Daniel Ginsburg
HW11
Algorithm:
Let SC = supercomputer
We simply order the jobs by longest f time to shortest f time (O(nlog(n))). We then feed them into the SC one at a time (longest f time first). When the SC finishes processing the job we feed it into one of the regular computers and feed the next job into the SC. We continue until all jobs have been processed. Besides for the sorting which is O(nlog(n)), everything else is O(n) so the algorithm is O(nlogn).

Proof:
I will use proof by Contradiction...I think. (If not then it's just a proof but the instructions said any method acceptable so here we go...)
Let T = the total time taken process the jobs.
To prove our solution is optimal, we must prove that T_n is the lowest achievable. T is essentially comprised of all p_i plus the length of time it takes the last job to finish on a PC after the SC has finished processing all jobs.
We know that every job must be fed into the SC, thus $T \geq \sum_{i=1}^{n} p_i$. The order of the p_i's doesn't affect T because no matter how they are ordered, the SC must process all of them. However, the order of the $f's$ does affect T. Thus our algorithm aims at minimizing how much time the f part of the job affects T.

First we state that if a solution has any delay (as in we wait before feeding the next job into the CS) we can simply remove the delay without increasing T. This is obvious because delaying the start time can only make things take longer (or if we are lucky keep them the same). Thus our solution will contain no delays.

In any given ordering, there will be one job who's f component finishes last (or multiple in the case of a tie). Let j_x be that job. We prove that with the greedy ordering G, there is no way to decrease the amount of time the latest f component will add to T, thus there is no way to decrease T, so T is the lowest it can be and therefore the solution is optimal. Let us assume that there was some way to decrease T_g. This can either be made by moving j_x and therefore its f component, either earlier or later.

Move j_x later:
This will simply increase T. Because j_x is starting later, it will end later and T will be increased.

Move j_x earlier:
Let us say we swap j_x with j_a. In this case j_x will finish earlier, and seemingly lower T. However, because all the f values before j_x are higher than (or equal to) f_x (definition of our greedy algorithm), T will increase. Since we swapped j_a and j_x, that means f_a starts at the same time f_x would have started. Thus, because it is longer, it must finish AFTER f_a originally finished and thus be-

comes the latest f value and adds MORE to T than f_a did originally. (If there are multiple jobs with the same f component the swapping them will neither increase nor decrease the time, so it won't be better than our solution).

Because the only possibilities are to either more j_x forward or backwards, and neither will decrease T, there is no way to improve our greedy answer. Thus greedy is optimal.