

Problemes 1

- 1.1. Suposem que tenim un vector A amb n nombres enters diferents, amb la propietat: existeix un únic índex p tal que els valors $A[1 \dots p]$ estan en ordre creixent i els valors $A[p \dots n]$ estan en ordre decreixent. Per exemple, en la següent vector tenim $n = 10$ i $p = 4$:

$$A = (2, 5, 12, 17, 15, 10, 9, 4, 3, 1)$$

Dissenyeu un algorisme eficient per trobar p donada una matriu A amb la propietat anterior.

Una solució: L'algorisme recursiu es descriu en l'Algorisme FINDPEAK. Donada la matriu A , la resposta s'obté amb una crida amb $i = 1$ i $j = n$. L'algorisme és una cerca binària, en cada pas, comparem els dos elements intermedis i veurem si estem en la part creixent o decreixent. El cas base resol el problema d'obtenir la posició del màxim, però per a una entrada amb mida constant.

```

function FINDPEAK( $A, i, j$ )
     $n = j - i + 1$ 
    if  $n \leq 5$  then
        return PosMAX( $A, i, j$ )
     $k = (i + j)/2$ 
    if  $A[k] < A[k + 1]$  then
        return FINDPEAK( $A, k + 1, j$ )
    else
        return FINDPEAK( $A, i, k$ )
```

Correctesa: Volem trobar l'índex p . Si $A[k] < A[k + 1]$, sabem que $A[i] < \dots < A[k]$ per $i < k$ i podem prescindir de forma segura els elements $A[i \dots k]$. De la mateixa manera, si $A[k] > A[k + 1]$, sabem que $A[k+1] > \dots > A[j]$ per $j > k+1$ i podem descartar amb seguretat els elements $A[k+1 \dots j]$. La posició de p coincideix amb la del valor màxim al vector, per tant el cas base és correcte.

Cost temporal: El cas base té cost constant. A cada pas, es redueix la mida del problema a la meitat i, a més, el cost de les operacions és constant. Així, tenim la recurrència $T(n) = T(n/2) + c$ per a alguna constant c . Sabem que, com a la cerca binària, $T(n) = O(\log n)$.

1.2. Un vector $A[n]$ conté tots els enters entre 0 i n excepte un.

- Dissenyeu un algorisme que, utilitzant un vector auxiliar $B[n + 1]$, detecti l'enter que no és a A , i ho faci en $O(n)$ passos.
- Suposem ara que $n = 2^k - 1$ per a $k \in \mathbb{N}$ i que els enters a A venen donats per la seva representació binària. En aquest cas, l'accés a cada enter no és constant, i llegir qualsevol enter té un cost $\lg n$. L'única operació que podem fer en temps constant es “recuperar” el j -èsim bit de l'enter a $A[i]$. Dissenyeu un algorisme que, utilitzant la representació binària per a cada enter, trobi l'enter que no és a A en $O(n)$ passos.

a) Como A tiene todos los valores menores que uno, en B solo queda una posición sin marcar, la del número que falta.

1 recorrido de A y 1 recorrido de $B \rightarrow$ coste $O(m)$

b) Para cada dígito, si en total hay más 0s que 1s el número que falta tiene un 1. Si no al revés.
Correcto ya que $n = 2^k - 1$ Y en total son todos los números con k bits.

$$\text{coste} = O(k \cdot m)$$

generar una lista que enlace solo los valores donde sigue faltando un valor.

Cada lista es la mitad de la anterior.

$$T(m) = T(m/2) + O(m)$$

$$T(m) \approx O(m)$$

1.3. El coeficient de Gini és una mesura de la desigualtat ideada per l'estadístic italià Corrado Gini. Normalment s'utilitza per mesurar la desigualtat en els ingressos, dins d'un país, però pot utilitzar-se per mesurar qualsevol forma de distribució desigual. El coeficient de Gini és un nombre entre 0 i 1, on 0 es correspon amb la perfecta igualtat (tots tenen els mateixos ingressos) i on el valor 1 es correspon amb la perfecta desigualtat (una persona té tots els ingressos i els altres cap).

Formalment, si $r = (r_1, \dots, r_n)$, amb $n > 1$, és un vector de valors no negatius, el *coeficient de Gini* es defineix com:

$$G(r) = \frac{\sum_{i=1}^n \sum_{j=1}^n |r_i - r_j|}{2(n-1) \sum_{i=1}^n r_i}.$$

Proporcioneu un algorisme eficient per calcular el coeficient de Gini donat el vector r .

