b$$

Thus, in either case, $P(j+1)$ has been shown to hold.

By the principle of simple induction, it has been shown that $P(\lfloor \log_2(b) \rfloor + 1)$ is true at the beginning of iteration number $\lfloor \log_2(b) \rfloor + 1$. Since `exp` was initialized as b and has been halved $\lfloor \log_2(b) \rfloor + 1$ times, the value of `exp` at the beginning of the iteration is 0, which means that the loop terminates. Using the fact that $P(\lfloor \log_2(b) \rfloor + 1)$ holds,

$$\texttt{mult}^{\texttt{exp}} \cdot \texttt{ans} = \texttt{mult}^0 \cdot \texttt{ans} = \texttt{ans} = a^b$$

The program returns `ans` on line 10, which satisfies its postconditions. □

Question 8. Prove that `majority()` is correct.

Proof. Define the loop variant to be $\texttt{len}(A) - i$. Notice that it is a natural number and decreases every iteration. Thus the program will terminate eventually.

Before continuing to prove correctness, a new definition will be defined to make the proof more concise and clear to read: A list of integers that has more than half of its entries equal to some integer x is called a **majority list** for x.

Let A be a list with more than half of its entries equal to each other, and let x be that value. As well, let n_i represent the absolute difference between the number of occurences of x and the other values in $A[0:i]$. For a positive natural i, define the loop invariant

$$P(i) : \text{if } A[0:i] \text{ is a majority list for } x, \text{ then } m = x \text{ and } c \geq n_i. \text{ As well,}$$

$$\text{if } A[0:i] \text{ is a not a majority list for } x, \text{ then } c \leq n_i \text{ or } m = x$$

It will be shown using complete induction that for $i \in \{1, 2, ..., \texttt{len}(A)\}$, $P(i)$ is true at the beginning of the ith loop iteration.

Base Case. Consider two base cases.

Let $i = 1$. Suppose that $A[0:1]$ is a majority list for x. Then it must be that $A[0] = x$ and $n_1 = 1$. Upon entering the loop for the first time, $m = A[0]$ and $c = 1 \geq n_1$, which satisfies the conclusion of the first implication. Secondly, sppose that $A[0:1]$ is not a majority list for x. Then $A[0] \neq x$ and $n_1 = 1$. The first time entering the loop, $c = 1 \leq n_1$, which is sufficient to prove the second statement to be true.

Thus at the beginning of the 1st iteration, $P(1)$ is true.

Induction Hypothesis. Let $i = k$, for some $k \in \{1, 2, ..., \texttt{len}(A) - 1\}$. Suppose that at the start of the kth iteration, $P(k)$ is true. The end goal is to show that $P(k+1)$ is true at the beginning of the $k+1$th iteration.

Induction Step. Let m_i, c_i denote the values of m and c at the start of the ith iteration. To prove the first implication, suppose that $A[0:k+1]$ is a majority list for x. Consider the following two cases:

1. $A[0:k]$ is a majority list for x:

   By the induction hypothesis, $P(k)$ is true at the start of the kth iteration. That is, $m_k = x$ and $c_k \geq n_i$. Executing the kth iteration, it is clear that $c_k \neq 0$, as $c_k \geq n_i > 0$. Thus lines 11 and 12 are skipped. Then, if $A[k] = m_k = x$, by line 14, $c_{k+1} = c_k + 1 \geq n_k + 1 = n_{k+1}$. Also, $x = m_k = m_{k+1}$. Therefore the first implication holds.

2. $A[0:k]$ is not a majority list for x:

   The only way for $A[0:k]$ to not be a majority list for x at the same time as $A[0:k+1]$ is if $A[k] = x$ and $A[0:k]$ has half its elements equal to x. Since $n_k = 0$, by the induction hypothesis, $c_k = 0$ or $m_k = x$. Executing the kth iteration, if $c_k = 0$, running lines 11 and 12 result in $m_{k+1} = A[k] = x$ and $c_{k+1} = 1$. Otherwise, if $c_k \neq 0$, then $m_k = x$. $A[k] = x$, so the condition on line 13 is satisfied, running line 14, so $c_{k+1} = c_k + 1 \geq n_k + 1 = n_{k+1}$. m remains unchanged so $x = m_k = m_{k+1}$. In either case, the first implication is still shown to be true.

Therefore, the first implication holds true.

Now, proving the second implication, suppose that $A[0:k+1]$ is not a majority list for x. Again, consider cases.

