Step 1: By Hand

1	2	3	4	5	6	7	8	9	10	11	12	13	14	= 5.5
1	2	3	4	5	6	7	8	9	10	11	12	13	14	= 6.5
1	2	3	4	5	6	7	8	9	10	11	12	13	14	= 7.5
1	2	3	4	5	6	7	8	9	10	11	12	13	14	= 8.5
1	2	3	4	5	6	7	8	9	10	11	12	13	14	= **9.5**

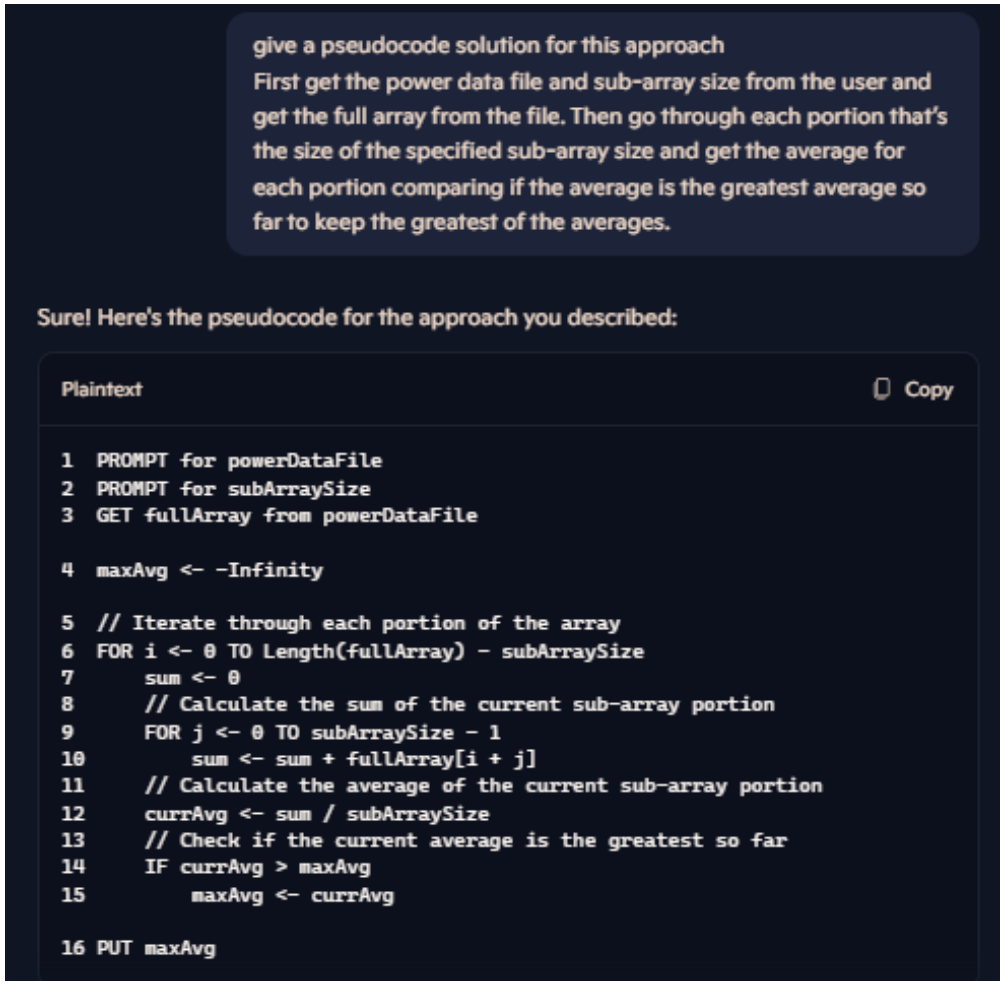
Step 2: Approach

First get the power data file and sub-array size from the user and get the full array from the file. Then go through each portion that's the size of the specified sub-array size and get the average for each portion comparing if the average is the greatest average so far to keep the greatest of the averages.

Step 3: Pseudocode

```
1  PROMPT for powerDataFile
2  PROMPT for subArraySize
3  GET fullAraray from powerDataFile
4
5  subArray ← []
6  sum ← 0
7  // Starts the subarray as the first range to get average for
8  FOR i ← 0 … subArraySize – 1
9     subArray[i] ← fullArray[i]
10    sum ← sum + subArray[i]
11 prevAvg ← sum / subArraySize
12 maxAvg ← prevAvg
13
14 FOR i ← 0 … Length(fullArray) – 1 – subArraySize
15    // gets the avg of the new subArray based off avg of previous subArray
16    currAvg ← ( prevAvg x subArraySize – subArray[i] + fullArray[subArraySize + i] ) / subArraySize
17    IF currAvg > maxAvg
18       maxAvg ← currAvg
19    prevAvg ← currAvg
20    // replaces oldest item in subArray with next item in fullArray after subArray to move it through fullArray
21    subArray[i] ← fullArray[subArraySize + i]
22
23 PUT maxAvg
```

Step 4: Microsoft Copilot



Step 5: Compare and Contrast

- Provide an analysis as to the pros and cons of the two solutions?
  Mine has a better big O efficiency, but the calculations in copilots are simpler.
- How can your solution be improved based on what Copilot provided?
  I can make my calculations for the averages more simple and break them down.
- How can Copilot's solution be improved based on what you know?
  It can split things up or even just get rid of one for loop to give a better efficiency.
- Does the pseudocode in Step 3 and Step 4 match the algorithm you performed in Step 1?
  Yes they do.

