Lab 12: Power Design
Margaret Binns

1) By Hand

27, 33, 41, 22, 38, 45, 29, 36, 49, 24, 31, 47, 25, 43
(27+33+41+22+38+45+29+36+49+24)/10 = 34.4

27, 33, 41, 22, 38, 45, 29, 36, 49, 24, 31, 47, 25, 43
(33+41+22+38+45+29+36+49+24+31)/10 = 34.8

27, 33, 41, 22, 38, 45, 29, 36, 49, 24, 31, 47, 25, 43
(41+22+38+45+29+36+49+24+31+47)/10 = 36.2

27, 33, 41, 22, 38, 45, 29, 36, 49, 24, 31, 47, 25, 43
(22+38+45+29+36+49+24+31+47+25)/10 = 34.6

27, 33, 41, 22, 38, 45, 29, 36, 49, 24, 31, 47, 25, 43
(38+45+29+36+49+24+31+47+25+43)/10 = 36.7

Fifth interval (36.7) is the highest.

2) Approach

I'll set up the list or array with the 14 values in order of when they were recorded (chronological). A while loop will then go through five cycles, each for a different ten-number interval. A nested for loop will go through the ten numbers in each interval, add them up, then divide that number by ten (get the average). A new variable will be overwritten if the current average is higher than it.

3) Pseudocode

```
measurementList = [27, 33, 41, 22, 38, 45, 29, 36, 49, 24, 31, 47, 25, 43]
highestAverage = 0
startIndex = 0
counter = 0

WHILE counter <  5:
        currentAverage = 0
        addedNums = 0

        FOR i in range(startIndex, startIndex + 9):
                number = measurementList[i]
                addedNums = addedNums + number

        currentAverage = addedNums/10

        IF currentAverage > highestAverage:
                highestAverage = currentAverage

        startIndex = startIndex + 1
        counter = counter + 1
```

4) Microsoft Pilot

```
# Initialize the list with 14 values in chronological order
values = [value1, value2, ..., value14]

# Initialize a variable to store the highest average
highest_average = 0

# Initialize the starting index for the intervals
start_index = 0

# While loop to go through five cycles
while start_index <= 4:
    # Initialize the sum for the current interval
    interval_sum = 0

    # For loop to go through the ten numbers in the current
interval
    for i from start_index to start_index + 9:
        interval_sum = interval_sum + values[i]

    # Calculate the average for the current interval
    current_average = interval_sum / 10
```

```
    # Update the highest average if the current average is higher
    if current_average > highest_average:
        highest_average = current_average

    # Move to the next interval
    start_index = start_index + 1

# Output the highest average
print(highest_average)
```

5) Compare and Contrast

- Provide an analysis as to the pros and cons of the two solutions?

  Well, my pseudocode and the Copilot's pseudocode are almost the same. A con both mine and Copilot's has is that it's very hardcoded and going in to change the iteration length or length of the initial list is not easy. A pro that both have is that it's easy to understand what being done.
- How can your solution be improved based on what Copilot provided?

  One thing I could do differently is to maybe avoid 'clutter' by removing my counter variable and just having the while loop go off of the starting variable instead. Frankly, I like the counter though.
- How can Copilot's solution be improved based on what you know?

  Were I to improve this, I'd have more variables at the beginning to make it easier to adjust either the initial list length or the interval length.
- Does the pseudocode in Step 3 and Step 4 match the algorithm you performed in Step 1?

  Yes, as far as I can tell.

6) Update

```
measurementList = [27, 33, 41, 22, 38, 45, 29, 36, 49, 24, 31, 47, 25, 43]
intervalLength = 10
highestAverage = 0
startIndex = 0
counter = 0

WHILE counter <  (measurementList(len) - (intervalLength-1)):
        currentAverage = 0
        addedNums = 0

        FOR i in range(startIndex, startIndex + (intervalLength-1)):
                number = measurementList[i]
                addedNums = addedNums + number

        currentAverage = addedNums/intervalLength

        IF currentAverage > highestAverage:
                highestAverage = currentAverage

        startIndex = startIndex + 1
        counter = counter + 1
```

7) Trace

Create a program trace of the algorithm which computes the data below with a sub-list of size 4.

41	45	47	32	49	40	32

Loop#	highestAverage	startIndex	counter	currentAverage	number	addedNums
0	0	0	0	0	--	0
1					41	41
					45	86
					47	133
	41.25	1	1	41.25	32	165
2					45	45
					47	92
					32	124
	43.25	2	2	43.25	49	173
3					47	47
					32	79
					49	128
	43.25	3	3	42	40	168
4					32	32
					49	81
					40	121
	43.25	4	4	38.25	32	153

8) Efficiency

As far as I can tell, the nested FOR loop inside the WHILE loop, as well as the fact that the interval length and list length can be changed, results in the efficiency being O(n log n).

```
Step 1 By Hand: 15 minutes

Step 2 Approach: 20 minutes

Step 3 Pseudocode: 40 minutes

Step 4 Copilot: 3 minutes

Step 5 Compare and Contrast: 5 minutes

Step 6 Update: 10 minutes

Step 7 Trace: 30 minutes

Step 8 Efficiency: 6 minutes
```