(a) Give an algorithm in pseudocode (either an outline or paragraph works) to find a stable assignment that is hospital optimal. Hint: it should be very similar to the Gale-Shapley algorithm, with hospitals taking the role of the men, and residents of the women.

```
Initialize all hospitals(h) ∈ m and students(s) ∈ n to free
Pre-processing students' ranking of the hospitals in an inverse preference array.
  while ∃ hospital(h) who still has slots available and has not proposed to every student in n {
    s = first student on h's list to whom h has not yet proposed
    if (s is free)
      (h, s) become paired
    else some pair (h', s) already exists
      if s prefers h to h'
        h' becomes free
        (h, s) become paired
      else
        (h', s) remain paired
  }
  Return the set S of assigned pairs
```

(b) Give the runtime complexity of your algorithm in (a) in Big O notation and explain why.

Note: Full credit will be given to solutions that have a complexity of O(mn).

For the preprocessing, creating an inverse preference array takes about (m) time for each student and there are total (n) students. So the total time is (m*n). However, we must only create the array once at the beginning of the algorithm, not in each execution of the while loop.

For the rest of algorithm, assuming the worst case that each hospital proposed to all students (n) in order to get its slots occupied. And there are total m hospitals (m). Therefore, the total time is (m*n). Then the runtime complexity is 2*m*n, which is clearly O(mn).

(c) Give an algorithm in pseudocode (either an outline or paragraph works) to find a stable assignment that is resident optimal Hint: it should be very similar to the Gale-Shapley algorithm, with residents taking the role of the men, and hospitals of the women.

```
Initialize all hospitals(h) ∈ m and students(s) ∈ n to free

Pre-processing hospitals' ranking of the students in an inverse preference array.

  while ∃ student(s) who still has not been assigned and has not proposed to every hospital {
     h = first hospital on s's list to whom s has not yet proposed
     if (h still has slots)
        (s, h) become paired
     else some pairs (s', h), (s'', h) ), (s''', h)  already occupy all slots
        if h prefers s to least_pref (s', s'' and s''')
           least_pref (s', s'' and s''') becomes free
           (s, h) become paired
        else
            (s', h) and (s'', h) remain paired
  }
  Return the set S of assigned pairs
```

(d) Give the runtime complexity of your algorithm in (c) in Big O notation and explain why.

Note : Try to make your algorithm as efficient as you can, but you may get full credit even if it is not O(mn) as long as you clearly explain your running time and the difficulty of optimizing it further.

For the preprocessing, creating an inverse preference array takes about (n) time for each hospital and there are total (m) hospitals. So the total time is (m*n). However, we must only create the array once at the beginning of the algorithm, not in each execution of the while loop.

For the rest of algorithm, assuming the worst case that each student proposed to all hospitals (m) in order to get assigned. When a student proposed to a hospital that is already full, we must first find the lowest rank of the current assigned student, which takes x time (the number of slots of that hospital). Then we can compare the new student and the lowest rank student to decide whether he can be assigned. Therefore, assuming the total number of slots of all hospitals is X, for each student, the worst case is to compare with every lowest rank student in each hospital. So the worst case running time is X*n.

Then the runtime complexity is O(m*n+ X*n), where X is bigger than m.

The difficulties of optimizing it are how to decide whether the student can be assigned into hospital if it is already full within run time O(1). Perhaps for each hospital we can add a variable to keep track of the current lowest ranking student in its slots and compare with student who is currently proposing. But still, it is hard to implement such a variable without adding additional running time.

(e) Use the Brute Force Implementation (see below) to verify that your algorithm is indeed resident optimal.

By using Brute Force Implementation, we can find every stable matching in a given input, from which we can pick up the resident optimal solution. It can be used to verify whether the Gale-Shapley algorithm returns a resident optimal matching.