|  |
| --- |
|  |
|  | //By James Partsafas (40170301) and Christina Darstbanian (40097340) |
|  |  |
|  | Algorithm displaySeniorsIncreasingOrder(pName, pDOB, n) Assigning so becomes 3logn |
|  | Input: String array pName with corresponding string array pDOB, with n representing number of seniors (those 65 years old or older) |
|  | Output: Display the seniors ordered chronologically by date of birth |
|  |  |
|  | //Get date (use current date) |
|  | int currentDate <- 20210521 //formatted yyyymmdd logn |
|  | int seniorAge <- 650000 //represents 65 years old in dating system used here. logn |
|  |  |
|  | int length <- pName.length logn |
|  |  |
|  | //Special cases |
|  | if (length != pDOB.length OR length = 0) then 2logn |
|  | Return |
|  | if (length = 1) then //Only one person in array logn |
|  | Return |
|  |  |
|  | //General case now |
|  | int mid <- length/2 logn |
|  | string[mid] leftName logn |
|  | string[mid] leftDate logn |
|  | string[length-mid] rightName logn |
|  | string[length-mid] rightDate logn |
|  |  |
|  | 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 length - 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 |
|  | displaySeniorsIncreasingOrder(leftName, leftDate, n) logn |
|  | displaySeniorsIncreasingOrder(rightName, rightDate, n) logn |
|  |  |
|  | //call mergeSeniors on every part of tree |
|  | mergeSeniors(pName, pDOB, leftName, leftDate, rightName, rightDate, mid, length-mid) logn \* (13+589n) |
|  |  |
|  | //Check if seniors are ordered |
|  | boolean done <- false logn |
|  | int counter = 0 logn |
|  | for (i <- 0 to length - 1) do logn \* (1 + 1+n) |
|  | if (getAge(pDOB[i]) >= seniorAge) logn\* 284 + n |
|  | increment counter nlogn |
|  | increment i nlogn |
|  | if (counter = n) then logn |
|  | done <- true logn |
|  |  |
|  | if (done) then logn |
|  | for (i <- n-1 to 0) do logn \* (1 + 1+n) |
|  | display (pName[i] + "\t" + pDOB[i]) nlogn |
|  | decrement i nlogn |
|  |  |
|  | return |
|  |  |
|  |  |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  |  |
|  |  |
|  | Algorithm mergeSeniors(pName, pDOB, leftName, leftDate, rightName, rightDate, left, right) 8 |
|  | Input: string arrays pName, pDOB, leftName, leftDate, rightName, rightDate and integers left, right |
|  | Output: Reorders the arrays such that everything is in chronological order |
|  |  |
|  | //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 |
|  |  |
|  | //Move through left and right lists and reorder original in the process |
|  | int leftAge 1 |
|  | int rightAge 1 |
|  |  |
|  | while(i < left AND j < right) do n +n |
|  | leftAge <- getAge(leftDate[i]) n\*284 |
|  | rightAge <- getAge(rightDate[j]) ]) n\*284 |
|  |  |
|  | 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 |
|  | pName[k] <- leftName[i] n |
|  | pDOB[k] <- leftDate[i] n |
|  | increment I n |
|  | increment k n |
|  | while (j < right) do n |
|  | pName[k] <- rightName[j] n |
|  | pDOB[k] <- rightDate[j] n |
|  | increment k n |
|  | increment j n |
|  |  |
|  | return |
|  |  |
|  |  |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  |  |
|  |  |
|  | 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 |