laackvet Thor Laack-Veeder Problem 2.1, Hand-in 3 CS252, Winter 2014

Time estimate: 300 minutes Collaborated with Pat Dale, Colby Seyferth

I will begin by assuming zero base in this problem.
Given:
Set R = $\{r_1, r_2, ..., r_n\}$, the set of robots.
Set G = $\{g_1, g_2, ..., g_n\}$, the set of generators.
r_i's preference order on g $\forall$ i

Output:
A perfect matching of R to G with no instabilities.
A matching of R ang G is a pairing {(r,g) | r $\in$ R, | g $\in$ G} and no r or g is matched > once.
A perfect matching is one which every generator is shut down by one robot, that is, everyone has a match.
An instability exists if (r_1, g_2) are matched, but r_2 visits g_2 after match, destroying the robot and leaving an undestroyed generator later.

Algorithm

1. Let P represent the generator preference list, which is the reverse of the list of the sorted list of robot services for that generator.
   (a) For example, let r_n represent the last robot to service generator g_n and let r_k represent the second to last robot to service generator g_n. P[g_n] = [r_n,r_k].
2. Let r_n represent the robot that services a particular generator, g_n, last.
   (a) Note: r_n can be a single robot or a group.
   (b) Note: If r_n is a group of robots, all the robots in that group will visit a different generator first because no two robots can service the same generator at the same time.
3. For each robot from r_n to r_1
   r_i sets to destroy first scheduled generator
   Generator will accept/be destoryed if:
   (a) open/hasn't been set to be destroyed by a particular robot
   or
   indice of r_i in P[ ] < indice of current match, r_j in G[ ].

Proof of Correctness

Claim: The algorithm termiates in $\leq n^2$ steps.
Proof: Each iteration through the list causes a new service/virus request. There are n^2 possible service/virus requests.

Claim: The algorithm returns a perfect matching.
Proof: Suppose not.
Then $\exists$ an un-destroyed generator g $\in$ G. Then no robot destroyed g. $|R| = |G| = n$, therefore $\exists$ a robot that did not destroy a generator. Therefore, r is an unmatched robot who hasn't been designed to destroy a particular generator. Therefore, the algorithm did not terminate, which is a contradiction. Because there exists a contradition, we reject the notion that the algorithm does not return a perfect matching.

Proof of Efficiency
Claim: The algorithm's efficiency is on the order of $O(n^2)$

Proof: To prove the algorithm's efficiency, I want to look at both the pre-processing scheme, and then the match creation of robot and generator.
In the pre-processing scheme, the algorithm constructs an array P, to represent the preference list for a particular generator g, which takes into account the service schedules of each robot. Again, the preference list for a particular generator g is simply the reverse of the list of robots servicing g throughout the day, where r_n is the last robot to service g, and the first element of array P. Because there are n robots that serve each generator, to construct a preference list, P, for a given generator g, will be on the order of O(n).
$|R| = |G| = n$, therefore there must a preference list created for n generators, which makes the efficiency $O(n * n)$, which equals $O(n^2)$.
After the preference list for each generator is created, the matching scheme begins.
For each robot from r_n to r_1, r_i will see if it is to destroy its first scheduled generator. Because there are n robots, this step will be O(n). The generator will accept if it is open to be destroyed, of if the particular r_i is higher on its preference list. Again, since there are n robots trying to destory n generators, this operation will also be O(n). Because the each process described is implemented with a for loop and they are nested for loops, you again multiply the efficiencies of each process, O(n * n), which becomes $O(n^2)$.
To determine the total efficiency of the algorithm, you add the efficiencies of the pre-processing scheme and matching-scheme, $O(n^2 + n^2)$, which equals $O(2n^2)$, which equals $O(n^2)$.