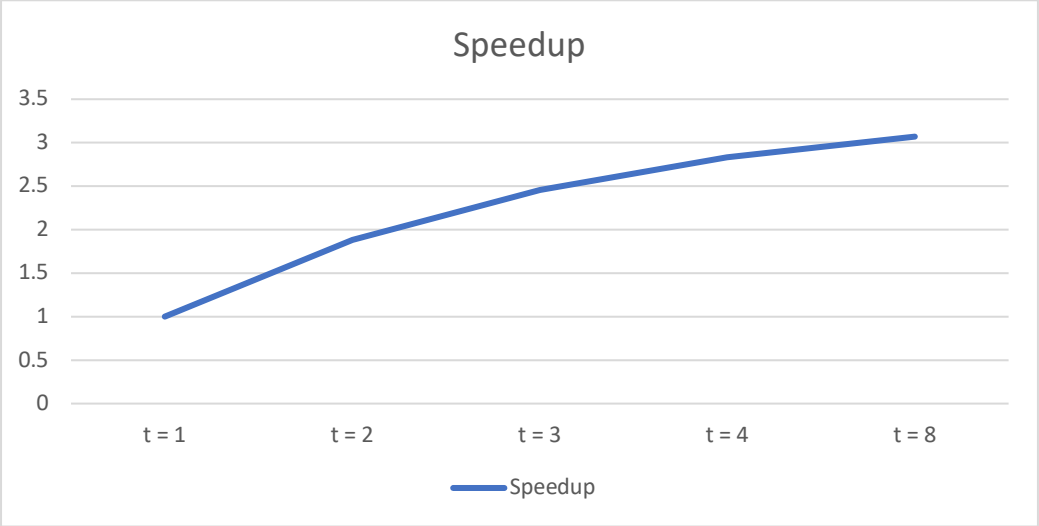
COMP409 Winter2020 A3
Eric Shen 260798146

Q1: I set my n value: 80000000; s value: 66.
I compare the following cost time of Thread t value (all time measured as an average of 5 trials):

T=1: 1653ms; T=2: 880ms; T=3: 672ms; T=4: 584ms; T=8: 538ms

Plot:



Explanation:
First, in order to find more accuracy speedup, I set n as a very large number. Then, we can see a relatively big speed up (1.88) from 1 thread to 2 thread, and speedup becomes smaller and smaller when thread number increases. Finally hits to 3.
As we learned from class, an application with p processors at most would take time that sequential time / p and critical path time (Tp <= T1/p + Tinfinite). Thus, we can see when t =2, the speed up is almost 2, which is almost linear speedup. But as t increases, the curve becomes flat.

Q2: I fixed d = 20, t = 5 and n = 1111.
I choose p and e as following: p: 4, 8, 16 and e: 1, 2, 4

The performance table (measured as time ms, average of 5 trials):

	p = 4	p = 8	p = 16
e = 1	12341	12751	12673
e = 2	12454	12578	12747
e = 4	12402	12754	12521

Explanation:
Since n is relatively large and we always have parallel threads. They almost end at same time, thus the program executes time is almost the same. The slower or faster our program mainly depends on d value.
But elimination array effectiveness depends on its size e. I think my array is used more effectively when p is larger. To be detail, when p = 4 and e = 4, array is not that effective since there are not many data changes because array's capacity is four and we only have 4 threads. As we learned from class, the larger elimination array, the harder thread will meet at same position, if thread fixed. However, the smaller array size, the harder for thread to find an empty space. For example, for p = 16, and e is 4, gives us better effectiveness.