Designing Algorithms

1.

a.

Iterate over each character in the sequence from the beginning to end, replacing each A, T, G, and C with its T, A, C, and G, respectively.

b.

No!

c.

```
def complement(sequence):
    """ (str) -> str

    Return the complement of sequence.

    >>> complement('AATTGCCGT')
    'TTAACGGCA'
    """

    complement_dict = {'A': 'T', 'T': 'A', 'C': 'G', 'G': 'C'}
    sequence_complement = ''

    for char in sequence:
        sequence_complement = sequence_complement + complement_dict[char]

    return sequence_complement
```

2.

a.

```
index = 0
smallest = L[0]

for i in range(1, len(L)):
    if L[i] < smallest:
        index = i
        smallest = L[i]
```

b.

```
def min_index(L):
    """ (list) -> (object, int)
```

```
    Return a tuple containing the smallest item from L and its index.

    >>> min_index([4, 3, 2, 4, 3, 6, 1, 5])
    (1, 6)
    """

    index = 0
    smallest = L[0]

    for i in range(1, len(L)):
        if L[i] < smallest:
            index = i
            smallest = L[i]

    return (smallest, index)
```

c.

```
def min_or_max_index(L, flag):
    """ (list, bool) -> tuple of (object, int)

    Return the minimum or maximum item and its index from L, depending on
    whether flag is True or False.

    >>> min_or_max_index([4, 3, 2, 4, 3, 6, 1, 5], True)
    (1, 6)
    >>> min_or_max_index([4, 3, 2, 4, 3, 6, 1, 5], False)
    (6, 5)
    """

    index = 0
    current_value = L[0]

    if flag:
        for i in range(1, len(L)):
            if L[i] < current_value:
                index = i
                current_value = L[i]
    else:
        for i in range(1, len(L)):
            if L[i] > current_value:
                index = i
                current_value = L[i]

    return (current_value, index)
```

3.

a.

- Read the description line.
- Keep reading the comment lines until we read the first piece of data.
- Add the first piece of data to an empty list.
- Read the remaining lines one at a time, appending the data to the list.

b.

```
def hopedale_average(filename):
    """ (str) -> float

    Return the average number of pelts produced per year for the data in
Hopedale
    file named filename.
    """

    with open(filename, 'r') as hopedale_file:

        # Read the description line.
        hopedale_file.readline()

        # Keep reading comment lines until we read the first piece of data.
        data = hopedale_file.readline().strip()
        while data.startswith('#'):
            data = hopedale_file.readline().strip()

        # Now we have the first piece of data append it to an empty list.
        pelts_list = []
        pelts_list.append(int(data))

        # Read the rest of the data.
        for data in hopedale_file:
            pelts_list.append(int(data.strip()))

    return sum(pelts_list) / len(pelts_list)
```

4.

```
# Two items; smallest first.
>>> find_two_smallest([1, 2])
(0, 1)

# Two items; smallest second.
>>> find_two_smallest([3, 2])
(1, 0)

# Two items; same values.
>>> find_two_smallest([3, 3])
(0, 1)

# Three items items; 2nd smallest is duplicated.
>>> find_two_smallest([3, 1, 3])
(1, 0)

# Multiple items: smallest at beginning; 2nd smallest at middle.
>>> find_two_smallest([1, 4, 2, 3, 4])
(0, 2)

# Multiple items: smallest at middle; 2nd smallest at end.
>>> find_two_smallest([4, 3, 1, 5, 6, 2])
(5, 3)
```

```
# Multiple items: smallest at end; 2nd smallest at beginning.
>>> find_two_smallest([-2, 4, 3, 2, 5, 6, -1])
(3, 4)
```

5.

If passed a list of length one, it should return a tuple containing the index of the smallest. If passed a list of length zero, it should return an empty tuple.

```
Return a tuple of the indices of the two smallest values in list L.
If there is only one item in L or zero items in L, return a tuple
containing the index of that one item or an empty tuple, respectively.
```

6.

```
def dutch_flag(color_list):
    """ (list of str) -> list of str

    Return color_list rearranged so that 'red' strings come first, 'green'
second,
    and 'blue' third.

    >>> color_list = ['red', 'green', 'blue', 'red', 'red', 'blue', 'red',
'green']
    >>> dutch_flag(['red', 'green', 'blue', 'red', 'red', 'blue', 'red',
'green'])
    >>> color_list
    ['red', 'red', 'red', 'red', 'green', 'green', 'blue', 'blue']
    """

    i = 0

    # The start of the green section.
    start_green = 0

    # The index of the first unexamined color.
    start_unknown = 0

    # The index of the last unexamined color.
    end_unknown = len(color_list) - 1

    print(color_list)
    print('start')

    while start_unknown <= end_unknown:

        # If it is red, swap it with the item to the right of the red
section.
        if color_list[start_unknown] == 'red':
            color_list[start_green], color_list[start_unknown] \
            = color_list[start_unknown], color_list[start_green]
            start_green += 1
            start_unknown += 1
```

```
        # If it is green, leave it where it is.
        elif color_list[start_unknown] == 'green':
            start_unknown += 1

        # If it is blue, swap it with the item to the left of the blue
section.
        else:
            color_list[start_unknown], color_list[end_unknown] \
            = color_list[end_unknown], color_list[start_unknown]
            end_unknown -= 1
```