The parallel or serial scheduling problem

work efficient: work is same as serial alg. Most parallel algorithms aren't work efficient It even matters if the work is different by a constant factor.

Problem:

- Computer gets a bunch of tasks over time.
- Each task has a serial and parallel implementation.
- Goal: high throughput
  - **modified time**: time steps where all tasks given so far have been completed.
  - There are n tasks (we don't know n in advance).
  - Goal more formally: minimize total modified time until all tasks are completed.

There is a tradeoff for the scheduler. If there are a lot of tasks, then it makes sense to schedule them all in parallel, because the serial implementations are faster and paralelization across tasks is possible. On the other hand, if there are very few tasks, then the scheduler should probably run the parallel implementations of the tasks, even though they aren't work efficient, so that it at least achieves some amount of parallelism.

This problem has a parameter p (which is fixed. lolololololololol).

Further problem notes:

- The scheduler is alowed to stop tasks and restart them using a different implementation
  - unles we want to decide that decisions are irevokable
- All problem factors are up to us. But probably version closest to reality is best.
- for now assume span is $O(1)$.

Our objective is to prove something like one of these:

- $O(1)$-competitive: i.e. total running time is within a constant factor of OPT
- $O(\log p)$-competitive alg
- lowerbound: $\omega(1)$? impossbile to be $O(1)$-competitive
- suppose you have resource augmentation over OPT, then can you do better?
  - e.g. time steps are 1/2 of OPT's time steps
  - e.g. all work is divided by resource augmentation factor

misc stuff

- need to do lit review to make sure it is an open question
- cool further question: what if the program is recursive???? [later]