1. $A[0:k]$ is not a majority list for x:

   By the induction hypothesis, $c_k \leq n_k$ or $m_k = x$. Consider what happens if $A[k] = x$. Executing the kth iteration, if $c_k = 0$, lines 11 results in $m_{k+1} = A[k] = x$. If not, then the value of m is unchanged so $m_{k+1} = m_k$. Next, if the condition on line 13 is true, then $x = A[k] = m_k = m_{k+1}$. Finally, if line 16 runs, it means that $m_k = x \neq A[k]$, thus $c_k \leq n_k$ must be true. Then $c_{k+1} = c_k - 1 \leq n_k - 1 = n_{k+1}$. In all situations, either $m_{k+1} = x$ or $c_{k+1} \leq n_{k+1}$.

   Now, consider what happens when $A[k] \neq x$. Again, executing the kth iteration, if $c_k = 0$, line 12 results in $c_{k+1} = 1$, but notice that since $A[0:k]$ is not a majority list, appending a value not equal to x will increase the value of n_{k+1}. Particularly, $n_{k+1} = n_k + 1$. Then $c_{k+1} \leq n_k + 1 = n_{k+1}$. If $c_k \neq 0$, then similar to the previous case, $m_{k+1} = m_k$. If $m_k = A[k] \neq x$, it must be true that $c_k \leq n_k$. Then, line 14 will add 1 to c_k, so $c_{k+1} = c_k + 1 < n_k + 1 = n_{k+1}$. Otherwise, if $m_k \neq A[k]$, then m_k can either be equal or not equal to x. If $m_k = x$, the conclusion follows immediately. If not, $c_k \leq n_k$ and from line 16, c_k is decremented so $c_{k+1} = c_k - 1 \leq n_k - 1 \leq n_{k+1}$.

   Therefore the second implication holds true at the beginning of the $k+1$th iteration.

2. $A[0:k]$ is a majority list for x.

   The only situation where this can be true is if $n_k = 1$ and $A[k] \neq x$. If this condition is not true, then $A[0:k+1]$ is a majority list for x. Continuing on, by the induction hypothesis, $c_k \geq n_k = 1$ and $m_k = x$. Iterating through the kth iteration, notice that c_k is non-zero. Thus lines 11 and 12 are skipped. The value of m_k remains unchanged, so $m_{k+1} = m_k = x$, which makes the second implication true.

Thus it has been shown that both the implications are true, so $P(k+1)$ is true at the beginning of the $k+1$th iteration.

By the principle of simple induction, for all $i \in \{1, ..., \texttt{len}(A)\}$, $P(i)$ holds true at the beginning of the ith iteration. When $i = \texttt{len}(A)$, the loop condition is false so the loop terminates. Notice that $A[0:\texttt{len}(A)] = A$ is a majority list for x, so by $P(i)$, the program correctly returns $m = x$, satisfying the postconditions of the program.

□

Question 9. Studying bubblesort.

(a) State and prove an invariant for the inner loop

Proof. Let L be a list of numbers. Define the inner loop invariant to be

$$Q(n) : \text{For all } j \in \mathbb{N}, j < i,\ L[j] \leq L[i]$$

Let $k \in \{0, 1, ..., \texttt{len}(L) - 1\}$. It will be proven using simple induction that for all $n \in \{1, 2, ..., \texttt{len}(L) - k + 1\}$, at the start of the nth iteration, $Q(n)$ is true.

Base Case. Let $n = 1$. At the beginning of the first iteration, $i = 0$. It follows that $Q(1)$ is vacuously true because there is no natural number $j < 0$.

Induction Hypothesis. Suppose that for some $n \in \{1, 2, ..., \texttt{len}(L) - k\}$, at the beginning of the nth iteration, $Q(n)$ is true. The loop condition is always true for this n, so the loop body will be entered. It will be shown that at the start of the $n + 1$th iteration, $Q(n + 1)$ is true.

Induction Step. Notice that at the end of every iteration, i is incremented once. Thus $i = n - 1$ at the beginning of the nth iteration. Iterating through the loop, if $L[n-1] > L[n]$, their positions in the list will be swapped. This guarantees that $L[n-1] \leq L[n]$ after line 11. Next, i is incremented by 1, so its new value is n. This marks the end of the nth iteration and the start of the $n + 1$th iteration.