$$S = 0 \quad n = 0$$

for $i = 1, \dots, m$

$$n += r_i$$

for $j = 1, \dots, m$

$$S += |r_i - r_j|$$

$$\text{return } (S / n * ((m-1) * m))$$

$$\Theta(m^2)$$

Ordenar \rightarrow coste $O(m \log(m))$

$$r_1 > r_2 > r_3 > \dots > r_m$$

r_i aparece sumando $m-i$

aparece restando $i-1$

$$\sum_{i=1}^m (m-i - (i-1)) * r_i$$

numerosa

→ Ordenar los valores en orden creciente

$$n = 0$$

for $i = 1, \dots, m$

$$n += (n+2 * i + 1) * r_i$$

$$S += r_i$$

$$\text{return } (n / ((m-1) * S))$$

1.4. Per a cadascú dels algorismes, digueu quin és el temps en cas pitjor, quan l'entrada és un enter positiu $n > 0$.

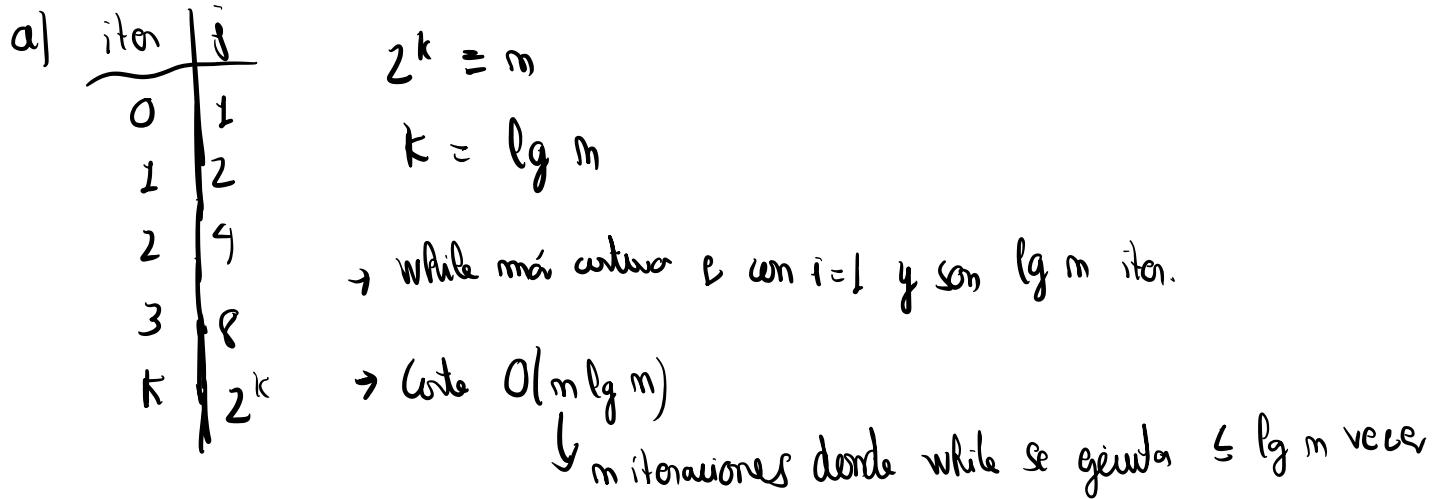
- (a)

```
for i = 1 to n do
    j = i
    while j < n do
        j = 2 * j
```
- (b)

```
for i = 1 to n do
    j = n
    while i * i < j do
        j = j - 1
```
- (c)

```
for i = 1 to n do
    j = 2
    while j < i do
        j = j * j
```
- (d)

```
i = 2
while (i * i < n) i (n mod i ≠ 0) do
    i = i + 1
```



$m - \sqrt{m}$
 θ) (a)
 θ) si $i \geq \sqrt{m} \rightarrow$ no entra en el while

$\sqrt{m} + \text{si } i < \sqrt{m} \rightarrow$ el caso peor q. $i=1 \rightarrow O(m)$ it. while
 θ)

$$O(\underbrace{m - \sqrt{m}}_{\alpha} + \underbrace{\sqrt{m}}_{\beta} m)$$

c) Case von mehr it. while $\& i = m$

it	j
0	2
1	$4 \rightarrow 2 \cdot 2 = 2^2$
2	$16 \rightarrow 2^2 \cdot 2^2 = 2^4$
3	$2^4 \cdot 2^4 = 2^8$
k	2^{2^k}

$$2^{2^k} = m$$

$$\lg(2^{2^k}) = \lg m$$

$$2^k = \lg m$$

$$\lg(2^k) = \lg \lg m$$

$$k = \lg \lg m$$

1.5. El problema 2SAT té com a entrada un conjunt de clàusules, on cada clàusula és la disjunció (OR) de dos literals (un literal és una variable booleana o la negació d'una variable booleana). Volem trobar una manera d'assignar un valor cert o fals a cadascuna de les variables perquè totes les clàusules es satisfagin - és a dir, hi hagi al menys almenys un literal cert a cada clàusula. Per exemple, aquí teniu una instància de 2SAT:

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_1 \vee x_4).$$

Aquesta instància és satisfactible: fent x_1, x_2, x_3, x_4 cert, fals, fals i cert, respectivament.

El propòsit d'aquest problema és conduir-vos a una manera de resoldre 2SAT de manera eficient reduint-ho al problema de trobar les components connexes fortes d'un graf dirigit. Donada una instància F de 2SAT amb n variables i m clàusules, construïm un graf dirigit $G_F = (V, E)$ de la següent manera.

- G_F té $2n$ nodes, un per a cada variable i un per a la seva negació.
- G_F té $2m$ arcs: per a cada clàusula $(\alpha \vee \beta)$ de F (on α, β són literals), G_F té un arc des de la negació d' α a β , i un de la negació de β a α .

Tingueu en compte que la clàusula $(\alpha \vee \beta)$ és equivalent a qualsevol de les implicacions $\neg\alpha \Rightarrow \beta$ o $\neg\beta \Rightarrow \alpha$. En aquest sentit, G_F representa totes les implicacions directes en F .

- (a) Realitzeu aquesta construcció per a la instància de 2SAT indicada amunt.
- (b) Demostreu que si G_F té una component connexa forta que conté x i $\neg x$ per a alguna variable x , llavors no és satisfactible.
- (c) Ara demostreu la inversa: és a dir, que si no hi ha cap component connexa forta que contingui tant un literal com la seva negació, llavors la instància ha de ser satisfactible.
- (d) A la vista del resultat previ, hi ha un algorisme de temps lineal per resoldre 2SAT?

- 1.6. En una festa, un convidat es diu que és una celebritat si tothom el coneix, però ell no coneix a ningú (tret d'ell mateix). Les relacions de coneixença donen lloc a un graf dirigit: cada convidat és un vèrtex, i hi ha un arc entre u i v si u coneix a v . Doneu un algorisme que, donat un graf dirigit representat amb una matriu d'adjacència, indica si hi ha o no cap celebritat. En el cas que hi sigui, cal dir qui és. El vostre algorisme ha de funcionar en temps $O(n)$, on n és el nombre de vèrtexs.

1.7. Llisteu les següents funcions en ordre *creixent*, és a dir, si l'ordre és $f_1; f_2; \dots$, aleshores $f_2 = \Omega(f_1); f_3 = \Omega(f_2)$; etc.

$$(\log n)^{100}, n \log n, 3^n, \frac{n^2}{\log n}, n2^n, 0.99^n, n^3, \sqrt{n}.$$

1.8. Digueu si cadascuna de les afirmacions següents són certes o falses (i per què).

- (a) Asimptòticament $(1 + o(1))^{\omega(1)} = 1$
- (b) Si $f(n) = (n+2)n/2$ aleshores $f(n) \in \Theta(n^2)$.
- (c) Si $f(n) = (n+2)n/2$ aleshores $f(n) \in \Theta(n^3)$.
- (d) $n^{1.1} \in O(n(\lg n)^2)$
- (e) $n^{0.01} \in \omega((\lg n)^2)$

a)

$$\begin{aligned} 1 + o(1)^{\omega(1)} &= 1 \\ 1 + o(1)^{\omega} &= 1 \\ 1 + o(1)^{\omega} &= 1 \\ o(1)^{\omega} &= 1 \end{aligned}$$

