Problemes 1

1.1. **(Quin no hi és?)** 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.

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

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

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

1.3. **(Celebritat?)** 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.

(a) Doneu una formalització de la propietat de ser celebritat.

(b) 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.

1.4. **(És 2-colorable?)** Un graf $G = (V, E)$ es k-colorable si existeix una coloración dels vèrtex de G amb k-colors tal que els extrems de una aresta a E tenen color diferents.

(a) Doneu una formalització de la propietat de ser 2-colorable.

(b) Demostreu que un graf es 2-colorable si i només si es bipartit.

(c) Fent servir la propietat anterior doneu un algorisme eficient per determinar si un graf és 2-colorable.

1.5. **(k-mers)** 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.

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?

1.6. **(Un joc d'actrius i actors)** Tenim dos jugadors P_0 i P_1 i dues llistes, X i Y. La llista X conté els noms d'n actrius amb les pel·lícules on han participat, i la llista Y conte els noms d'n actors i les pel·lícules on han participat. Cada jugador disposa de còpies de les llistes X i Y amb la informació completa de qui ha aparegut en una pel·lícula amb qui.

El joc consisteix en el següent, el jugador P_0 diu el nom d'una actriu $x_1 \in X$, aleshores P_1 ha de donar el nom d'un actor $y_1 \in Y$ que ha aparegut a una pel·lícula amb x_1, a continuació P_0 ha de donar el nom d'una actriu $x_2 \in X$, $x_2 \neq x_1$, que ha aparegut a una pel·lícula amb y_1, i així successivament fins que un dels dos jugadors no pot donar cap nom diferent. Noteu que la restricció important és que no es poden repetir noms al llarg de la partida. El jugador que per primer cop no pot donar cap nom és el jugador que perd.

Podeu observar que a una partida P_0 i P_1 generen col·lectivament una seqüència

$$x_1, y_1, x_2, y_2, \ldots, x_i, y_i, \ldots,$$

de manera que cada actor/actriu en la seqüència ha coprotagonitzat al costat de l'actriu/actor immediatament anterior, i no es repeteixen noms.

Una estratègia per al jugador P_0 és un algorisme que pren com a entrada una seqüència $x_1, y_1, x_2, y_2, \ldots, x_i, y_i$ i genera un x_{i+1} legal, si això és possible. De la mateixa manera podem definir una estratègia per a P_1.

Si a més ens donen una forma de formar n parelles actriu-actor, de manera que cap actor ni cap actriu apareix a més d'una parella i a més per cada parella actriu-actor tots dos han participat junts a alguna película. Voldrias jugar com P_0 o com P_1? Com jugaries? Justifica la teva resposta.

1.7. 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) $n^{0.01} \in \omega((\lg n)^2)$

1.8. 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

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

1.9. **(És fortament connex?)** 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.

1.10. **(És semiconnex?)** 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 $O(n+m)$.

1.11. Considereu el següent algorisme de dividir-i-vèncer que té com entrada un vector d'enters.

```
SRT(A[1..k])
1: if n ≤ 2 then
2:     if A[1] > A[k] then
3:         Interchange A[1] with A[k]
4: else
5:     m = ⌊n/3⌋
6:     SRT(A[1..k − m])
7:     SRT(A[m + 1..k])
8:     SRT(A[1..k − m])
9: return (A)
```

(a) Demostra que $\mathsf{SRT}(A[1..n])$ torna el vector A ordenat creixentment.

(b) Dona el cost temporal de SRT indicant com ho has obtingut.

1.12. **(Clique max?)** 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 a arestes totes les arestes 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 $1, 2, \ldots, n$, on $n = |V|$
2: Si $n = 1$ tornar V
3: Dividir V en $V_1 = \{1, 2, \ldots, \lfloor n/2 \rfloor\}$ i $V_2 = \{\lfloor n/2 \rfloor + 1, \ldots, n\}$
4: Sigui $G_1 = G[V_1]$ i $G_2 = G[V_2]$
5: C_1= CliqueDV(G_1) i C_2= 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:

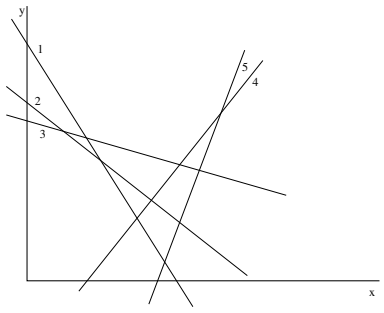
(a) Demostreu que l'algorisme CliqueDV sempre retorna un subgraf de G que és un clique.