By the induction hypothesis, for all natural $j < n - 1$, $L[j] \leq L[n - 1] \leq L[n]$ if line 11 did not run, or $L[j] < L[n]$ if it did. Regardless, $L[j] \leq L[n]$. Additionally, $L[n - 1] \leq L[n]$ is true based on the previous iteration of the loop. It can be concluded that for $j < n = i$, $L[j] \leq L[n] = L[i]$, so $Q(n + 1)$ is true.

By the principle of simple induction, for all $n \in \{1, 2, ..., \texttt{len}(L) - k + 1\}$, at the start of the nth iteration, $Q(n)$ is true.

□

(b) State and prove an invariant for the outer loop

Proof. Define the outer loop invariant to be

$$P(m) : \text{The last } m \text{ elements of } L \text{ are sorted.}$$

In other words, it is true that $L[\texttt{len}(L) - m] \leq L[\texttt{len}(L) - m + 1] \leq \cdots \leq L[\texttt{len}(L) - 1]$. It will be proven using simple induction that for $m \in \{1, 2, ..., \texttt{len}(L) + 1\}$, $P(m)$ is true.

Base Case. Let $m = 1$. When the loop is first entered, the last element of L is obviously sorted. Thus $P(1)$ holds true.

Induction Hypothesis. Suppose that for some $m \in \{1, ..., \texttt{len}(L)\}$, $P(m)$ is true at the beginning of the mth iteration. Notice that k is initialized to be 0 and is incremented every outer loop iteration. Thus $k = m - 1 < \texttt{len}(L)$, so the loop body is entered.

Induction Step. Running the mth iteration, i is initialized to be 0 and the inner loop runs. By the previous part, $Q(\texttt{len}(L) - m + 2)$ is true at the beginning of the $(\texttt{len}(L) - m + 2)$th iteration. Recall that at the beginning of this iteration, $i = \texttt{len}(L) - m + 1 = \texttt{len}(L) - k$, so the loop terminates. Thus before line 13, $L[\texttt{len}(L) - m] \leq L[\texttt{len}(L) - m + 1]$, as $\texttt{len}(L) - m < i$. After line 13, the $m + 1$th iteration begins.

By the induction hypothesis, there is a typo lol. □

(c) Prove that `bubblesort` is correct, according to its specifications

Proof. Let the inner loop variant be $\texttt{len}(L) - k - i$. This value is natural and decreasing, so the inner loop will terminate. Similarly, let the outer loop variant be $\texttt{len}(L) - k$. It is easy to see that the same applies to this value as well. Thus the program will eventually terminate.

From the previous part, it is true that $P(\texttt{len}(L))$ holds at the beginning of the $\texttt{len}(L)$th iteration. At this iteration, $k = \texttt{len}(L) - 1$, so the loop condition is true and the loop body enters. When this iteration reaches the inner loop, notice that $i = 0 - \texttt{len}(L) - k - 1$, so the inner loop does not run. Then k is incremented and the loop returns to the top of the loop, where the loop condition fails and terminates the loop. Since L remained unchanged throughout the iteration, $P(\texttt{len}(L))$ is still true, so the last $\texttt{len}(L)$ elements of the list are sorted, which is the entire list. Thus the program satisfies its post conditions. □

Question 10. Consider the following generalization of the `min` function.

(a) Prove that this algorithm is correct

Proof. This program, given that A is a non-empty list of numbers and k is a positive natural less than or equal to $\texttt{len}(A)$, returns the kth smallest element. It will be shown using complete induction on $\texttt{len}(A)$ that this program is correct.

Base Case. Let A be a list of numbers such that $\texttt{len}(A) = 1$. Then $k = 1$. Taking $A[0]$ as the pivot, line 4 will partition an empty list, so $\texttt{len}(L) = 0$. Then the condition on line 5 is true, so the program correctly returns the pivot as the smallest element on line 6.

Induction Hypothesis. Suppose that for any list B such that $\texttt{len}(B) \leq n$, `extract(B, k)` correctly returns the kth smallest element.

Induction Step. Let A be a list with $\texttt{len}(A) = n + 1$. Running through the program, choosing $A[0]$ as the pivot, line 4 separates A into two lists L, G, where L contains all elements in A such that $A < \texttt{pivot}$ and G contains all elements in A such that $A \geq \texttt{pivot}$. Continuing on to line 5, if $\texttt{len}(L) = k - 1$, then L contains $k - 1$ elements that are smaller than the pivot, so the pivot is the kth smallest element, which the program returns on line 6.

Otherwise, notice that $\texttt{len}(L), \texttt{len}(G) < n + 1 \implies \texttt{len}(L), \texttt{len}(G) \leq n$.

