

Power Design

The purpose of this assignment is to design an algorithm that finds the largest average of a length of numbers in an array that contains many more numbers. The amount of time spend on each step is listed in the heading of that step, but is also an estimate and not an actual measured time.

Step 1: By Hand (20 minutes)

I will compute by hand the largest average in a list of numbers 14 elements long with a sample size of 10. The list of numbers that I will use to compute the highest average by hand is:

[34, 50, 16, 39, 18, 84, 91, 30, 96, 92, 32, 11, 33, 90]

The following is a table showing each average.

List	Average
[34, 50, 16, 39, 18, 84, 91, 30, 96, 92 , 32, 11, 33, 90]	55
[34, 50, 16, 39, 18, 84, 91, 30, 96, 92, 32 , 11, 33, 90]	54.8
[34, 50, 16, 39, 18, 84, 91, 30, 96, 92, 32, 11 , 33, 90]	50.9
[34, 50, 16, 39, 18, 84, 91, 30, 96, 92, 32, 11, 33 , 90]	52.6
[34, 50, 16, 39, 18, 84, 91, 30, 96, 92, 32, 11, 33, 90]	57.7

Looking at this table of averages, the largest average there is 57.7, which means 57.7 is the largest average of 10 numbers in my list of 14 numbers.

Step 2: Approach (15 minutes)

The program will start by adding a set amount of numbers together and storing that in a variable. (Those numbers being the first however many in the array). The program will then divide by the number of numbers added together to get the average, and store that in a new list. Then the program will add the next number in the array of numbers and subtract the first of the added numbers. The program will replace the original value for storing this with the new one. The program follows by dividing and storing the new number in the list at the end. This process repeats until it can't add another number at the end of the array.

The program finishes by finding the greatest number in the list of averages and getting the index. This index represents the index of the starting number in the sequence that contains the largest average. The program can then show you the sub-array that represents the largest average.

Step 3: Pseudocode (30 minutes)

The following is the pseudocode for this algorithm.

```
number_array = INPUT
range = INPUT
end_index = LENGTH_OF(number_array) - range
avg_list = empty list
total = 0

# First average
FOR i = 0; i < range; i++
    total += number_array[i]
avg_list.ADD(total / range)

# Remaining averages
FOR start_index = 1; start_index < end_index; start_index ++
    total -= number_array[start_index - 1]
    total += number_array[start_index + range]
```

```

    avg_list.ADD(total / range)

# Find largest average
largest_avg = avg_list[0]
largest_avg_index = 0
avg_list_len = LENGTH_OF(avg_list)
FOR i = 1; i < avg_list_len; i++
    current_avg = avg_list[i]
    IF current_avg > largest_avg
        largest_avg = current_avg
        largest_avg_index = i

# Report largest average
PRINT largest_avg
# OPTIONAL: follow with code to report the sub-array

```

Step 4: Gemini (1 minute)

The following is the code generated from Gemini when given my description of my algorithm from Step 2.

```

FUNCTION sliding_average(numbers)

# Initialize variables
current_sum = 0
averages = []
num_added = 0

# Loop through numbers until reaching the end
FOR number IN numbers
    # Add number to current sum and increase counter
    current_sum += number
    num_added += 1

```

```

# Check if there are numbers remaining
IF num_added < LENGTH(numbers)
    # Calculate average and store it
    average = current_sum / num_added
    averages.append(average)

    # Update sum and counter for next iteration
    current_sum = current_sum - numbers[0] + numbers[num_added]
    numbers.remove(numbers[0])
ELSE
    # No more numbers to add, break the loop
    BREAK
ENDFOR

# Find the index of the largest average
largest_index = 0
largest_average = averages[0]
FOR i = 1 TO LENGTH(averages)
    IF averages[i] > largest_average
        largest_average = averages[i]
        largest_index = i
    END IF
ENDFOR

# Print the sub-array with the largest average
PRINT "Largest average sub-array: "
FOR i = largest_index TO largest_index + num_added - 1
    PRINT numbers[i]
ENDFOR

END FUNCTION

```

Step 5: Compare and Contrast (10 minutes)

Gemini's version of the program is good. It is probably a quicker way to write the program. However, it's less efficient because it takes slightly more repetitions than mine does. In the long run, the algorithmic efficiency appears to be almost identical however. It also ignores the correct index to find the correct index or sub-array. I will add here that I'm tired and am likely missing things that Gemini is doing well.