\rightarrow No es

Exemple \rightarrow $f \in o(1) \quad \lim_{m \rightarrow \infty} f(m) = 0$
 $g \in \omega(1) \quad \lim_{m \rightarrow \infty} g(m) = \infty$

$$\lim (1 + f)^{\omega} \rightarrow \text{ind} \quad 1^{\infty}$$

$$\boxed{\lim \left(1 + \frac{1}{m}\right)^m = e} \rightarrow \text{falso}$$

b) $f(m) = (m+2) \cdot \frac{m}{2} \rightarrow f(m) \in \Theta(m^2)$

$$\lim_{m \rightarrow \infty} \frac{(m+2) \cdot \frac{m}{2}}{m^2} = \lim_{m \rightarrow \infty} \frac{\frac{m^2}{2} + m}{m^2} \rightarrow \frac{1}{2} + 0 =$$

$$= \lim_{m \rightarrow \infty} \left(\frac{1}{2} + \frac{1}{m} \right) = \frac{1}{2} + 0$$

c) Amb exercici 6

d) $m^{1.1} \leq c \cdot m (\log m)^2$

$$m^{0.1} \leq c \cdot (\log m)^2$$

$$m^{\frac{1}{10}} \leq c \cdot (\log m)^2$$

$$m \leq c^{10} \cdot (\log m)^{20}$$

Falso porque los polinomios crecen + rápido
que los pd. en logaritmos

e) $\lim_{m \rightarrow \infty} \frac{(\lg m)^c}{m} = \lim_{m \rightarrow \infty} c \lg m^{c-1} \cdot \frac{1}{m}$

e) $m^{0.01} \in W((\log(m))^2)$

$$\lim_{m \rightarrow \infty} \frac{m^{0.01}}{\log(m)^2} = \lim_{m \rightarrow \infty} \frac{\sqrt[100]{m}}{\log(m)^2} = \lim_{m \rightarrow \infty} \frac{m}{\log(m)^{200}} = \infty \rightarrow m^{0.01} \in W(\log(m)^2)$$

1.9. Digueu si la següent demostració de

$$\sum_{k=1}^n k = O(n)$$

és certa o no (i justifiqueu la vostra resposta).

Demostració: Per a $k = 1$, tenim $\sum_{k=1}^1 k = 1 = O(1)$. Per hipòtesi inductiva, assumim $\sum_{k=1}^n k = O(n)$, per a una certa $n > 1$. Llavors, per a $n + 1$ tenim

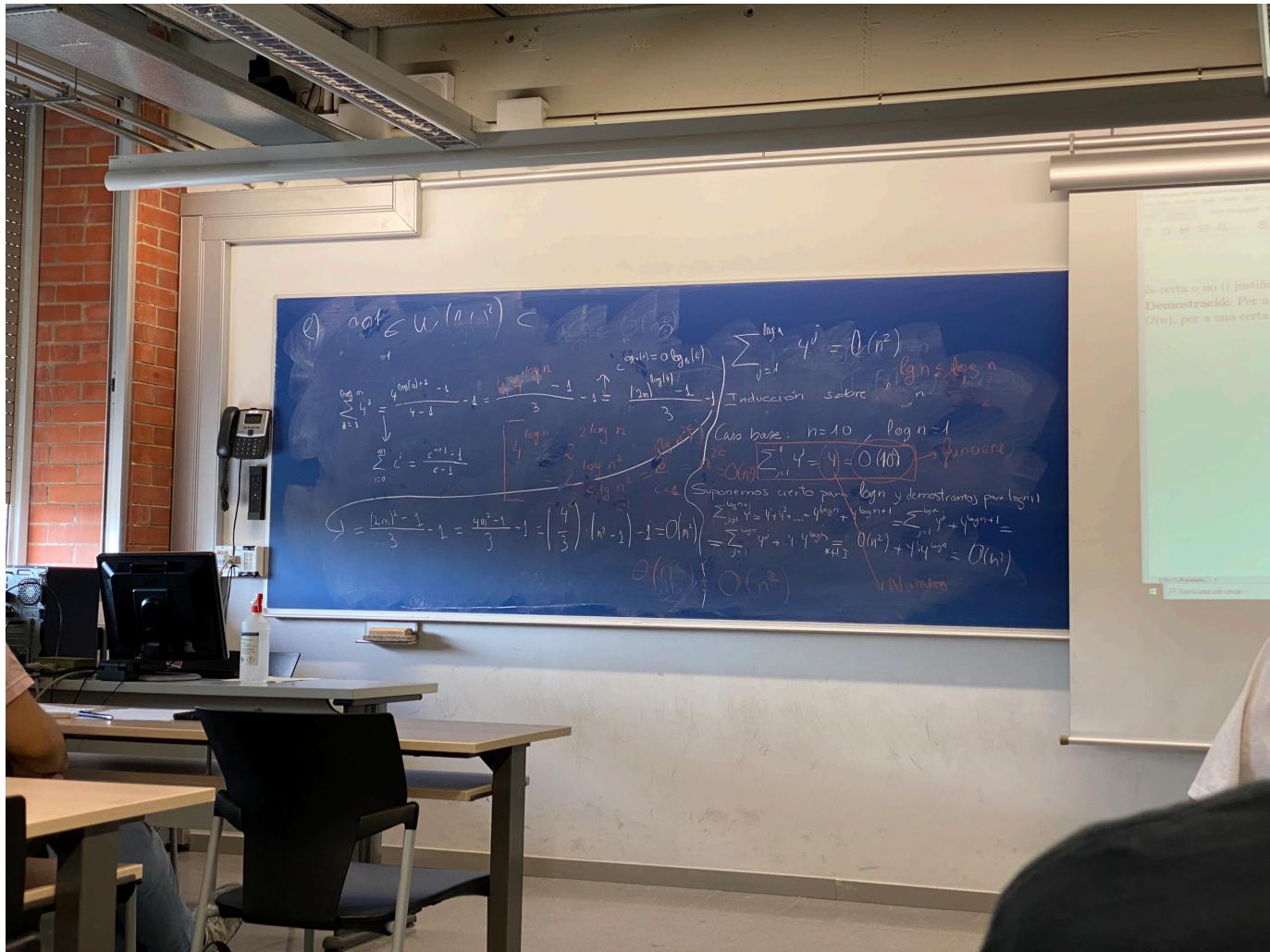
$$\sum_{k=1}^{n+1} k = n + 1 + \sum_{k=1}^n k = n + 1 + O(n) = O(n).$$

$$4^{\log n} = 2^{2 \log n} \\ = 2^{\log n^2}$$

1.10. Demostre que $\sum_{j=1}^{\log n} 4^j = O(n^2)$.

$$\sum_{j=1}^{\log n} 4^j = \frac{4^{\log(n)+1} - 1}{4 - 1} = \frac{4^{\log(2 \cdot m)} - 1}{3} = \frac{(2m)^{\log 4} - 1}{3} = \frac{c^{m+1} - 1}{c - 1}$$

$$\hookrightarrow = \frac{(2m)^2 - 1}{3} = \frac{4m^2 - 1}{3} = \left(\frac{4}{3}\right) \cdot (m^2 - 1) = O(m^2)$$



