Question 1 : Classes de complexité (3 points)

a) **[2 points]** Expliquez la différence entre les classes $\mathcal{P}$ et $\mathcal{NP}$.

Réponse: La classe de complexité $\mathcal{P}$ est l'ensemble des problèmes de décision qu'on peut résoudre en un temps polynomial dans la taille de l'exemplaire. La classe de complexité $\mathcal{NP}$ est l'ensemble des problèmes de décision dont on peut vérifier *une réponse* affirmative *en un temps polynomial dans la taille de l'exemplaire.*

b) **[1 point]** À quoi peut servir la théorie de la $\mathcal{NP}$ complétude pour un ingénieur concepteur d'algorithme ?

Réponse: En démontrant qu'un problème est $\mathcal{NP}$ complet, il établit que ce problème est aussi difficile que tout un ensemble de problèmes déjà très étudiés et réputés difficiles. Cela sert à justifier son incapacité à concevoir un algorithme exact efficace (polynomial) pour ce problème et motive l'utilisation d'approches heuristiques, approximatives, pseudo-polynomiales ou même probabilistes.

Question 2 : Algorithmes de recherche locale (5 points)

On vous confie la tâche de concevoir un logiciel pour planifier l'emplacement des entrepôts d'une grande entreprise agro-alimentaire afin de desservir les supermarchés québécois. Votre solution devrait minimiser l'ensemble des coûts de construction des entrepôts et de distribution aux supermarchés. Vous reconnaissez rapidement qu'il s'agit d'un problème de localisation d'entrepôt : étant donné m sites potentiels pour les entrepôts et n clients de situation géographique connue à desservir, décider où bâtir des entrepôts et, pour chaque client, quel entrepôt le prendra en charge, tout en considérant la capacité des entrepôts et la demande de chacun des clients. Vous décidez d'adopter un algorithme de recherche locale, en représentant une solution par un vecteur de taille n identifiant, pour chaque client, le site de l'entrepôt qui le desservira. Par exemple, si on a trois sites potentiels A, B et C pour les entrepôts et cinq clients, le vecteur $\langle B, B, A, B, A\rangle$ représente une solution où l'entrepôt A dessert les troisième et cinquième clients alors que l'entrepôt B s'occupe des trois autres.

a) **[1 point]** Vous choisissez d'utiliser un voisinage de type réaffectation, changeant l'entrepôt desservant un des clients. Quelle est la taille de votre voisinage, en notation asymptotique ?

Réponse: On choisit un client parmi n et un nouveau site d'entrepôt pour celui-ci parmi $m-1$, ce qui donne $\Theta(mn)$.

b) **[2 points]** Décrivez une méthode de descente (amélioration locale) pour ce problème, qui utilise le voisinage précédent. Pour simplifier, vous n'avez pas à vous préoccuper du respect de la capacité des entrepôts.

Réponse:

Démarrer avec une certaine solution $s = \langle e_1, \ldots, e_n \rangle$ *;*

tant que $\exists\, c \in \{1, \ldots, n\}, e \in \{1, \ldots, m\} \setminus \{e_c\}$
tels que $f(\langle e_1, \ldots, e_{c-1}, e, e_{c+1}, \ldots, e_n \rangle) < f(s)$ **faire**
$s \leftarrow \langle e_1, \ldots, e_{c-1}, e, e_{c+1}, \ldots, e_n \rangle$ *;*

retourner *s ;*

c) **[2 points]** Si vous utilisez simplement cette méthode de descente, vous atteindrez un minimum local. Décrivez deux stratégies employées par les métaheuristiques pour tenter d'échapper à un minimum local.

Réponse:

recherche avec tabou : on se déplace toujours vers le meilleur voisin (même s'il est plus mauvais) mais on interdit le retour à une solution déjà visitée pendant un certain nombre d'itérations.

recuit simulé : on choisit un voisin au hasard et on se déplace vers lui même s'il dégrade la qualité de la solution (mais avec une faible probabilité).

recherche à voisinage variable : plusieurs types de voisinages sont utilisés ; lorsqu'on atteint un minimum local dans un, on passe à un autre voisinage.

Question 3 : Algorithmes probabilistes [2 points]

Afin de résoudre un problème de décision, vous avez sous la main :

(A) un algorithme Monte Carlo $\frac{1}{2}$-correct biaisé pour la réponse "oui" et qui prend 2 secondes à chaque exécution ;

(B) un algorithme Monte Carlo $\frac{3}{4}$-correct biaisé pour la réponse "non" et qui prend 3 secondes à chaque exécution ;

On vous demande de concevoir un algorithme Monte Carlo qui soit au moins $\frac{9}{10}$-correct et le plus rapide possible. Que proposez-vous et combien de temps prendra votre algorithme ?

Réponse: On procède par amplification stochastique :

Quatre répétitions de A donneront une probabilité de succès d'au moins $\frac{15}{16} > \frac{9}{10}$ *et prendront 8 secondes en tout.*

Deux répétitions de B donneront une probabilité de succès d'au moins $\frac{15}{16} > \frac{9}{10}$ *et prendront 6 secondes en tout.*

La seconde option est donc la meilleure.

