2.28 Ja sabeu que fer la fusió ordenada de dues seqüències ordenades d'm i n elements, respectivament, comporta fer m + n moviments de dades (penseu, per exemple, en un merge durant l'ordenació amb mergesort d'un vector). Però si hem de fer la fusió d'N seqüències, dos a dos, l'ordre en què es facin les fusions és rellevant. Imagineu que tenim tres seqüències A, B i C amb 30, 50 i 10 elements, respectivament. Si fusionem A amb B i després el resultat el fusionem amb C, farem 30 + 50 = 80 moviments per a la primera fusió i 80 + 10 = 90 per a la segona, amb un total de 170 moviments. En canvi, si fusionem primer A i C i el resultat el fusionem amb B farem un total de 130 moviments.

Dissenyeu un algorisme golafre (greedy) per fer les fusions i obtenir la seqüència final ordenada amb mínim nombre total de moviments. Justifiqueu la seva correctesa i calculeu-ne el cost temporal del vostre algorisme en funció del nombre de seqüències N .

Donada una llista Sol, inicialment buida. Sol serà la llista amb l'ordre de les cadenes òptim.
Donada la llista S amb totes les seqüències:

Afegir cada cadena de S a una cua de prioritats. Aquesta cua estarà ordenada pel nombre d'elements de la seqüència de cada cadena de manera creixent.

```
while(S > 2)
        a <- S.extract_min()
        b <- S.extract_min()
        c <- merge(a,b)
        S.insert(c, /c/)    // on /c/ es la clau
fi while
return merge(S[0], S[1])
```

El criteri que s'utilitza, és fusionar les 2 cadenes més petites sempre.
El codi anterior donarà una solució seguint el criteri anterior, ja que el que s'està fent és tenir ordenat S pel nombre de cadenes de manera creixent i, mentre S sigui major que 2, el que es fa es treure els 2 primers elements (els més petits), fusionar-los i afegir la fusió a la llista S, ja que, s'ha d'inserir de manera ordenada per a respectar el criteri. Finalment, quan només quedin 2, (segur que arribarà un moment en que això passi, ja que si S < 3, ja es compleix i si S > 2, estem eliminant 2 cadenes i afegint 1 (la seva fusió), per tant estem disminuint S en 1 a cada iteració, fusionant els 2 primers elements). Finalment, quedaran dos cadenes i el resultat serà la fusió de les dues.

Sigui n totes les cadenes
El cost d'ordenar les n cadenes es de O(nlogn)
El cost de fer el while és, com que a cada iteració el que es fa es reduir S en 1 ja que es treu 2 elements i s'afegeix 1, el nombre d'iteracions seran n. Per cada iteració, extreure i fer el merge és constant, però afegir la cadena resultant de la fusió de les dues primeres te cost logn, per tant, es pot veure que el while serà de O(nlogn).
Finalment, el cost total és O(nlogn) + O(nlogn) = O(nlogn).

Utilitzar aquest criteri és l'òptim ja que:
Siguin X < Y < Z cadenes amb X,Y,Z elements:
Si fem primer:
X + Y i després ho fusionem amb Z: X + Y + Z. Haurem fet 2X + 2Y + Z moviments

Suposem que hi ha una altre manera que és òptima, fent X + Z i després ho fusionem amb Y: X + Z + Y. Haurem fet 2X + 2Z + Y moviments.
Com que hem suposat que la segona és millor, és a dir, té menys moviments:
2X + 2Z + Y < 2X +2Y + Z ⇔ 2Z + Y < 2Y + Z ⇔ Z < Y. Això no és cert segons la suposició de que X < Y < Z ja que Y < Z, per tant no pot ser óptima.

Similarment, si suposem que la solució és óptima fem l'altre possibilitat, fent primer la fusió de Y + Z i després fusionar-la amb Y: Y + Z + X. Obtenim X + 2Y + 2Z. Com que és optima:
X + 2Y + 2Z < 2X + 2Y + Z ⇔ X + 2Z < 2X + Z ⇔ Z < X. Però no és cert degut a la suposició inicial (X < Y < Z).

Com que no hi ha cap altre possibilitat, la millor de les tres es la que utilitza les dos cadenes més petites per fusionar. Si la cadena és més gran que 3, sempre hi hauran combinacions amb X Y i Z els tres primers valors i els altres, valors més grans que Z, i per tant donaran valors més grans, si mirem les combinacions i apliquem el que em vist abans. Llavors, amb el criteri utilitzat, sempre es fusionarà primer les cadenes més petites i serà lo òptim.