1.11. Donat el següent algorisme per conduir un robot, que té com a entrada un enter no negatiu n ,

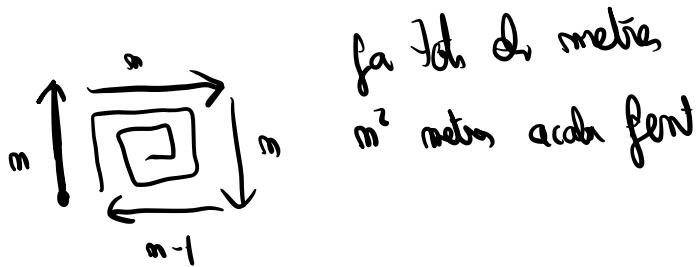
Funció CAMINAR-(n)
 si $n \leq 1$ retornar i aturar-se
 caminar nord n metres
 caminar est n metres
 caminar sud n metres
 caminar oest $n - 1$ metres
 caminar nord 1 metre
 CAMINAR ($n - 2$)

Sigui $C(n)$ el nombre de metres que el robot camina quan executem l'algorisme amb paràmetre n . Doneu una estimació asimptòtica del valor de $C(n)$.

En cada curs de la funció $m + m + m + m - 1 + 1 = 4m$

$$C(m) = C(m-2) + O(4m) \xrightarrow{\text{Subtract and conquer theorem}} \rightarrow C(m) = O(m^2)$$

Altres formes:



- 1.12. Tenim un conjunt de robots que es mouen en un edifici, cadascun d'ells és equipat amb un transmissor de ràdio. El robot pot utilitzar el transmissor per comunicar-se amb una estació base. No obstant això, si els robots són massa a prop un de l'altre hi ha problemes amb la interferència entre els transmissors. Volem trobar un pla de moviment dels robots, de manera que puguin procedir al seu destí final, sense perdre mai el contacte amb l'estació base.

Podem modelar aquest problema de la següent manera. Se'n dóna un graf $G = (V, E)$ que representa el plànor d'un edifici, hi ha dos robots que inicialment es troben en els nodes a i b . El robot en el node a voli viatjar a la posició c , i el robot en el node b vol viatjar a la posició d . Això s'aconsegueix per mitjà d'una planificació: a cada pas de temps, el programa especifica que un dels robots es mou travessant una aresta. Al final de la planificació, els dos robots han d'estar en les seves destinacions finals.

Una planificació és *lliure* d'interferència si no hi ha un punt de temps en el qual els dos robots ocupen nodes que es troben a distància menor de r , per a un valor determinat del paràmetre r .

Doneu un algorisme de temps polinomial que decideixi si hi ha una planificació lliure donats, el graf, les posicions inicials i finals dels robots i el valor de r .

Una solució. Per resoldre el problema considerarem l'espai de configuracions on els robots es poden moure. És a dir, el conjunt de parells de posicions que estan a distància més gran o igual que r :

$$C = \{(u, v) \mid u, v \in V \text{ i } d(u, v) \geq r\}$$

Podem considerar la relació entre configuracions definida pels moviments permesos. Així tenim

$$\begin{aligned} M = \{((u, v), (u', v')) \mid & (u, v), (u', v') \in C \text{ i} \\ & ((u = u' \text{ i } v, v') \in E) \text{ o } (v = v' \text{ i } (u, u') \in E)\}. \end{aligned}$$

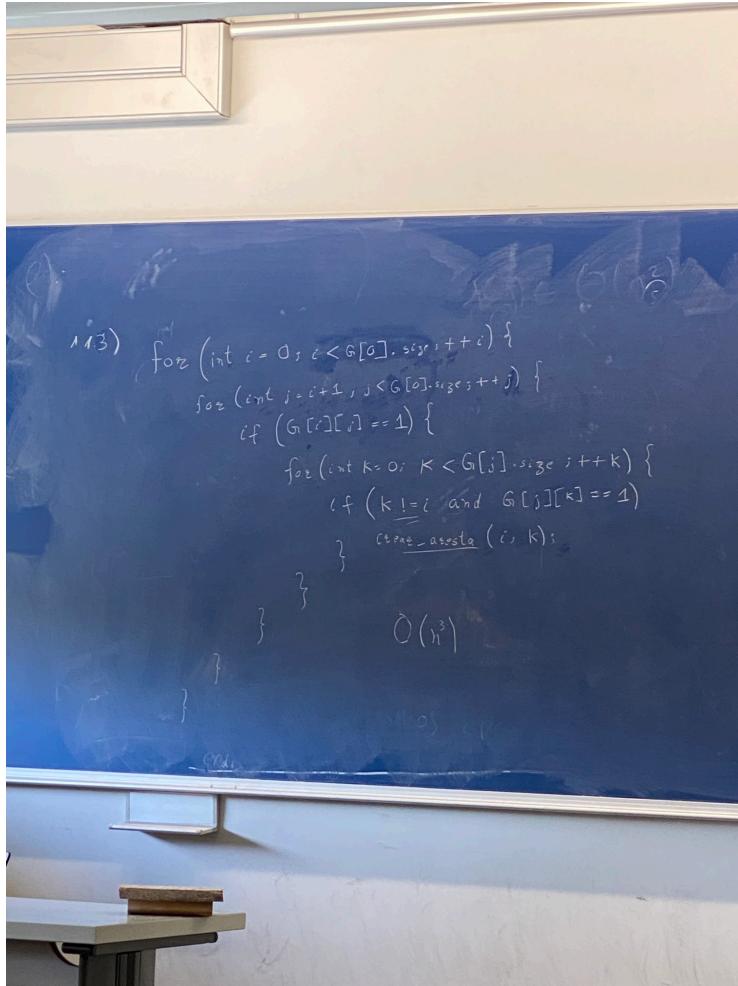
A l'espai de configuracions podem considerar el graf $\mathcal{G} = (C, M)$ on dos configuracions son veïnes si i només si un del robots pot canviar de posició sense interferir amb la posició de l'altre.

