Problème 1 :

1 Par l'absurde, si l'algorithme ne se termine pas en renvoyant les n couples, c'est qu'on arrive dans une situation où au moins un des garçons "non casés" est arrivé au bout de son classement. Appelons ce garçon Tristan. Tristan a, donc, dans les tours précédents, fait une proposition à toutes les filles et a été rejeté, soit directement, soit au bout de quelques étapes dès que la fille avec qui il était en couple a reçu une proposition plus intéressante (d'un garçon mieux classé). Cependant, toutes les filles à qui il a fait une proposition sont en couple, car :

- Soit elles ne l'étaient pas lors du tour de la proposition de Tristan et lès dès le tour suivant (soit avec Tristan soit un autre mieux classé).
- Soit elles l'étaient déjà lors du tour de la proposition de Tristan.

Or, lorsqu'une fille est en couple, elle le reste jusqu'à la fin de l'algorithme (pas nécessairement avec la même personne). Par conséquent, les n filles sont en couple à la fin de l'algorithme, or, elles le sont avec des garçons différents et il y a n garçons, c'est une contradiction avec le fait que Tristan n'est pas sensé être en couple.

2 (a) Si on garde les prénoms de l'exemple, au premier tour, Alexis va faire une proposition à Camille, comme elle est en tête de son classement. Camille acceptera la proposition d'Alexis, vu qu'il est en tête de son classe. Si durant les différentes étapes, Camille reçoit d'autres propositions, c'est forcément pour des garçons plus bas dans son classement, par conséquent elle gardera Alexis jusqu'à la fin.

(b) Non, par exemple, avec l'exemple donné dans l'énoncé, si Alexis continue de proposer, il sera forcément accepter par Marwa, car il est en tête de son classement. En conséquence, il ne sera plus avec Camille. De plus, un tel algorithme risque de créer des personnes célibataire : on peut se convaincre que Camille sera célibataire car elle perdra Alexis au 3e tour et n'a pas de proposition au 3e tour (qui sera le dernier, il y a autant de tour que de filles dans un tel algorithme).

3 A chaque étape, comme il reste au moins un garçon, une proposition est faite. Or, il ne peut y avoir qu'au plus $n \times n$ propositions (le nombre de filles $\times$ le nombre de classement), donc l'algorithme s'arrête en plus n^2 étapes.

4 (a) `"Nicolas"`

(b) `['Marwa', 'Camille', 'Lucie']`

5 `[0]*len(NomG)` ou `[0 for i in NomG]`

6 `['']*len(NomF)` ou `[' ' for i in NomF]`

7 `[1, 2, 3]` et `['Nicolas', 'Alexis', 'Rémi']`

8
```
def verification(L,M):
    for k in L :
        if k not in M:
            return False
    return True
```

9

```
def verification2(LL,M):
    for l in LL:
        if not verification(l,M):
            return False
    return True
```

10 `verification2(choixF,NomG)` et `verification2(choixG,NomF)`

11

```
def ordre(a,b,L):
    for k in L:
        if k==a:
            return True
        if k==b:
            return False
    print("Problème : les éléments demandés ne sont pas dans la liste")
    return
```

12

```
def arrêt(ind,l,NomG,selection):
    for k in range(len(ind)):
        if ind[k]!=l and NomG[k] not in selection:
            return True
    return False
```

13

```
def Recherche_indice(L,a):
    for k in range(len(L)):
        if L[k]==a:
            return k
    print("erreur a n'est pas dans L")
    return
```

14

```
while arrêt(ind,len(NomF),NomG):
    for k in range(len(NomG)):
        if NomG[k] not in selection:
            if ind[k]!=len(NomF):
                r=Recherche_indice(NomF,choixG[k][ind[k]])
                if selection[r]=='':
                    selection[r]=NomG[k]
                else :
                    if ordre(NomG[k],selection[r],choixF[r]):
                        selection[r]=NomG[k]
                ind[k]=ind[k]+1

print(selection)
print(NomF)
```

Problème 2 :

1
```
def pas(mu,x):
    return mu*x*(1-x)
```

2
```
def suitelogi(mu,x0,n):
    x=x0
    L=[x0]
    for k in range(n):
        x=pas(mu,x)
        L=L+[x]
    return L
```

3
```
def trace(mu,x0,N):
    Y=suitelogi(mu,x0,N)
    plt.plot(range(N+1),Y,'-o')
    plt.title('mu ='+str(mu)+', x0='+str(x0) + ', N= '+str(N))
    plt.show()
```

4
```
def beg_gauche(L):
    M=[L[0]]
    for k in L[1:-1]:
        M=M+[k,k]
    return M
```

5
```
def beg_droite(L):
    M=[]
    for k in L[1:-1]:
        M=M+[k,k]
    return M+[L[-1]]
```

6
```
def trace2(mu,x0,N):
    L=suitelogi(mu,x0,N)
    XX=pointsequidi(0,1,100)
    YY=[pas(mu,k) for k in XX]
    X=beg_gauche(L)
    Y=beg_droite(L)
    plt.plot(X,Y,'.--r',label='représentation des valeurs de la suite')
    plt.plot(XX,YY,'-b',label="droite y="+str(mu)+"x(1-x)")
    plt.plot(XX,XX,color='black',linestyle="--",label="droite d'équation y=x")
    plt.title('mu ='+str(mu)+', x0='+str(x0) + ', N= '+str(N))
    plt.legend()
    plt.show()
```

7
```
def pointsequidi(a,b,N):
    pas=(b-a)/(N-1)
    L=[a]
    pos=a
    for k in range(N-1):
        pos=pos+pas
        L=L+[pos]
    return L
```

8 La variable `N` indique le nombre de points utilisés dans le segment $[3,4]$.

9 La ligne 3 calcule la liste des termes de la suite de 0 à 500 à μ fixé, mais on ne récupère que les éléments d'indice supérieur ou égal à 400. Comme ils servent d'ordonnées à des points d'abscisse mu, on suppose que les termes sont suffisamment proches d'une valeur d'adhérence de la suite (u_n).

10 Une boucle for est effectuée sur `pointsequidi(3,4,N)` qui est une liste contenant N points. Dans la boucle, aucune des instructions ne dépend de N, elles s'effectue en une complexité de $\Theta(1)$ (le nombre d'opérations élémentaires est constant et ne dépend pas de N). L'algorithme a une complexité de $\Theta(1) \times N = \Theta(N)$, il est également linéaire.