```
//By James Partsafas (40170301) and Christina Darstbanian (40097340)

Algorithm rearrangeParticipants(pName, pDOB, n)  Assigning so becomes 3logn
Input: String array pName with corresponding string array pDOB, both of length n
Output: return number of seniors (those 65 years old or more) while reordering
array such
that seniors are listed chronologically at start and non-seniors are listed
reverse-
chronologically immediately after them

//Get date (use current date)
int currentDate <- 20210521 //formatted yyyymmdd       logn
int seniorAge <- 650000 //represents 65 years old in dating system used here.  logn

//Special cases
if (pName.length != pDOB.length OR length = 0) then     2logn
       return -1                                          logn
if (n = 1) then //Only one person in array                logn
       int age <- getAge(pDOB[0])                        logn * 284
       if (age >= seniorAge) then                  logn
              return 1                                    logn
       else
              return 0                                    logn

//General case now
int mid <- n/2             assigning logn times
string[mid] leftName        assigning logn times
string[mid] leftDate        assigning logn times
string[n-mid] rightName      assigning logn times
string[n-mid] rightDate   assigning logn times


for i <- 0 to mid - 1 do            logn * (1 + 1+n)
       leftName[i] <- pName[i]          nlogn
       leftDate[i] <- pDOB[i]           nlogn
       increment i                        nlogn
for i <- mid to n - 1 do                  logn * (1+1+n)
       rightName[i-mid] <- pName[i]        nlogn
       rightDate[i-mid] <- pDOB[i]           nlogn
       increment I                          nlogn

//Create tree-like structure recursively
rearrangeParticipants(leftName, leftDate, mid)         logn
```

```
rearrangeParticipants(rightName, rightDate, n-mid)          logn

//call mergeParticipants on every part of tree
int seniorCount <- mergeParticipants(pName, pDOB, leftName, leftDate, rightName,
rightDate, mid, n-mid)   logn * 18+1193n

return seniorCount           logn


*************************************************************************
*


Algorithm mergeParticipants(pName, pDOB, leftName, leftDate, rightName, rightDate,
left, right)                    8
Input: string arrays pName, pDOB, leftName, leftDate, rightName, rightDate and
integers left, right
Output: The number of seniors (people over 65 years of age) while also reordering
the array in the manner specified
by rearrangeParticipants above.

//i traverses left lists, j the right ones, and k the final sorted one
int i <- 0       1
int j <- 0        1
int k <- 0           1

int seniorCount <- 0             1
int seniorAge <- 650000            1

//Move through left and right lists and reorder original in the process
int leftAge           1
int rightAge             1
boolean leftIsSenior      1
boolean rightIsSenior     1

while(i < left AND j < right)          n +n
        leftAge <- getAge(leftDate[i])  n*284
        rightAge <- getAge(rightDate[j])     n*284
        leftIsSenior <- false  n
        rightIsSenior <- false    n

        if (leftAge >= seniorAge) then  n
                leftIsSenior <- true          n
```

```
        if (rightAge >= seniorAge) then          n
                rightIsSenior <- true            n

        if (leftIsSenior AND NOT rightIsSenior) then //left is senior, right isn't
                        2n
                pName[k] <- leftName[i]            n
                pDOB[k] <- leftDate[j]              n
                increment i                 n
                increment k                   n
                increment seniorCount           n
        else if (rightIsSenior AND NOT leftIsSenior) then //right is senior, left
isn't                 2n
                pName[k] <- rightName[j]           n
                pDOB[k] <- rightDate[j]             n
                increment k         n
                increment j             n
                increment seniorCount       n
        else if (rightIsSenior AND leftIsSenior) then //both are seniors
                                2n
                increment seniorCount               n
                if (leftAge >= rightAge) then            n
                        pName[k] <- leftName[i]         n
                        pDOB[k] <- leftDate[i]           n
                        increment i       n
                        increment k           n
                else
                        pName[k] <- rightName[j]          n
                        pDOB[k] <- rightDate[j]           n
                        increment k             n
                        increment j              n
        else //both are juniors
                if (leftAge <= rightAge) then        n
                        pName[k] <- leftName[i]        n
                        pDOB[k] <- leftDate[i]           n
                        Increment i                 n
                        increment k               n
                else
                        pName[k] <- rightName[j]   n
                        pDOB[k] <- rightDate[j]     n
                        increment k        n
                        increment j        n
```

```
//While loop is done. Make sure both left and right lists are exhausted before
returning
while (i < left) do          n
       if (getAge(leftDate[i]) >= seniorAge) then          n*284 + n
              increment seniorCount     n
       pName[k] <- leftName[i]            n
       pDOB[k] <- leftDate[i]         n
       increment i             n
       increment k             n
while (j < right) do                n
       if (getAge(rightDate[i]) >= seniorAge) then           n*284 + n
              increment seniorCount n
       pName[k] <- rightName[j]            n
       pDOB[k] <- rightDate[j]              n
       increment k                 n
       increment j                  n

return seniorCount               1


**********************************************************************************
*


Algorithm getAge(dob)            1
Input: String dob representing date of birth, formatted as day-month-year (strings
are understood as an array of chars)
Output: An integer representing age.

String[3] stringDate           1
int word <- 0             1
int letter <- 0         1
//Get day, month, and year in array of strings
for (i <- 0 to dob.length - 1) do              10
       if (dob[i] = '-') then          2
              increment word         2
              letter <- 0           2
       else
              stringDate[word][letter] <- dob[i]         8
       increment i                11

//Verify that day and month are not single digit and add 0 at start if that's the
case
```

```
if (stringDate[0].length = 1) then              1
       stringDate[0] <- '0' + stringDate[0]   1+1
if (stringDate[1].length = 1) then          1
       stringDate[1] <- '0' + stringDate[1]       1+1

string fullDate <- stringDate[2] + stringDate[1] + stringDate[0] 1 +1+1

//Convert this string to an integer
int date <- 0   1
int temp
int power <- 1          1
for (i <- fullDate.length - 1 to 0) do            1+1+8
       temp <- (int) fullDate[i]            8
       for (j <- 0 to power - 1) do      (1+ 1)*(36)
              temp = temp * 10              (1+1)*36
              increment j               36
       date = date + temp           (1+1)*8
       increment power                8
       decrement i                    8

int currentDate <- 20210521 //formatted yyyymmdd           1

date <- currentDate – date         1+1

return date                     1
```