Els robots són inicialment a la configuració (a, b) i s'han de desplaçar amb moviments vàlids fins a la configuració (c, d) . Això serà possible únicament si hi ha un camí de (a, b) a (c, d) . D'acord amb el raonament anterior tenim que comprovar hi ha un camí entre dos vèrtexs a \mathcal{G} . Podem detectar-ho amb un BFS en temps $O(|C| + |M|)$.

Per calcular el cost hem de tenir en compte la mida de l'entrada. Si $G = (V, E)$ i $n = |V|$ i $m = |E|$, tenim $|C| \leq n^2$ i $|M| \leq 2m$. Suposant que ens donen G mitjançant llistes d'adjacència la mida de l'entrada és $n + m$. Construir una descripció de \mathcal{G} mitjançant llistes d'adjacència té cost $O(n^2 + m)$. Fer un BSF sobre \mathcal{G} té cost $O(n^2 + m)$. El cost total es $O(n^2 + m)$ però $m \leq n^2$. Llavors l'algorisme proposat té cost $O(n^2)$.

1.13. El quadrat d'un graf $G = (V, E)$ és un altre graf $G' = (V, E')$ on $E' = \{(u, v) | \exists w \in V, (u, w) \in E \wedge (w, v) \in E\}$. Dissenyeu i analitzeu un algorisme que, donat un graf representat amb una matriu d'adjacència, calculi el seu quadrat. Feu el mateix amb un graf representat amb llistes d'adjacència.

metre e demarrer en Trunge



anadir a la lista de i todos
los vecinos de j que no sean i
anadir a la lista de j los de i que no sean
i.

1.14. Es diu que un vèrtex d'un graf connex és un *punt d'articulació* del mateix si, en suprimir aquest vèrtex i totes les arestes que hi incideixen, el graf resultant deixa de ser connex. Per exemple, un graf en forma d'anell no té cap punt d'articulació mentre que tot node que no sigui fulla d'un arbre és punt d'articulació. Dissenyeu un algorisme que, donat un graf connex no dirigit $G = (V, E)$, indiqui quins vèrtexos del graf són punts d'articulació. Calculeu el seu cost. Indiqueu quina implementació del graf és la que proporciona un algorisme més eficient.

$G(V, E)$

verticec corte (s, d)

visitador [s] = true

profundidad = d

low = d

para todo $k \in \text{Adj}(s)$ hacen

if visited [k] == false then

verticec corte ($k, d+1$),

low [k] = min (low [s], low [k]);

if low [k] \geq profundidad [s] + then

if s = mo en raiz OR m^e hijos (s) > 1 then

s es vertice de corte

endif

endif

else if padre [s] $\neq k$ then

low [s] = min (low [s], profundidad [k])

endif

- 1.15. Un graf dirigit és *fortament connex* quan, per cada parell de vèrtexs u, v , hi ha un camí de u a v . Doneu un algorisme per determinar si un graf dirigit és fortament connex. Quin és l'algorisme més ràpid que coneixeu per obtenir les components connexes fortes d'un graf?

G pro arriba orientada al revés

para $u \in V$

para $v \in N(u)$

poner u como vecino de v en $\overset{\leftarrow}{G}$

$$O(m + mn)$$

- 1.16. Un graf dirigit $G = (V, E)$ és *semiconnex* si, per qualsevol parell de vèrtexs $u, v \in V$, tenim un camí dirigit de u a v o de v a u . Doneu un algorisme eficient per determinar si un graf dirigit G és semiconnex. Demostreu la correctesa del vostre algorisme i analitzeu-ne el cost. Dissenyeu el vostre algorisme fent us d'un algorisme que us proporcioni les components connexes fortes del graf en temps $O(n + m)$.

1.17. Definim els *k-mers* com les subcadenes de DNA, amb grandària k . Per tant, per a un valor donat k podem assumir que tenim una base de dades amb tots els 4^k *k-mers*. Una manera utilitzada en l'experimentació clínica per a identificar noves seqüències de DNA, consisteix a agafar mostres aleatòries de una cadena i determinar quins *k-mers* conté, on els *k-mers* es poden solapar. A partir d'aquest procés, podem reconstruir tota la seqüència de DNA.

Volem resoldre un problema més senzill. Donada una cadena $w \in \{A, C, T, G\}^*$, i un enter k , sigui $S(w)$ el multi-conjunt de tots els *k-mers* que apareixen a w . Notem que $|S(w)| = |w| - k + 1$. *Donat un multi-conjunt C de k-mers, trobar, si n'hi ha, la cadena de DNA w tal que S(w) = C.*

Trobeu un algorisme per resoldre aquest problema i analitzeu la seva complexitat.

(Ajut: A partir de qualsevol entrada al problema de la seqüenciació, hem de construir en temps polinòmic un graf dirigit G que sigui entrada al problema del camí Eulerià, i tal que existeixi una solució al problema de la seqüenciació si G té un camí Eulerià. Podeu considerar com a vèrtexs els $k - 1$ -mers que apareixen en el multiconjunt. Heu de decidir com definir les arestes (\vec{u}, v) i demostrar la correctesa de la vostra construcció.)

1.18. El Professor JD ha corregit els exàmens finals del curs, de cara a tenir una distribució maca de les notes finals decideix formar k grups, cada grup amb el mateix nombre d'alumnes, i donar la mateixa nota a tots els alumnes que són al mateix grup. La condició més important és que qualsevol dels alumnes al grup i han de tenir nota d'examen superior a qualsevol alumne d'un grup inferior (grups de 1 fins a $i - 1$). L'ordre dintre de cadascun dels grups es irrelevant. Dissenyeu un algorisme que donada una taula A no ordenada, que a cada registre conté la identificació d'un estudiant amb la seva notes d'examen, divideix A en els k grups, amb les propietat descrita a dalt. El vostre algorisme ha de funcionar en temps $O(n \lg k)$. Al vostre anàlisis podeu suposar que n és múltiple de k i k és una potència de 2.

Una solució: Sigui AGRUPAR l'algorisme recursiu que, té com a entrada una taula de alumnes-notes N i dos enters ℓ i t , i fa el següent:

- Mentre $\ell \neq 1$ troba la mediana de A i fa una partició al seu voltant en temps $O(|N|)$.
- Considerem la sub-taula N_e esquerra i la sub-taula dreta N_d .
- Cridem recursivament
AGRUPAR($N_e, \ell/2, 2t$) i AGRUPAR($N_d, \ell/2, 2t + 1$).
- Quan $\ell = 1$, la taula constitueix la partició t .

