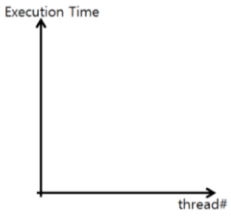
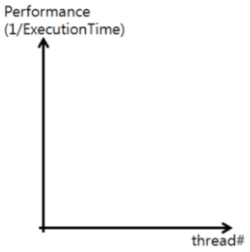
[Problem 1] In our lab. class, we looked at the JAVA program (ex4.java) that computes the number of 'prime numbers' between 1 and 200000. The java code (ex4.java) creates threads for parallel computation using static load balancing approach. However, The parallel implementation of ex4.**java** does not give satisfactory performance because of bad load balancing. The problem is that (i) higher ranges have fewer primes and (ii) larger numbers are harder to test. Therefore thread workloads become uneven and hard to predict. For better performance, we implemented dynamic load balancing approach as project 1 where each thread takes a number one by one and test whether the number is a prime number.

(i) Write 'C with OpenMP' code that computes the number of prime numbers between 1 and 200000. Your program should take two command line arguments: scheduling type number (1=static, 2=dynamic, 3=guided) and number of threads (1, 2, 4, 8, 16) as program input argument. Use **schedule(static)** , **schedule(dynamic,4)** , and **schedule(guided,4)**. Your code should print the execution time as well as the number of the prime numbers between 1 and 200000.
command line execution: > **a.out scheduling_type# #_of_thread**
execution example> **a.out 2 8** <---- this means the program use dynamic scheduling using 8 threads.

(ii) Write a document that reports the parallel performance of your code. The graph that shows the execution time when using 1,2,4,8,16 threads. There should be at least three graphs the show the result of static, dynamic, and guided scheduling policy. Your document also should mention which CPU (dualcore? or quadcore?, clock speed) was used for executing your code.





exec time	1	2	4	8	16
static					
dynamic					
guided					

performace (1/exec time)	1	2	4	8	16
static					
dynamic					
guided					