(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.

1.13. **(Línies visibles)** 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, $L_1, \ldots, L_n$, 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 $\mathcal{L} = \{L_1, \ldots, L_n\}$, 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).

1.14. **(Moltes còpies?)** 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 tageta. 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?

1.15. **(Ordena segment I)** 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.

1.16. **(Ordena segment II)**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.

1.17. Com ordenar eficientment elements de longitud variable:

(a) Donada una taula d'enters, on els enters emmagatzemats poden tenir diferent nombre de dígits. Però sabem que el nombre total de dígits 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 al 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, $a < ab < b$.)

1.18. **(Mediana amb dades repartides)** 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. Volen 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 $\mathsf{Element}(i, A)$ o $\mathsf{Element}(i, B)$, per $1 \leq i \leq n$, que retornen l'element i-ésim a A o a B, respectivament (amb cost $O(1)$).

1.19. **(Suma elements)** Tenim un vector $A[1,\ldots,n]$ no ordenat on cada registre conté un enter de valor entre -2^n i 2^n. 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.

1.20. **(Millor tall)** Tenim el següent problema:

L'entrada és una seqüència de nombres diferents $S = (x_1, x_2, \ldots, x_n)$, tal que cada element $x_i \in S$ té associat un pes $w(x_i) > 0$. Sigui $W = \sum_{x_i \in S} w(x_i)$ la suma dels pesos de tots els elements de la seqüència.

Donat un valor X $(0 \leq X \leq W)$, el problema consisteix a trobar l'element x_j de la seqüència tal que

$$\sum_{x_i < x_j} w(x_i) < X \qquad \text{i} \qquad w(x_j) + \sum_{x_i < x_j} w(x_i) \geq X$$

Dissenyeu un algorisme eficient ($o(n \log n)$) per a resoldre aquest problema.

1.21. **(Azamon)** L'empresa de paqueteria Azamon vol col·locar una nova guixeta de recollida de paquets a un poble que no en té cap. Demanen a l'ajuntament del poble que els doni informació sobre possibles llocs on els hi deixarien instal·lar i on els habitants del poble s'hi podrien desplaçar fàcilment.

La informació que proporcionen consisteix en n punts $\{p_1, p_2, \ldots, p_n\}$, que indiquen les possibles localitzacions per a les guixetes, i uns pesos positius associats $\{w_1, w_2, \ldots, w_n\}$, que en quantifiquen la idoneïtat en funció del nombre de persones que hi viuen a prop, les seves edats, discapacitats, etc. Aquests pesos estan normalitzats de manera que $\sum_{i=1}^{n} w_i = 1$.

Azamon ha decidit que posarà la guixeta al punt p que minimitzi la suma

$$S = \sum_{i=1}^{n} w_i \cdot d(p, p_i), \tag{1}$$

on $d(a, b)$ és la distància entre dos punts a i b. Considereu les dues següents situacions:

a) **Versió 1d:** Els n punts $\{p_1, p_2, \ldots, p_n\}$ indiquen les localitzacions de possibles llocs d'instal·lació al llarg d'un camí que travessa el poble; per tant $\forall p_i : p_i \in \mathbb{Z}^+$, i la distància entre dos punts a i b és $d(a, b) = |a - b|$.

   Podeu suposar que tot punt $p_i \leq 1000\ n$.

b) **Versió 2d:** Els n punts $\{p_1, p_2, \ldots, p_n\}$ indiquen les localitzacions de possibles llocs d'instal·lació en un mapa amb coordenades; per tant $\forall p_i : p_i = (x_i, y_i) \in \mathbb{Z}^+ \times \mathbb{Z}^+$, i la distància entre dos punts $p_a = (x_a, y_a)$ i $p_b = (x_b, y_b)$ és la distància de Manhattan, és a dir, $d((x_a, y_a), (x_b, y_b)) = |x_a - x_b| + |y_a - y_b|$.

