1a.

Create reverse preference list – O(n + mm) = O(nm)

Add all locations to queue – O(m)

while queue is not free

while currentLocation (first location in queue) has opening

propose to top employee (at currentLocation.counter),

if employee not matched yet-

currentLocation.numOpenings -= 1

match location and student

else if currentLocation ranked higher-

currentLocation.numOpenings -= 1

match currentLocation and employee/ unmatch oldLocation

oldLocation.numopenings += 1

add oldLocation to queue

else (lower ranked) do nothing

currentLocation.counter++

remove currentLocation from queue

return set of matchings

1b.

Suppose the number of employees will always be greater than the number of locations, and that the maximum number of openings is n. Let ai = the number of slots at an arbitrary location i. The maximum total number of proposals for m locations is-

n + (n – a0) + (n – a0 – a1) + … + am-1 => n[1 + (1 – a0/n) + (1 – a0/n – a1/n) + … + am-1/n]

=> n[m – ((m-1)a0 + (m-2)a1 + … (m-i-1)ai)/n]

Factoring out m in the numerator after multiplying the a’s,

=> n[m – (mn – (a0 + 2a1 + … + (m+1)ai)/n]

The expression a0 + 2a1 + … + (m+1)ai is less than mn because you would need m a0 + a1 + … ai . Therefore, we can say,

=> n[m – (mn – mn)/n] = O(nm)

1c.

Create reverse preference list – O(nm)

Add all employees to queue – O(n)

Each employee has a separate counter (num) for each location corresponding to the num of slots for each location (not currentEmployee.counter)

while queue is not free

if currentEmployee.counter == m (number of locations) , remove from queue and continue

currentEmployee proposes to top location choice (currentEmployee.counter)

currentEmployee.counter++

remove currentEmployee from queue

propose to location.copy[num]

if ranked higher

add oldEmployee to queue (if exist)

match currentEmployee and location (and unmatch old (if exist))

else add currentEmployee back into queue

if location.numSlots == num

num = 0

currentEmployee.counter++

else

num++

return set of matchings

1d.

The initial size of the queue is n. Let ai = the number of slots at an arbitrary location i. In particular, all the slots summed should be less or equal to n. The maximum number of proposals is-

a0 + a1 + a2 + a3 + … + am-1 +

a0 + a1 + a2 + a3 + … + am-1 - 1+

a0 + a1 + a2 + a3 + … + am-1 - 2+

etc.

In particular, locations are first filled before any ranks are checked.

=> n + (n-1) + (n-2) + (n-3) + … + 1 = n(n+1)/2 = O(n^2)

It is difficult to improve this further because we must somehow be aware of the minimum rank currently assigned to a location. In other words, we need an O(1) replacement policy. However, sorting an array takes O(nlogn) at minimum and finding the minimum for a location can take O(n).