La crida inicial la farem amb N , $\ell = k$ i $t = 0$. La correctesa ve de com particionem els elements. Sempre tenim dos meitats i els elements a N_e són més petits que la mediana i els elements a N_d són més grans o iguals que la mediana. Aconseguirem $\ell = 1$ després de $\lg k$ iteracions, en aquell moment la taula té n/k elements. La variable t comptabilitza l'ordre de les crides. Al primer nivell tenim només una taula i $t = 0$. Al segon tindrem dos taules, la de l'esquerra etiquetada amb 0 i la de la dreta amb 1. Al següent nivell, tindrem 0,1,2,3 (e-e,e-d,d-e,d-d). Llavors t comptabilitza l'ordre correcte de les particions per garantir la propietat requerida.

El cost de l'algorisme és $T(n, k) = 2T(n/2, k/2) + \Theta(n)$ amb $T(n, 1) = \Theta(1)$, per a tot n . Desplegant la recursió tenim

$$\begin{aligned} T(n, k) &= 2T(n/2, k/2) + cn = 4T(n/4, k/4) + 2c(n/2) + cn \\ &= 4T(n/4, k/4) + 2cn = k + cn \lg k. \end{aligned}$$

llavors, $T(n) = \Theta(n \lg k)$.

1.19. Resoleu les següents recurrències

- (a) $T(n) = 16T(n/2) + \binom{n}{3} \lg^4 n$
- (b) $T(n) = 5T(n/2) + \sqrt{n}$
- (c) $T(n) = 2T(n/4) + 6.046\sqrt{n}$
- (d) $T(n) = 2T(n/2) + \frac{n}{\lg n}$
- (e) $T(n) = T(n - 10) + n$

- 1.20. Donat un graf no dirigit $G = (V, E)$ i un subconjunt de vèrtex V_1 , el subgraf induït per V_1 , $G[V_1]$ té com a vèrtex V_1 i con aarestes totes lesarestes a E que connecten vèrtexs en V_1 . Un clique és un subgraf indiut per un conjunt C on tots els vèrtexs estan connectats entre ells.

Considereu el següent algorisme de dividir-i-vèncer per al problema de *trobar un clique* en un graf no dirigit $G = (V, A)$.

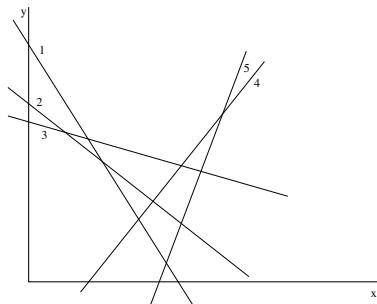
CliqueDV(G)

- 1: Enumereu els vèrtexs V com $1, 2, \dots, n$, on $n = |V|$
- 2: Si $n = 1$ tornar V
- 3: Dividir V en $V_1 = \{1, 2, \dots, \lfloor n/2 \rfloor\}$ i $V_2 = \{\lfloor n/2 \rfloor + 1, \dots, n\}$
- 4: Sigui $G_1 = G[V_1]$ i $G_2 = G[V_2]$
- 5: $C_1 = \text{CliqueDV}(G_1)$ i $C_2 = \text{CliqueDV}(G_2)$
- 6: $C_1^+ = C_1$ i $C_2^+ = C_2$
- 7: **for** $u \in C_1$ **do**
- 8: **if** u està connectat a tots els vèrtexs a C_2^+ **then**
- 9: $C_2^+ = C_2^+ \cup \{u\}$
- 10: **for** $u \in C_2$ **do**
- 11: **if** u està connectat a tots els vèrtexs a C_1^+ **then**
- 12: $C_1^+ = C_1^+ \cup \{u\}$
- 13: Retorneu el més gran d'entre C_1^+ i C_2^+

Contesteu les següents preguntes:

- (a) Demostreu que l'algorisme **CliqueDV** sempre retorna un subgraf de G que és un clique.
- (b) Doneu una expressió asimptòtica del nombre de passos de l'algorisme **CliqueDV**.
- (c) Doneu un exemple d'un graf G on l'algorisme **CliqueDV** retorna un clique que no és de grandària màxima.
- (d) Creieu que és fàcil modificar **CliqueDV** de manera que sempre done el clique màxim, sense incrementar el temps pitjor de l'algorisme? Expliqueu la vostra resposta.

1.21. El problema de l'eliminació de superfícies ocultes és un problema important en informàtica gràfica. Si des de la teva perspectiva, en Pepet està davant d'en Ramonet, podràs veure en Pepet però no en Ramonet. Considereu el següent problema, restringit al pla. Us donen n rectes no verticals al pla, L_1, \dots, L_n , on la recta L_i ve especificada per l'equació $y = a_i x + b_i$. Assumim, que no hi han tres rectes que es creuen exactament al mateix punt. Direm que L_i és *maximal* en x_0 de la coordenada x , si per qualsevol $1 \leq j \leq n$ amb $j \neq i$ tenim que $a_i x_0 + b_i > a_j x_0 + b_j$. Direm que L_i és *visible* si té algun punt maximal.



Donat com a entrada un conjunt de n rectes $\mathcal{L} = \{L_1, \dots, L_n\}$, doneu un algorisme que, en temps $O(n \lg n)$, torne les rectes no visibles. A la figura de sobre teniu un exemple amb $\mathcal{L} = \{1, 2, 3, 4, 5\}$. Totes les rectes excepte la 2 són visibles (considerem rectes infinites).

La recta s'ordenen per pendent

③ Mira interseccions

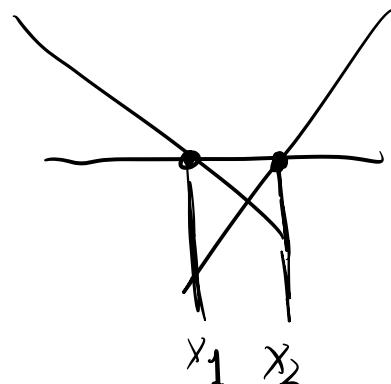
② Mira paral·lelos

④ merge([mitad1, mitad2])

$O(m)$ junts

① Return rectas

Dempués restar las visibles



$O(m)$ punto intersección



1.22. Suposeu que sou consultors per a un banc que està molt amoïnat amb el tema de la detecció de fraus. El banc ha confiscat n targetes de crèdit que se sospita han estat utilitzades en negocis fraudulents. Cada targeta conté una banda magnètica amb dades encriptades, entre elles el número del compte bancari on es carrega la targeta. Cada targeta es carrega a un únic compte bancari, però un mateix compte pot tenir moltes targetes. Direm que dues targetes són *equivalents* si corresponen al mateix compte.

