


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 ressol 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.

- (a) 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.
- (b) 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.

- 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 .

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 $(n \bmod i \neq 0)$ **do**
 $i = i + 1$

$\begin{cases} 3\text{-SAT} \in \text{NP} \\ 2\text{-SAT} \in \text{P} \end{cases}$ ← buscar
 com es fa

- 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 satisfuguin - 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 .

- Realitzeu aquesta construcció per a la instància de 2SAT indicada amunt.
- 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.
- 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.
- 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) $n^{0.01} \in \omega((\lg n)^2)$


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

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

- 1.10.  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'ns dóna un graf $G = (V, E)$ que representa el plànol d'un edifici, hi ha dos robots que inicialment es troben en els nodes a i b . El robot en el node a vol 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

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

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 tems $O(|C| + |M|)$.

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 $O(n^2)$.

- 1.11. El quadrat d'un graf $G = (V, E)$ és un altre graf $G' = (V, E')$ on $E' = \{(u, v) \mid \exists w \in V, (u, w) \in E \wedge (w, v) \in E\}$. 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.

- 1.12. 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 proporcionï les components connexes fortes del graf en temps $O(n + m)$.

- 1.13. Definim els k -mers com les subcadenaes 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.

Formalment, 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 $|S(w)| = |w| - k + 1$. El problema consisteix en, donat un multi-conjunt C de k -mers, trobar la cadena de DNA w tal que $S(w) = C$.

- a Demostreu que hi ha una reducció polinòmica d'aquest problema al problema del camí Hamiltonià, que és NPC (utilitzeu els k -mers com a vèrtexs i el solapament entre k -mers com condició d'existència d'arestes).
- b Demostreu que hi ha una reducció d'aquest problema al problema del camí Eulerià, que és a P (aquest cop, utilitzeu k -mers com a arestes dirigides).
- c Vol dir això que aquest problema és a P i a NPC, i per tant $P=NP$?

- 1.14. ✎ 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 irrellevant. 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 anàlisi podeu suposar que n és múltiple de k i k és una potencia 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 las 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, $T(n) = \Theta(n \lg k)$.

1.15. 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) $T(n) = T(n - 10) + n$

- 1.16. 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 com a arestes totes les arestes a E que connecten vèrtexs en V_1 . Un clique és un subgraf induït 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 $G = (V, A)$.

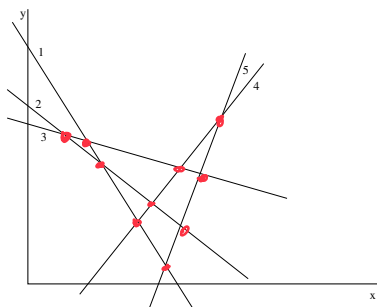
CliqueDV(G)

- 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.

- 1.17. 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 las 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).

$$L_1 = 2x + 3$$

$$L_2 = -x + 7$$

1. Calcular tots els punts de creuament. $O(n^2)$.

2. Mirar per cada x el valor de cada recta.

$S = \text{vector de } \text{bool} > (\text{true, false})$

① for $i = 1, i \leq n, ++i$
 $j = i + 1$

while $j \leq n$;
 $x = \frac{L_{i,b} - L_{j,b}}{L_{i,a} - L_{j,a}}$

$A = \text{add}(x)$
 $++j$;

3. Si el punt en que les 2 rectes es creuen es troba d'un punt maximal
 altrament les 2 rectes son visibles.

Utilitzar un parí $L_{i,j} >$ per emmagatzemar els index de les 2 rectes.

2 rectes.

② for $i = 1, i \leq A, ++i$
 $\text{int aux} = L_i(A[i])$
 $j = i + 1$ de i ;
 while $j \leq n$;
 $\text{if} (\text{aux} \leq L_j(A[j]))$
 $\text{aux} = L_j(A[j])$
 $j = j + 1$;
 $++i$;

$\text{if} (!S[i]) \text{ set} = \text{true};$

- 1.18. Supposeu 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. Sabríeu com fer-ho en temps lineal?

- 1.19. ✎ Donat un vector $A[1..n]$, un element x s'anomena majoritari si x apareix més de $n/2$ cops a A . Donada una taula A doneu un algorisme amb cost $O(n)$ que determini si existeix un element majoritari en A i, en cas que existeixi, digui quin és l'element majoritari.

