

IFT436 – Algorithmes et structures de données
Université de Sherbrooke

Examen final

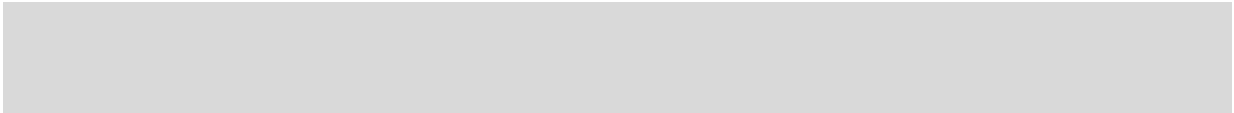
Enseignant: Michael Blondin
Date: mercredi 22 décembre 2021
Durée: 3 heures

Directives:

- Vous devez répondre aux questions dans le **cahier de réponses**, et *non* sur ce questionnaire;
- **Une seule feuille** de notes au format $8\frac{1}{2}'' \times 11''$ est permise;
- Les **fiches récapitulatives** des chapitres 5 à 8 se trouvent à la fin de ce questionnaire;
- **Aucun matériel additionnel** n'est permis;
- **Aucun appareil électronique** (calculatrice, téléphone, montre intelligente, etc.) n'est permis;
- Vous devez donner **une seule réponse** par sous-question;
- L'examen comporte **5 questions** sur **5 pages** valant un total de **50 points**;
- La correction se base notamment sur la **clarté**, l'**exactitude** et la **concision** de vos réponses, ainsi que sur la **justification** pour les questions qui en requièrent une;
- Les indices d'une séquence **débutent à 1**; autrement dit, $s = [s[1], s[2], \dots, s[n]]$ si $n = |s|$.

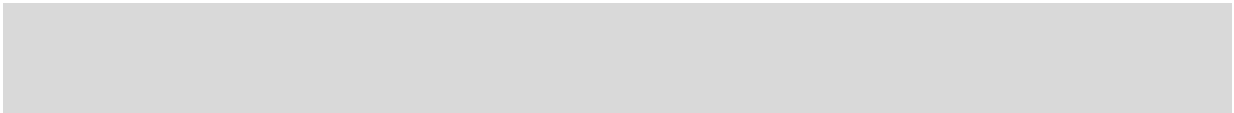
Question 1: analyse d'algorithmes récursifs

(a)



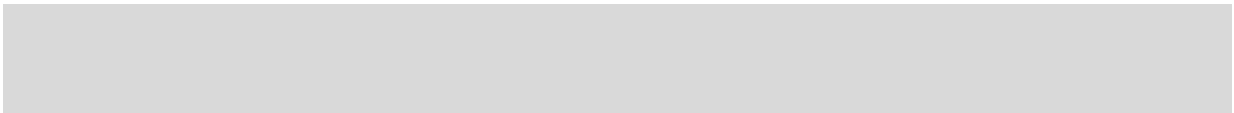
2,5 pts

(b)