És molt difícil de llegir directament el número de compte d'una targeta intel·ligent, però el banc té una tecnologia que donades dues targetes permet determinar si són equivalents.

La qüestió que el banc vol resoldre és la següent: donades les n targetes, volen conèixer si hi ha un conjunt on més de $\lceil n/2 \rceil$ targetes són totes equivalents entre si. Suposem que les úniques operacions possibles que pot fer amb les targetes és connectar-les de dues en dues, al sistema que comprova si són equivalents.

Doneu un algorisme que resolgui el problema utilitzant només $O(n \lg n)$ comprovacions d'equivalència entre targetes. Sabréu com fer-ho en temps lineal?



se hacen parejas

Si son iguales me quedo con una

si son distintas la tiro

Si hay una desemparejada me la quedo

- 1.23. Donada una taula A amb n registres, on cada registre conté un enter de valor entre 0 i 2^n , i els continguts de la taula estan desordenats, dissenyeu un algorisme lineal per a obtenir una llista ordenada dels elements a A que tenen valor més gran que els $\log n$ elements més petits a A , i al mateix temps, tenen valor més petit que els $n - 3 \log n$ elements més grans a A .

$O(n \cdot n)$

~~$O(n)$~~ Ordenar $1 \times - \text{elen } \lg n \text{ e y elem. } 3 \log n$

$\sim O(n \cdot n)$

~~$O(n)$~~ Ordenar elem z $x \leq z \leq y \rightarrow S$

$O(\lg n \log \lg n)$ Ordenar S

Entrada son n valors con m bits

\rightarrow Comparar $\in O(m)$

$$|x| = m^2$$

$$\text{Cate} \rightarrow O((m + \log m \log \log m) m) = O(m^2)$$

1.24. Tenim un taula T amb n claus (no necessàriament numèriques) que pertanyen a un conjunt totalment ordenat. Doneu un algorisme $O(n + k \log k)$ per a ordenar els k elements a T que són els més petits d'entre els més grans que la mediana de les claus a T .

① Obtenir element x que ocupa $\text{pos}\left[\frac{n+1}{2}\right] \quad O(n)$

② Na quedar-nos S , tots els que són $\geq x \quad O(n)$

③ Obtenir el k -èsim element de S

④ Na quedar-nos amb els k elements $\leq y$

⑤ Ordenar estos k elements amb Merge Sort $O(k \lg k)$

Coste $O(n+k + \lg k)$

1.25. Com ordenar eficientment elements de longitud variable:

- (a) Donada una taula d'enters, on els enters emmagatzemats poden tenir diferent nombre de díigits. Però sabem que el nombre total de díigits sobre tots els enters de la matriu és n . Mostreu com ordenar la matriu en $O(n)$ passos.
- (b) Se us proporciona una sèrie de cadenes de caràcters, on les diferents cadenes poden tenir diferent nombre de caràcters. Com en el cas previ, el nombre total de caràcters sobre totes les cadenes és n . Mostreu com ordenar les cadenes en ordre alfabètic fent servir $O(n)$ passos. (Tingueu en compte que l'ordre desitjat és l'ordre alfabètic estàndard, per exemple, $a < ab < b$.)

Q)

```
void countSort(string a[], int size, int b[]){
    string *b = NULL, int *c = NULL;
    b = new string[size];
    c = new int[257];
    initializing count array
    for (int i = 0; i < 257; i++){ → O(257)
        c[i] = 0;
        //cout << c[i] << endl;
    } count each character
    for (int j = 0; j < size; j++){ → O(a)
        if (k < a[j].size()) {
            c[(int)(unsigned char)a[j][k] + 1]++;
        }
        else {
            c[0]++;
        }
        //a[j] is a string
        //cout << c[a[j]] << endl;
    } sum all counts of characters
    for (int f = 1; f < 257; f++){ → O(257)
        c[f] += c[f - 1];
    } in f sorted arrays
    for (int r = size - 1; r >= 0; r--){ → O(a)
        if (k < a[r].size()) {
            b[c[(int)(unsigned char)a[r][k] + 1] - 1] = a[r];
            c[(int)(unsigned char)a[r][k] + 1]--;
        }
        else {
            b[c[0] - 1] = a[r];
            c[0]--;
        }
    } a = f
    for (int l = 0; l < size; l++){ → O(a)
        a[l] = b[l];
    }
    // avoid memory leak
    delete[] b;
    delete[] c;
}
```

int k = std::max(b.size(), n); $\rightarrow O(a)$
 $\downarrow \geq 0; \forall i \in [0, n]$ // size_t is unsigned, so avoid using digit < 0, which is always zero, n); $\rightarrow O(k * (3 * 257 + 3a)) = O(k * a)$

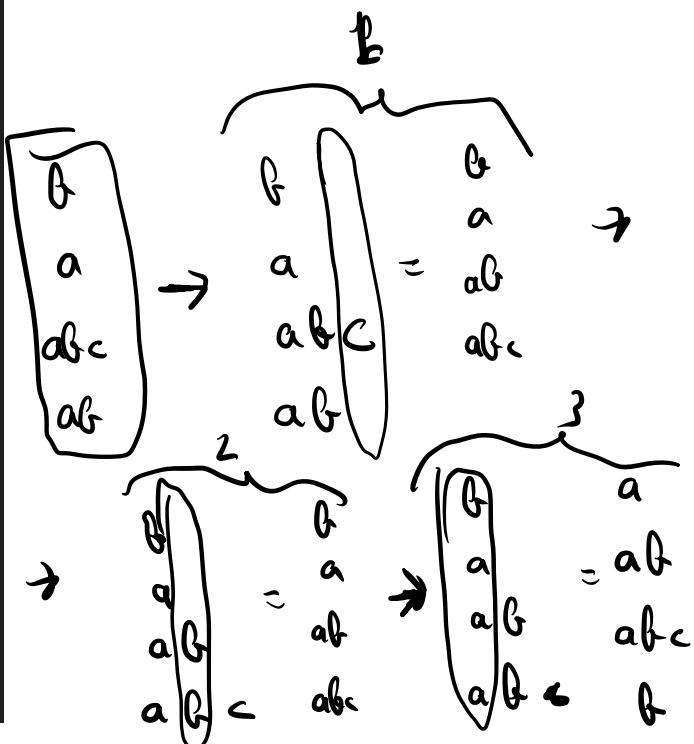
$a =$ nombre total de cadenes

$k =$ digits de la cadena amb més digits

$$k \leq m \quad a \leq m$$

$$\sum_{i=1}^a k = m = a \cdot k$$

R: Radix por la izquierda



a)

1.26. Hi ha un concurs de TV amb n participants on cada participant escull un enter entre 0 i 1000000. El premi és per als dos concursants que escullen els enters més propers. Dissenyeu un algorisme que, en temps lineal, li digui al presentador quins són els dos concursants guanyadors o l'indiqui que hi ha més d'un parell de concursants candidats a rebre el premi.