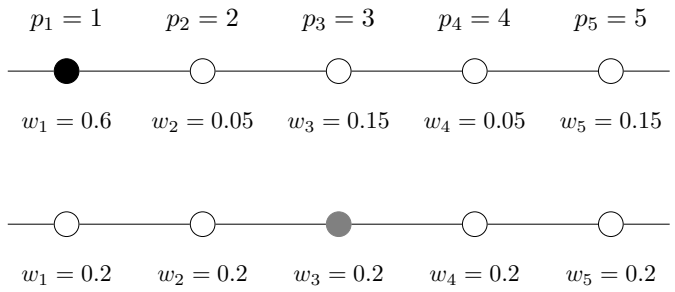
   Aquí també podeu suposar que tot punt $p_i = (x_i, y_i)$ té ambdues coordenades afitades superiorment $x_i, y_i \leq 1000\ n$.

Es demana:

Doneu un algorisme eficient que, donats n punts i els seus n pesos, calculi el punt p on col·locar la guixeta d'Azamon de manera que p minimitzi la suma esmentada. Resoleu el problema per cadascuna de les dues versions (1d i 2d).

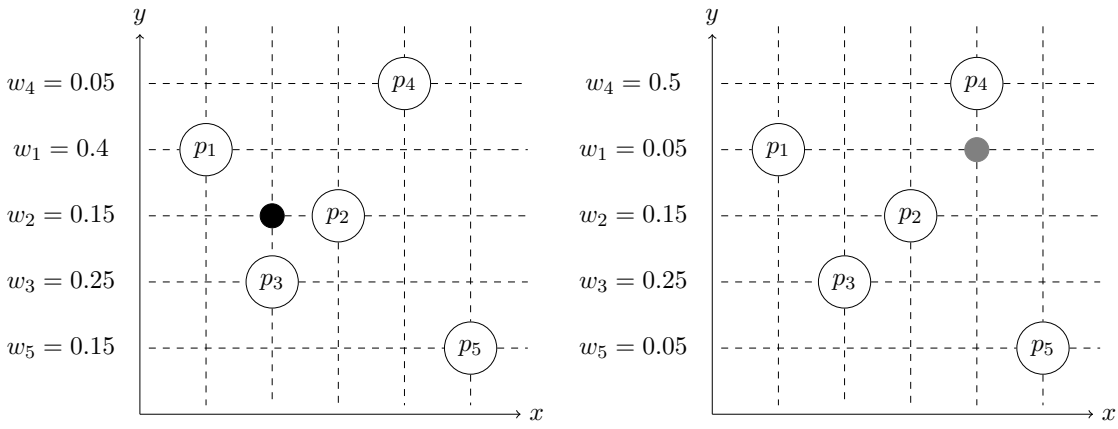
Exemples:

a) Sigui $\{1, 2, 3, 4, 5\}$ el conjunt de punts i $\{0.6, 0.05, 0.15, 0.05, 0.15\}$ el conjunt dels respectius pesos. Considerant la distància entre punts a una recta numèrica (versió 1d). Llavors la solució buscada és el punt $p = p_1$ (en negre, a la figura de sobre), doncs aquesta elecció de p minimitza la suma S. Observeu que, en canvi, si el conjunt dels pesos fos $\{0.2, 0.2, 0.2, 0.2, 0.2\}$, aleshores tindríem que és el punt $p = p_3$ (en gris, a la figura de sota) el que minimitza S.



b) Suposem ara que els punts són $p_1 = (1, 4)$, $p_2 = (3, 3)$, $p_3 = (2, 2)$, $p_4 = (4, 5)$ i $p_5 = (5, 1)$, i els pesos respectius $\{0.4, 0.15, 0.25, 0.05, 0.15\}$. Llavors la solució és el punt $p = (2, 3)$ (en negre, a la figura esquerra). A diferència del cas anterior, la solució no necessàriament és un punt dels n punts donats.

Un altre exemple: si els pesos fossin $\{0.05, 0.15, 0.25, 0.5, 0.05\}$ llavors la solució seria el punt $p = (4, 4)$ (en gris, a la figura dreta).



1.22. **(Mínim nombre de canvis)** Donat un vector A d'enters, dissenyeu un algorisme el més eficient possible que trobi quin és el mínim nombre (total) d'increments o decrements que s'han d'aplicar sobre els seus elements per tal que tots esdevinguin iguals.

Per exemple, si $A = [3, -1, 4]$ llavors la resposta és 5 (per exemple, incrementant $A[1]$ quatre vegades i decrementant $A[2]$ un cop). Si el vector A fos $[3, -1, 5, 6]$ llavors la resposta seria 9 (per exemple, 2 increments d'$A[0]$, més 6 increments d'$A[1]$, més 1 decrement d'$A[3]$).

Justifica la correctesa del teu algorisme i calcula'n el cost.