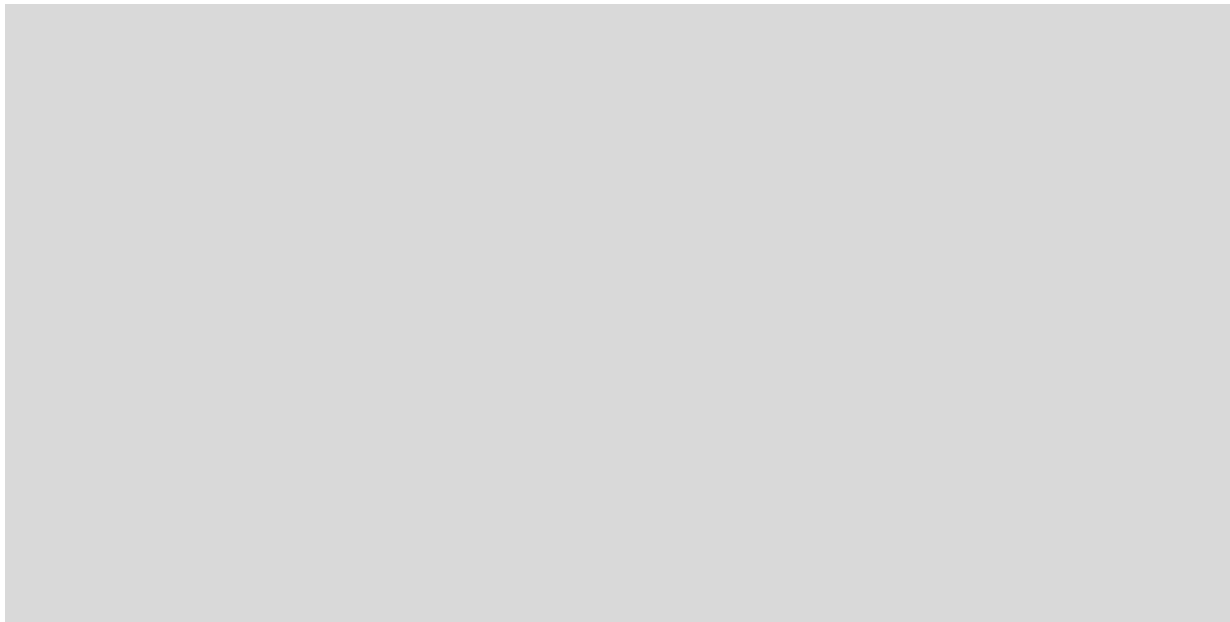


3 pts

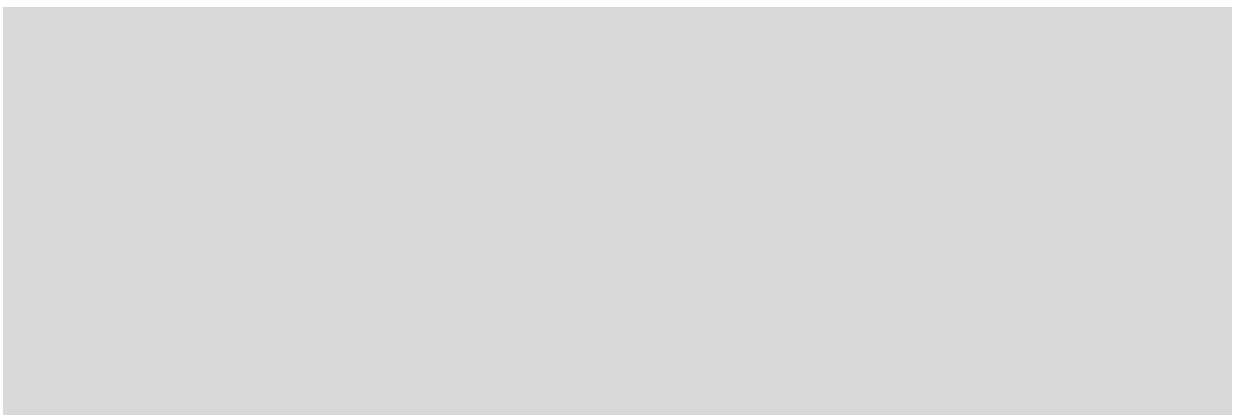
(c)



2,5 pts

Question 2: diviser-pour-régner

(a)



7 pts

(b)

2 pts

(c)

3 pts

Question 3: force brute et programmation dynamique

(a)

3 pts

(b)

2 pts

(c)

5 pts

(d)

2 pts

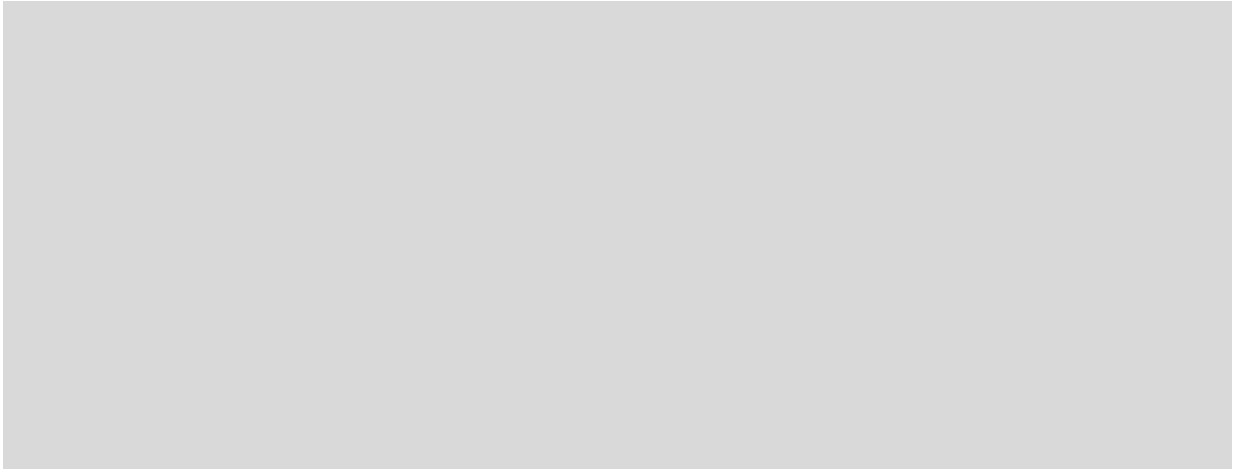
Question 4: plus courts chemins

(a)