1.27.  Donat un vector A amb n elements, és possible posar en ordre creixent els \sqrt{n} elements més petits i fer-ho en $O(n)$ passos?

Una solució: Seleccionar l'element \sqrt{n} -èsim i particionar al voltant, d'aquest element (cost $O(n)$). Ordenar la part esquerra en $O(\sqrt{n}^2)$.

Alternativament, construir un min-heap en $O(n)$ i extreure el mínim element \sqrt{n} cops, el nombre de passos és $O(n + \sqrt{n} \lg n) = O(n)$.

1.28. Tenim un vector $A = (a_1, \dots, a_n)$ d'elements d'un conjunt sobre els quals s'ha definit una relació d'ordre, i un vector $R = (r_1, \dots, r_p)$ d'enters i ordenat, amb $1 \leq r_1 < r_2 < \dots < r_p \leq n$.

Proporcioneu un algorisme que, donats A i R , i amb cost $o(pn)$, trobi l' r_1 -èsim, r_2 -èsim, \dots , r_p -èsim del vector A . Justifiqueu la correctesa de l'algorisme proposat i el seu cost en funció de n i de p .

Solució:

El problema es pot resoldre trivialment ordenant tot el vector A (cost d'aquesta solució: $\Theta(n \log n + p)$) o bé invocant p vegades un algorisme de selecció amb cost lineal en cas pitjor, per a obtenir l' r_1 -èsim, r_2 -èsim, etc. successivament (cost d'aquesta solució: $\Theta(n \cdot p)$). Cap d'aquestes solucions es considera vàlida ja que no satisfà el requeriment de ser $o(p \cdot n)$.

Un refinament és aplicar l'algorisme de selecció per trobar l' r_{i+1} -èsim en el subvector $A[r_i..n]$ donat que el pas previ ha particionat A . Això és una petita millora, però insuficient. Per exemple si $r_i \approx i \cdot (n/p)$ llavors el cost de l'algorisme és

$$\begin{aligned} S(n, p) &= \sum_{i=1}^p \Theta(n - i \cdot n/p) \\ &= \Theta(n/p) \sum_{i=1}^p \Theta(p - i) = \Theta((n/p) \cdot p^2/2) = \Theta(p \cdot n). \end{aligned}$$

Un possible solució és fer un algorisme **MULTISELECT** recursiu tal que si $p > 1$ agafa el rang central $r_{p/2}$, el selecciona fent servir un algorisme de selecció en temps lineal i a continuació es fan dues crides recursives a **MULTISELECT**: una per buscar els rangs $(r_1, \dots, r_{p/2-1})$ en el subvector $A[1..r_{p/2}-1]$ i l'altra per buscar els rangs $(r_{p/2+1}, \dots, r_p)$ en el subvector $A[r_{p/2}+1..n]$. La recurrència del cost per a quest algorisme seria

$$M(n, p) = \Theta(n) + M(n', p/2) + M(n'', p/2),$$

on $n' = r_{p/2} - 1$ i $n'' = n - r_{p/2}$. No cal resoldre-la. Considerem l'arbre de crides recursives. És un arbre quasi-complet, a l'arrel comencem amb p rangs, a les arrels dels fills esquerre i dret tenim $\lfloor p/2 \rfloor - 1$ rangs i $\lceil p/2 \rceil$ rangs, respectivament, etc. A cada node X_i ($0 \leq i < 2^k$) del nivell k d'aquest arbre es resol un problema de selecció amb un subvector de talla n_i i tenim $\sum_i n_i = n$. Llavors el cost de resoldre tots els problemes de selecció al nivell k és $\sum_i \Theta(n_i) = \Theta(n)$, sigui quin sigui k . Com que l'arbre té $\lg p$ nivells el cost de l'algorisme és $\Theta(n \cdot \log p)$. Aquest algorisme sí satisfà els requisits de l'enunciat.

Una altra psosible solució consisteix en un algorisme recursiu **MQUICKSELECT** que a cada crida recursiva fa el següent:

- (a) Escull un pivot x i particiona el vector A respecte a x . Sigui k la posició final del pivot.
- (b) Busquem (amb cerca dicotòmica i cost $O(\log p)$) el rang r_i tal que $r_i \leq k < r_{i+1}$. Adoptem el conveni $r_0 = 0$.
- (c) Cridem recursivament **MQUICKSELECT** en el subvector $A[1..k]$ per trobar els rangs (r_1, \dots, r_i) (si $r_i < k$) o (r_1, \dots, r_{i-1}) (si $r_i = k$).
- (d) Cridem recursivament **MQUICKSELECT** en el subvector $A[k+1..n]$ per trobar els rangs (r_{i+1}, \dots, r_p) .

L'anàlisi del cost d'aquest algorisme segueix el mateix raonament que abans. Però l'arbre de crides recursives no és ara quasi-complet, és un arbre binari qualsevol. LLavors el nombre de nivells en cas pitjor és p i per tant el cost és $\Theta(p \cdot (n + \log p)) = \Theta(p \cdot n)$. Però en promig anirem divint més o menys per la meitat el vector de rangs, de manera semblant al que fa per exemple quicksort i es pot demostrar que en promig hi haurà $\Theta(\log p)$ nivells i per tant el cost en cas promig d'aques algorisme serà $\Theta(\log p(n + \log p)) = \Theta(n \cdot \log p)$. Tot i que aquesta solució no satisfà el requisit en cas pitjor sí

ho fa en cas promig, i es considera una solució raonable. De fet a la pràctica pot funcionar millor que l'anterior doncs el factor amagats en el cost $\Theta(n)$ d'aquesta solució és molt més petit que el correponent de la solució anterior. Dit d'una altra manera: el cost $\Theta(n)$ per nivell de recursió de la primera solució és molt més elevat que el cost $\Theta(n)$ per nivell de la segona solució. Però la segona solució tindrà més (p.e. $5 \log p$) o molts més (p.e. p) nivells.