If `len`$(L) \geq k$, the kth smallest element of A is contained in L, and is in fact equal to the kth smallest element of L. The program returns `extract(L, k)`, which by the induction hypothesis, returns the kth smallest element of L, which is equal to the kth smallest element of A.

If neither of the cases above are true, then it must be that the kth smallest element of A is inside of G, and is equal to the $(k - \texttt{len}(A) - 1)$th smallest element of G. Indeed, the program returns `extract(G, k - len(L) - 1)`, which by the induction hypothesis returns the $(k - \texttt{len}(A) - 1)$th smallest element of G, which is the kth smallest element of A.

Thus the program returns the correct output for lists of size $n + 1$, and by the principle of complete induction, the program is correct.

□

(b) Analyze the worse-case running time of this algorithm.

The worst-case for this algorithm is $\mathcal{O}(nk)$, where n is the size of the list and k is the number so that the list is trying to find the kth smallest element. This happens exactly when the original list is sorted in either ascending or descending order, which causes `partition` to return one empty list and a list of $n - 1$ elements, if the original list contained n elements. This will also happen in each recursive case, as the list is sorted. The recursion goes up to depth k, with each recursive call using `partition`, which is $\mathcal{O}(n)$. Thus the worst-case runtime of `extract` is $\mathcal{O}(nk)$.

Q4. Solve questions 6,7,10,12, and 14 on pages 46 to 48 of the textbook. Only **two** of the five questions will be selected for marking. A serious attempt of all the mentioned questions is a necessary condition for receiving more than 0 points.

Question 6. Let $T(n)$ be the number of binary strings of length n where every 1 is immediately preceded by a 0.

(a) Develop a recurrence for $T(n)$

$T(n) = T(n-1) + T(n-2)$.

(b) Find a closed form for $T(n)$

$T(n) = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^{n+1} - \left(\frac{1-\sqrt{5}}{2} \right)^{n+1} \right)$

(c) Prove that the closed form is correct using induction

Question 7. Let $T(n)$ denote the number of distinct full binary trees with n nodes. Give a recurrence for $T(n)$. Then use induction to prove that $T(n) \geq \frac{1}{n} 2^{(n-1)/2}$.

Proof. Full binary trees can be defined recursively in the following way:

1. A leaf node is a full binary tree
2. If T_1, T_2 are full binary trees, then the binary tree obtained from attaching T_1, T_2 to the root node is a full binary tree.

This motivates the recursive definition for $T(n)$ to be

$$T(n) = 1 + 2 \sum_{i=1}^{\frac{n-1}{2}} T(n-i-1)$$

Now, it will be shown using complete induction that $T(n) \geq \frac{1}{n} 2^{(n-1)/2}$ for all $n \in \mathbb{N}^+$.

Base Case. Let $n = 1$. Then $T(n) = 1$ and $\frac{1}{n} 2^{(n-1)/2} = 1$, thus the base case holds.

Induction Hypothesis. Let $k \in \mathbb{N}^+$. Suppose that for all $i \leq k$, $T(i) \geq \frac{1}{i} 2^{(i-1)/2}$. The goal is to show that $T(k+1) \geq \frac{1}{k+1} 2^{(k)/2}$.

Induction Step. Using the recursive definition for $T(n)$, as well as the induction hypothesis,

$$T(k+1) = 1 + 2 \sum_{i=1}^{\frac{n-1}{2}} T(n-i-1) \geq 1 + 2 \sum_{i=1}^{\frac{n-1}{2}} \frac{1}{n-i-1} 2^{(n-i-2)/2}$$

□

Question 10. Let $H(n)$ denote the number of binary strings of length n that have no odd length blocks of 1's.

Develop a recursive definition for $H(n)$, and justify why it is correct. Then find a closed form for H using repeated substitution.

Question 12. Analyze the worse-case runtime of `fast_rec_mult`.

Question 14. Recall the recurrence for the worst-case runtime of quicksort:

$$T(n) = \begin{cases} c, & \text{if } n \leq 1; \\ T(|L|) + T(|G|) + dn, & \text{if } n > 1; \end{cases}$$

where L and G are the partitions of the list.

1. Suppose the lists are always evenly split; that is, $|L| = |G| = \frac{n}{2}$ at each recursive call. Find a tight asymptotic bound on the runtime of quicksort using this assumption.

2. Now suppose that the lists are always very unevenly split: $|L| = n - 2$ and $|G| = 1$ at each recursive call. Find a tight asymptotic bound on the runtime of quicksort using this assumption.