Question 4 : Programmation dynamique [3 points]

On considère le problème d'impression équilibrée d'un paragraphe sur une imprimante. Le texte d'entrée est une séquence de n mots de longueurs $\ell_1, \ell_2, \ldots, \ell_n$, mesurées en caractères. On souhaite imprimer ce paragraphe de façon "équilibrée" sur un certain nombre de lignes qui contiennent un maximum de M caractères chacune. Si une ligne donnée contient les mots de i à j, $i \leq j$, et qu'on laisse exactement un espace entre deux mots, le nombre de caractères d'espacement supplémentaires à la fin de la ligne correspond à

$$s_{ij} = M - j + i - \sum_{h=i}^{j} \ell_h.$$

Si les mots i à j ne tiennent pas sur la ligne, s_{ij} sera négatif : convenons plutôt que dans ce cas $s_{ij} = \infty$. On souhaite minimiser la somme, sur toutes les lignes, des carrés des nombres de caractères d'espacement supplémentaires à la fin de chaque ligne. Par exemple, si le texte tient sur 3 lignes et que les nombres de caractères d'espacement supplémentaires pour chacune des lignes sont 2, 4 et 1, la somme des carrés sera $2^2 + 4^2 + 1^2 = 21$. Définissons un tableau E de dimension $n \times n$ dont une entrée $E[j, k]$ représente la plus petite somme des carrés des nombres de caractères d'espacement supplémentaires pour imprimer les j premiers mots sur k lignes.

a) [**1 point**] Donnez une formule pour calculer $E[j, k]$ à partir d'autres cases du tableau. (Indice : utilisez s_{ij}.)

Réponse: $E[j, k] = \min_{i=2}^{j}\{E[i-1, k-1] + s_{ij}^2\}$.

b) [**1 point**] Où retrouve-t-on la valeur de la solution ?

Réponse: C'est la première valeur qui ne soit pas l'infini sur la dernière ligne du tableau, en partant de la gauche. (Ou encore, la plus petite valeur de la dernière ligne.)

c) [**1 point**] Donnez le temps d'exécution de votre algorithme en notation asymptotique.

Réponse: $\Theta(n^3)$

Question 5 : Parcours d'arbre et diviser-pour-régner (7 points)

Trouver la meilleure façon de dessiner un graphe n'est pas chose facile et plusieurs algorithmes ont été développés pour cela. Nous nous intéresserons ici à un cas bien particulier, celui des arbres binaires. Donnant pour chaque sommet, le cas échéant, ses fils gauche et droit, voici un exemple d'arbre binaire de racine A que nous utiliserons :

```
A: fils gauche B, fils droit C
B: fils gauche D
```

```
C: fils gauche E, fils droit F
D: fils droit G
E: fils gauche H, fils droit I
G: fils gauche J, fils droit K
```

Nous placerons ses sommets aux points d'intersection d'une grille (comme celle dans votre cahier d'examen). Nous cherchons à produire une représentation étagée selon la profondeur des sommets, ce qui déterminera la coordonnée en y de ceux-ci : la racine A sur la première ligne, ses enfants B et C sur la deuxième ligne, leurs enfants D, E et F sur la troisième, etc. Nous voulons aussi que les arêtes du dessin ne s'intersectent pas : il suffit pour cela de s'assurer que si un sommet u est placé à la gauche d'un sommet v sur la même ligne, alors les enfants de u sont aussi placés à la gauche des enfants de v.

a) **[2 points]** Voici une façon facile d'obtenir un tel dessin. Dessinez l'arbre précédent sur une grille en utilisant comme coordonnée en x le rang de chaque sommet dans un parcours en-ordre. (Je rappelle que la coordonnée en y correspondra à la profondeur du sommet dans l'arbre.)

Réponse:

b) **[3 points]** Le résultat de (a) n'est pas très esthétique. Tout en conservant une représentation étagée sur une grille sans croisement d'arêtes, donnez un algorithme diviser-pour-régner qui produira un dessin de l'arbre de plus petite largeur que celui obtenu en (a) et surtout où chaque parent est centré (en x) par rapport à ses deux enfants (lorsqu'il en a deux). Dessinez votre résultat pour l'arbre donné.

Réponse:

fonction *diviser-pour-regner(arbre de racine r) : dessin de cet arbre*

si *r n'a pas d'enfants* **alors retourner** *dessin trivial ;*

si *r a un fils gauche, g,* **alors** $D_g \leftarrow$ *diviser-pour-regner(sous-arbre de racine g) ;*

si *r a un fils droit, d,* **alors** $D_d \leftarrow$ *diviser-pour-regner(sous-arbre de racine d) ;*

si *r a deux enfants* **alors**

ajuster les coordonnées en x de D_d *de telle sorte que la distance horizontale avec* D_g *soit de 2 ou 3, plaçant les représentations de g et d à une distance paire l'une de l'autre ;*

placer r à mi-chemin entre g et d, formant le dessin D ;

si *r a un seul fils, à gauche (resp. droite)* **alors**

placer r à une unité à droite (resp. gauche) de ce fils, formant le dessin D ;

retourner *D*

c) **[2 points]** Faites l'analyse en pire cas du temps de calcul pris par votre algorithme. Refaites cette analyse sous l'hypothèse qu'on vous donne un arbre binaire complet (et donc équilibré).

Réponse: En général on aura la récurrence suivante pour un arbre sur n sommets : $T(n) = T(1) + T(n-2) + cn$. *Ce qui nous donne* $T(n) \in \Theta(n^2)$. *Si l'arbre est équilibré, on aura plutôt* $T(n) = 2T(n/2) + cn$, *qui donne* $T(n) \in \Theta(n \log n)$.