ISTANBUL TECHNICAL UNIVERSITY

FACULTY OF COMPUTER AND INFORMATICS

DEPARTMENT OF COMPUTER ENGINEERING

BLG212E

MICROPROCESSOR SYSTEM

HOMEWORK 2 REPORT

Çağla Mıdıklı - 150200011

Question 1)

	A	B	C	D
1	Index	Student No	CPU Frequency (MHz)	Timer Interrupt Period (ms)
85	84	150200011	8	324
86	85	150200012	64	244

Frequency = 8 MHz =$8*10^6$ (Hz)

Period = 324 ms =$324*10^{-3}$ (s)

Period = (Reload Value +1)/Frequency

Reload Value +1 = $324*10^{-3}$ (s)* $8*10^6$ (Hz)= 2592000

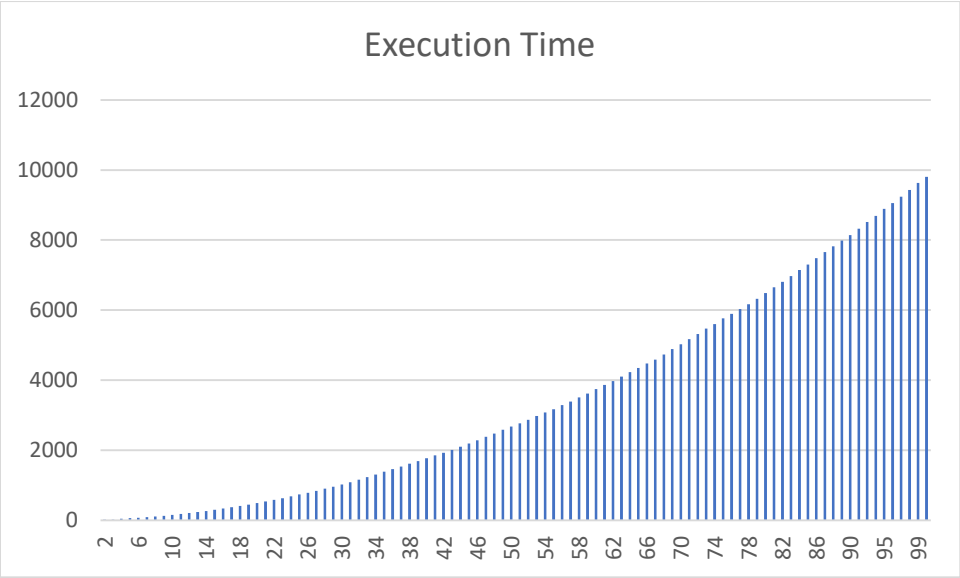
Reload Value = 2591999(decimal)= 0x00278CFF(hex)

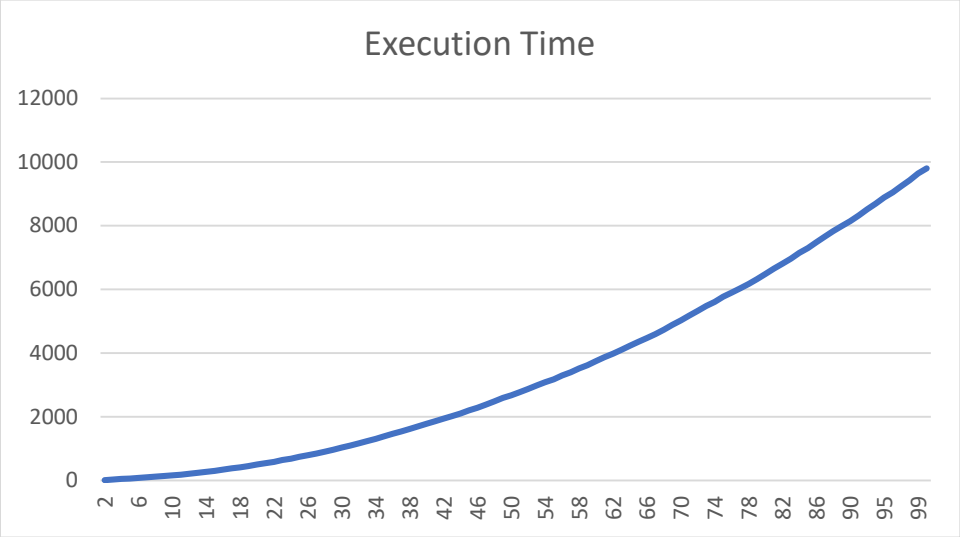
2591999 value should be loaded into the SysTick Reload Value Register.