Step 6: Update

```
PROMPT for powerDataFile
PROMPT for subArraySize
GET fullAraray from powerDataFile

subArray ← []
currSum ← 0
// Sets up the subarray and gets the average for the first subarray
FOR i ← 0 … subArraySize – 1
   subArray[i] ← fullArray[i] // copies the first part of fullArray to subArray, specified by subArraySize
   currSum ← currSum + subArray[i]
maxAvg ← currSum / subArraySize

FOR i ← 0 … Length(fullArray) – 1 – subArraySize // for each index in fullArray - number of indexes in subArray
   prevSum ← currSum
   currSum ← prevSum – subArray[i] + fullArray[subArraySize + i] // remove oldest num, add next num in fullArray
   currAvg ← currSum / subArraySize
   IF currAvg > maxAvg // set the maxAvg to the avg just calculated if it’s the highest so far
      maxAvg ← currAvg
   // replaces oldest num in subArray with next num in fullArray after the subArray to move it through fullArray
   subArray[i] ← fullArray[subArraySize + i]

PUT maxAvg
```

Step 7: Trace

line number	subArraySize	fullArray	subArray	i	prevSum	currSum	currAvg	maxAvg
8	4	41, 45, 47, 32, 49, 40, 32		0	/	0	/	/
9	4	41, 45, 47, 32, 49, 40, 32	41	0	/	0	/	/
10	4	41, 45, 47, 32, 49, 40, 32	41	0	/	41	/	/
8	4	41, 45, 47, 32, 49, 40, 32	41	1	/	41	/	/
9	4	41, 45, 47, 32, 49, 40, 32	41, 45	1	/	41	/	/
10	4	41, 45, 47, 32, 49, 40, 32	41, 45	1	/	86	/	/
8	4	41, 45, 47, 32, 49, 40, 32	41, 45	2	/	86	/	/
9	4	41, 45, 47, 32, 49, 40, 32	41, 45, 47	2	/	86	/	/
10	4	41, 45, 47, 32, 49, 40, 32	41, 45, 47	2	/	133	/	/
8	4	41, 45, 47, 32, 49, 40, 32	41, 45, 47	3	/	133	/	/
9	4	41, 45, 47, 32, 49, 40, 32	41, 45, 47, 32	3	/	133	/	/
10	4	41, 45, 47, 32, 49, 40, 32	41, 45, 47, 32	3	/	165	/	/
11	4	41, 45, 47, 32, 49, 40, 32	41, 45, 47, 32	3	/	165	/	41.25
13	4	41, 45, 47, 32, 49, 40, 32	41, 45, 47, 32	0	/	165	/	41.25
14	4	41, 45, 47, 32, 49, 40, 32	41, 45, 47, 32	0	165	165	/	41.25
15	4	41, 45, 47, 32, 49, 40, 32	41, 45, 47, 32	0	165	173	/	41.25
16	4	41, 45, 47, 32, 49, 40, 32	41, 45, 47, 32	0	165	173	43.25	41.25
18	4	41, 45, 47, 32, 49, 40, 32	41, 45, 47, 32	0	165	173	43.25	43.25
20	4	41, 45, 47, 32, 49, 40, 32	49, 45, 47, 32	0	165	173	43.25	43.25
13	4	41, 45, 47, 32, 49, 40, 32	49, 45, 47, 32	1	165	173	43.25	43.25
14	4	41, 45, 47, 32, 49, 40, 32	49, 45, 47, 32	1	173	173	43.25	43.25
15	4	41, 45, 47, 32, 49, 40, 32	49, 45, 47, 32	1	173	168	43.25	43.25
16	4	41, 45, 47, 32, 49, 40, 32	49, 45, 47, 32	1	173	168	42	43.25
20	4	41, 45, 47, 32, 49, 40, 32	49, 40, 47, 32	1	173	168	42	43.25
13	4	41, 45, 47, 32, 49, 40, 32	49, 40, 47, 32	2	173	168	42	43.25
14	4	41, 45, 47, 32, 49, 40, 32	49, 40, 47, 32	2	168	168	42	43.25
15	4	41, 45, 47, 32, 49, 40, 32	49, 40, 47, 32	2	168	153	42	43.25
16	4	41, 45, 47, 32, 49, 40, 32	49, 40, 47, 32	2	168	153	38.25	43.25
20	4	41, 45, 47, 32, 49, 40, 32	49, 40, 32, 32	2	168	153	38.25	43.25

Step 8: Efficiency

```
1  PROMPT for powerDataFile
2  PROMPT for subArraySize
3  GET fullAraray from powerDataFile
4
5  subArray ← []
6  currSum ← 0
7  FOR i ← 0 … subArraySize – 1 // O(n)
8     subArray[i] ← fullArray[i] // O(1)
9     currSum ← currSum + subArray[i] // O(1)
10 maxAvg ← currSum / subArraySize // O(1)
11
12 FOR i ← 0 … Length(fullArray) – 1 – subArraySize // O(n)
13    prevSum ← currSum // O(1)
14    currSum ← prevSum – subArray[i] + fullArray[subArraySize + i] // O(1)
15    currAvg ← currSum / subArraySize // O(1)
16    IF currAvg > maxAvg // O(1)
17       maxAvg ← currAvg // O(1)
18    subArray[i] ← fullArray[subArraySize + i] // O(1)
19
20 PUT maxAvg
```

There are two FOR loops, each O(n) since all that's in them are O(1) processes so this means the overall efficiency would be **O(n)**