Part C: Experiments

a) For the multi-processes application, I run the code with n = 10000000 lines of randomly generated inputs with a different number of processes while preserving the number of inputs 10000000. I did separate inputs into files in order to create many processes but did not change input n while doing this. Time elapsed counted in microseconds to get better insight.

(all time elapsed outputs belong to different run)
with 1 process running:
174521 ms
128575 ms
99952 ms

with 2 processes running:
325466 ms
238094 ms
377879 ms

with 4 processes running:
296163 ms
105984 ms
213649 ms

with 8 processes running:
280280 ms
235488 ms
196600 ms

In general, we see the results as we expected. Because my computer has only 1 CPU so additional processes did not speed up the program. I also expected slowdown because of context switches. Even though it is not obvious we can see some slowdown with the increase of process number.

For multi-threaded program

(all time elapsed outputs belong to different run)
with 1 thread running:
99380 ms
160131 ms
159677 ms

with 2 threads running:
160011 ms
166511 ms
209943 ms

with 4 threads running:
216130 ms
780121 ms
221739 ms

with 8 threads running:
143928 ms
837191 ms
303927 ms

again speed up did not occur and program slowdown occurred as number n threads are increasing.

b) for 2 processes

with n = 10000000:
504687 ms
830012 ms
723022 ms

with n = 5000000:
325466 ms
238094 ms
377879 ms

with n = 2500000:
271223 ms
124530 ms

147241 ms

with n = 1250000:
174136 ms
384917 ms
224249 ms

By recursion I estimated running time of my algorithm as T(n) = 2T(n/2) +n so by substitution theorem run time is T(n) =**θ**(nlogn). So i expected a speed up for time elapsed while n approaches 0. The program somewhat gave my expectations with the exception of last n input.