Question 2

I initially implemented the Bubble Sort algorithm in my code and captured the current values of the SysTick timer before and after each invocation of the Bubble Sort algorithm. I was able to determine how many cycles the Bubble Sort algorithm needed to execute for that particular input size by subtracting these figures. I divided the result by the frequency value, in this case 8 MHz, to translate this to time. The value obtained is expressed in microseconds. An important note is that M = 10^6 and a microseconds = 10^-6 seconds. When entering the interrupt, I set up a counter to increment by 1 each time an interrupt occurs since the current value is reset. While calculating the time, I multiply the counter value with the reload value, add the start current value, and subtract the last current value. Finally, I divide everything by the frequency. Since the result is required in microseconds, I divided by 8 instead of 8*10^6, as micro is 10^-6.

Question 3


Execution Time
12000
10000
8000
6000
4000
2000
0
2
6
10
14
18
22
26
30
34
38
42
46
50
54
58
62
66
70
74
78
82
86
90
95
99


Execution Time
12000
10000
8000
6000
4000
2000
0
2
6
10
14
18
22
26
30
34
38
42
46
50
54
58
62
66
70
74
78
82
86
90
95
99

Address: 0x20000004

```
0x20000004: 0014900465 0064634662 0087189553 0245669050 0246949813
0x20000018: 0280758725 0284239475 0340477087 0372937696 0392765028
0x2000002C: 0417027244 0440151139 0490964731 0536855191 0541970562
0x20000040: 0558848418 0576710762 0694894443 0729953797 0730927192
0x20000054: 0763252058 0787416846 0787968636 0843544929 0870468526
0x20000068: 0895465076 0952946260 0974357170 1086364575 1264650872
0x2000007C: 1287714564 1314998912 1430510379 1501583967 1514861734
0x20000090: 1550196229 1570447789 1785898810 1862841745 1923130233
0x200000A4: 1990519977 2007637142 2020772779 2108564367 2204259110
0x200000B8: 2218162469 2222010251 2238214715 2251566410 2278411724
0x200000CC: 2338593824 2414430348 2516207000 2554958012 2555280366
0x200000E0: 2585051240 2667114332 2760581988 2793078606 2824062139
0x200000F4: 2942530061 2961555144 2971092831 2988766824 3012359258
0x20000108: 3038858172 3072620544 3119816605 3179002887 3187461713
0x2000011C: 3290743083 3321592011 3327805323 3348832871 3356236520
0x20000130: 3423692909 3499822065 3503838013 3540994501 3601671271
0x20000144: 3624845437 3649718771 3653480512 3657507297 3675722122
0x20000158: 3779594794 3870484241 3895302494 3907052087 3949877314
0x2000016C: 3951777370 3999588828 4042866969 4110485569 4135785058
0x20000180: 4211641831 4263636488 4265258539 4283354583 4294967295
0x20000194: 0000000013 0000000021 0000000032 0000000045 0000000059
0x200001A8: 0000000074 0000000092 0000000111 0000000131 0000000154
0x200001BC: 0000000180 0000000208 0000000234 0000000266 0000000300
0x200001D0: 0000000336 0000000374 0000000409 0000000451 0000000497
0x200001E4: 0000000540 0000000581 0000000634 0000000681 0000000735
0x200001F8: 0000000785 0000000844 0000000904 0000000960 0000001026
0x2000020C: 0000001091 0000001157 0000001231 0000001306 0000001385
0x20000220: 0000001464 0000001536 0000001615 0000001694 0000001775
0x20000234: 0000001856 0000001931 0000002012 0000002097 0000002196
0x20000248: 0000002280 0000002382 0000002476 0000002585 0000002674
0x2000025C: 0000002769 0000002871 0000002981 0000003077 0000003171
0x20000270: 0000003289 0000003390 0000003510 0000003616 0000003745
0x20000284: 0000003863 0000003973 0000004099 0000004227 0000004349
0x20000298: 0000004473 0000004591 0000004735 0000004884 0000005024
0x200002AC: 0000005169 0000005321 0000005477 0000005605 0000005768
0x200002C0: 0000005898 0000006032 0000006170 0000006322 0000006486
0x200002D4: 0000006653 0000006803 0000006969 0000007142 0000007300
0x200002E8: 0000007480 0000007654 0000007825 0000007988 0000008141
0x200002FC: 0000008320 0000008517 0000008694 0000008890 0000009051
0x20000310: 0000009235 0000009427 0000009635 0000009806 0000000000
```

As seen in the tables and graphics above and in memory, the execution time of the Bubble Sort increases proportionally with the size of the input array. Time array starts in 0x20000194. Bubble sort algorithm exhibits an increasing time complexity as the size of the sorted array grows. The bubble sort algorithm's average/worst time complexity is $O(n^2)$. And given input array is reverse order and this situation equals to worst case. As seen in the graph, the time complexity curve of my written code resembles the n^2 curve. This means that the bubble sort algorithm I implemented adheres to the literature, and its time complexity increases as the value of 'i' grows.