Step 6: Update (5 minutes)

Here is my updated pseudocode.^[1]

```
number_array = INPUT
range = INPUT
end_index = LENGTH_OF(number_array) - range
avg_list = empty list
total = 0

# First average
FOR i = 0; i < range; i++
    total += number_array[i]
    #A
avg_list.ADD(total / range)
#B

# Remaining averages
FOR start_index = 1; start_index < end_index; start_index ++
    total -= number_array[start_index - 1]
    total += number_array[start_index + range - 1]
    avg_list.ADD(total / range)
    #C

# Find largest average
```

```

largest_avg = avg_list[0]
largest_avg_index = 0
#D
avg_list_len = LENGTH_OF(avg_list)
FOR i = 1; i < avg_list_len; i++
    current_avg = avg_list[i]
    IF current_avg > largest_avg
        largest_avg = current_avg
        largest_avg_index = i
#E

# Report largest average
PRINT largest_avg
#F

```

Step 7: Trace (30 minutes)

Below is the trace table using the numbers [41, 45, 47, 32, 49, 40, 32] with a sub-array size of 4.

Trace Location	total	avg_list	current_avg	largest_avg
A	41			
A	86			
A	133			
A	165			
B	165	[41.25]		
C	173	[41.25, 43.25]		
C	168	[41.25, 43.25, 42]		
C	153	[41.25, 43.25, 42, 38.25]		

Trace Location	total	avg_list	current_avg	largest_avg
D	153	[41.25, 43.25, 42, 38.25]		41.25
E	153	[41.25, 43.25, 42, 38.25]	43.25	43.25
E	153	[41.25, 43.25, 42, 38.25]	42	43.25
E	153	[41.25, 43.25, 42, 38.25]	38.25	43.25
F	153	[41.25, 43.25, 42, 38.25]	38.25	43.25

Step 8: Efficiency (7 minutes)

Below I use my pseudocode to determine the algorithmic efficiency, adding comments on each relevant line for the efficiency of that line.

```

number_array = INPUT # O(1)
range = INPUT # O(1)
end_index = LENGTH_OF(number_array) - range # O(1)
avg_list = empty list # O(1)
total = 0 # O(1)

# First average
FOR i = 0; i < range; i++ # O(n)
    total += number_array[i] # O(1)
avg_list.ADD(total / range) # O(1)

# Remaining averages
FOR start_index = 1; start_index < end_index; start_index ++ #
O(n)

```

```

    total -= number_array[start_index - 1] # O(1)
    total += number_array[start_index + range - 1] # O(1)
    avg_list.ADD(total / range) # O(1)

# Find largest average
largest_avg = avg_list[0] # O(1)
largest_avg_index = 0 # O(1)
avg_list_len = LENGTH_OF(avg_list) # O(1)
FOR i = 1; i < avg_list_len; i++ # O(n)
    current_avg = avg_list[i] # O(1)
    IF current_avg > largest_avg # O(1)
        largest_avg = current_avg # O(1)
        largest_avg_index = i # O(1)

# Report largest average
PRINT largest_avg # O(1)

```

Every part of the algorithm is $O(1)$ except in each of the loops where the overall efficiency is $O(n)$ for each loop. There are no nested loops. This means that the overall algorithmic efficiency is $O(n)$.

1. This isn't changed much. Gemini didn't contribute in a way that helped me, at least not with my current mental state. I did however add locations for the trace table. ↵