Una solución

El elemento majoritario, si lo hay, tiene que coincidir con la mediana. Podemos obtener la mediana en $O(n)$ pasos utilizando el algoritmo de selección con coste $O(n)$. Una vez obtenida la mediana, comprobamos si es o no el valor mayoritario con un recorrido del vector A . En total con coste $O(n)$.

- 1.20. Tenim un conjunt de $2n$ valors tots diferents. Una meitat dels valors estan emmagatzemats a una taula A i l'altra meitat a una taula B . Cadascuna de les dues taules està ordenada en ordre creixent i es troba a un ordinador diferent. No hi ha cap relació d'ordre entre els valors a A i els valors a B . Volem trobar la mediana del total dels $2n$ valors. Doneu un algorisme amb cost $O(\lg n)$ que permeti obtenir la mediana sota la hipòtesis que només podeu fer crides de la forma $\text{Element}(i, A)$ o $\text{Element}(i, B)$, per $1 \leq i \leq n$, que retornen l'element i -èsim a A o a B , respectivament (amb cost $O(1)$).

$|A| + |B| = 2n$ La idea és que tenim 2 arrays ordenats creixentment
 $|A| = n$ i volem trobar la seva mediana. Ho farem per dividir i conquerir i amb el mètode de binary search.
 $|B| = n$

$a_1 = \{1, 5, 7, 10, 13\}$

$a_2 = \{11, 15, 23, 30, 45\}$

$a_1 = (a_1, b_1, c, d, e)$

$a_2 = (f, g, h, i, k)$

index i fa com una mena de complementària de i. en $i + j = n - 1$.

int medianRec($a_1, a_2, \text{left}, \text{right}, n$) {

if ($\text{left} > \text{right}$) { // acabar la recursió de a_1 i començar la de a_2 .

return medianRec($a_2, a_1, 0, n-1, n$)

else
 $i = (\text{left} + \text{right}) / 2;$

$j = n - i - 1;$

if ($a_1[i] > a_2[j]$ i $a_1[i] < a_2[j+1]$) // aquí trobem la prova mediana.

if ($a_1[i] > a_2[j+1]$)
return ($a_1[i] + a_2[j]$) / 2;

else
return ($a_1[j] + a_2[i+1]$) / 2;

else if ($a_1[i] > a_2[j]$ i $a_1[i] > a_2[j+1]$) { // Mover la part esquerra
return medianRec($a_1, a_2, \text{left}, i-1, n$)

else
return medianRec($a_1, a_2, i+1, \text{right}, n$) // Mover la part dreta

- 1.21. Tenim un vector $A[1, \dots, n]$ no ordenat amb claus no necessàriament numèriques, però que pertanyen a un conjunt totalment ordenat de claus. Sigui x_i el 2^i -èsim element més petit en A . Doneu un algorisme per calcular la suma dels valors x_i , per $1 \leq 2^i \leq n$, en $\Theta(n)$ passos.

A no està ordenat

$x_i \rightarrow 2^i$ -èsim element més petit en A .

Si $x_i = s$ significa que $2^i = 16$ -element més petit

en A .

~~ordenar, buscar i sumar un
anar a buscar els $\log n$ elements
mitjançant selecció.~~

$$k \leq n$$

$$\sum_{i=0}^k 2^i = O(2^{k+1}) = O(n)$$

$$\left[\sum_{i=0}^n a_i = \frac{a^{n+1} - 1}{a - 1} \right] \quad x_i = \text{element}$$

La idea es com que cada vegada vas
trobar la meitat d'elements de cerca
de A . Una vegada que trobem
l'element 2^i deixem els elements més gran
que 2^i no ens fa falta. I així successivament
el següent seria fer $\text{select}(2^{i+1}, B_i)$.

$B_i \leftarrow \text{element} \leq x_i \text{ de } B_{i+1}$.

1. obtenir $k = \lceil \log n \rceil$ (k es l'element més gran)

2. $x_k = \text{select}(2^k, A) \rightarrow \text{cost}(n)$

3. Suma $\leftarrow x_k$

4. Sigui B_k els elements $< x_k$.
 B_k té ordre $\leq 2^k$ $\left. \vphantom{\begin{matrix} B_k \end{matrix}} \right\} O(n)$

$$2^k \leq n$$

5. For $i = k-1$ to 0

$x_i = \text{select}(2^i, B_{i+1})$ i Suma $\leftarrow x_i$

$B_i \leftarrow \text{element} \leq x_i \text{ de } B_{i+1}$

6. Return suma.

$$O(2^{i+1})$$