



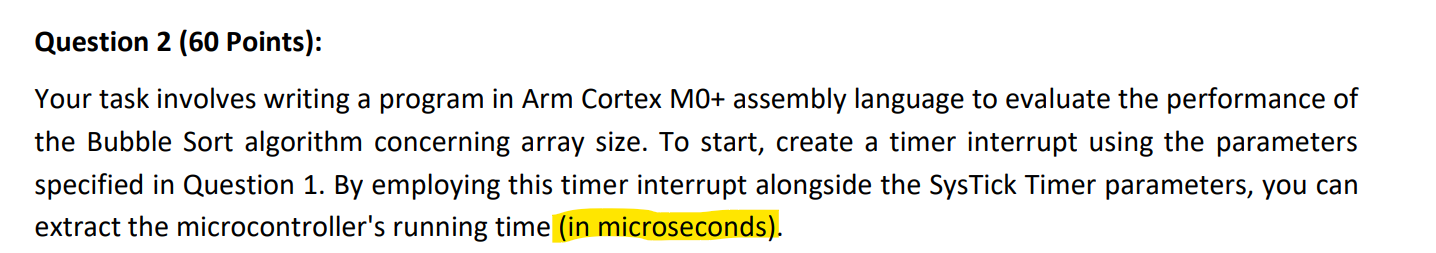
period = 1/frequency   
frequency = CPUfrequency/(1+reload) the 1 comes from the reload possibly being 0

We get:

CPUfrequency \* period – 1= reload  
And since the period is in ms, I will divide by 1000 to get seconds so that the units cancel

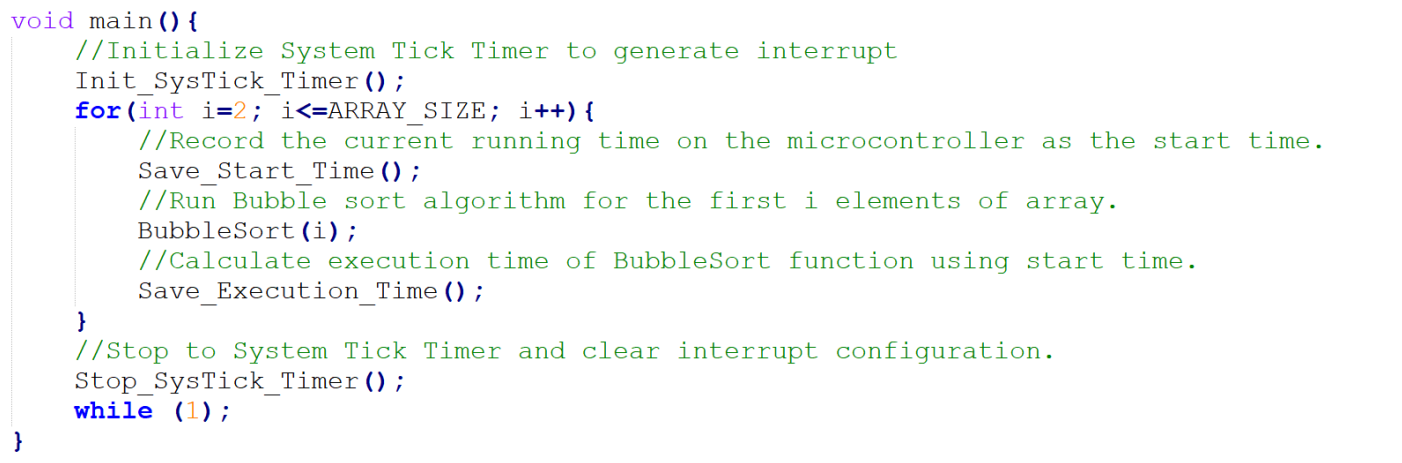
So the values are ((16\*106\*957)/1000) – 1 = reload  
thus I get 15312000 – 1 = reload = 15311999

This is if the value is in ms like its written in the excel sheet, but in the question sheet, its written as microseconds



Which would make it be 15312-1 = 15311.  
Since the first value is very big, I will be using the microseconds one.

I have a couple of comments regarding the pseudo code given as I followed it but changed it as some stuff didn’t make much sense.



Firstly, I followed the structure as I made procedures for each function call and I called them in my main function. The change that I made was regarding init\_sysTick\_Timer and Stop\_SysTick\_Timer. Despite them being called outside of the for loop, when I do that I get anomalies in my graph regarding the times, as whilst I am calculating the elapsed time, an interrupt might occur raising the value as shown in the graph below.

So to get a sensible graph, I called them during my calculation function, but they can be removed as in the pseudo code written in C. Also I assumed that the bubble sort is called with its parameter sent as a value and not a refrence or a pointer. Thus the copying time will be included in its calculation, thus I included it whilst measuring time by calling on the copying function within the bubble sort function.

In the graph we see a regular increase of time over the increase of the size of the array being sorted, bubble sort is not considered efficient for large datasets, as it has a worst-case and average-case time complexity of O(n^2). This means that its performance degrades rapidly as the size of the input array increases. And we can see this being displayed on the graph.