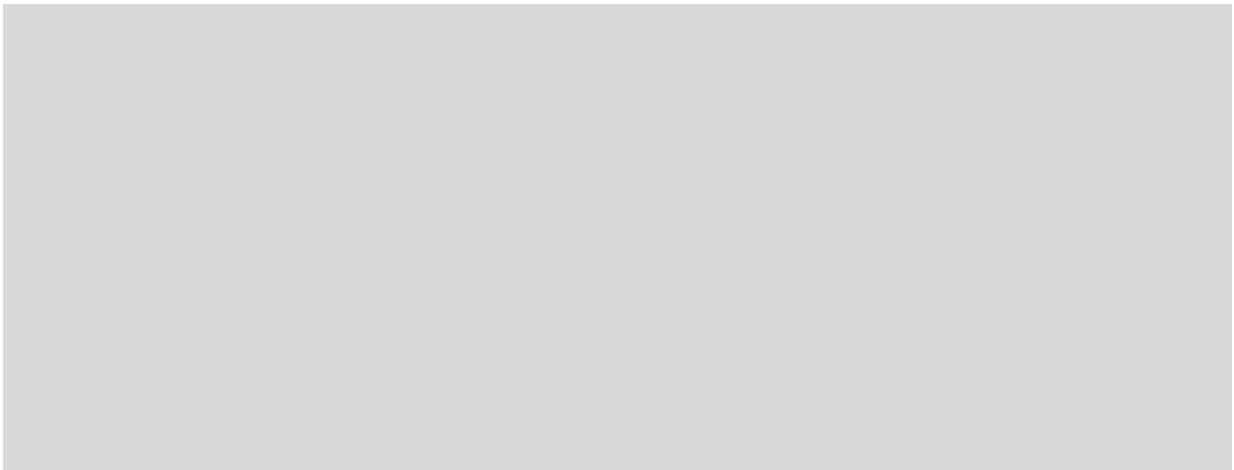
2,5 pts

(b)



3 pts

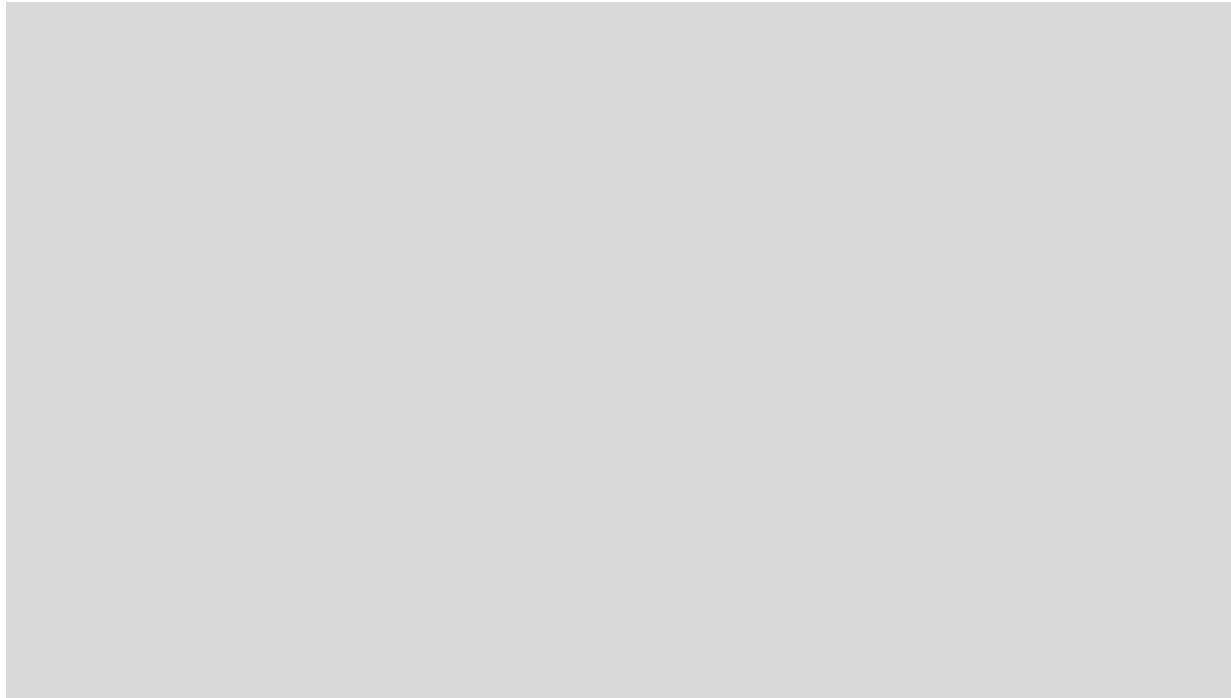
(c)



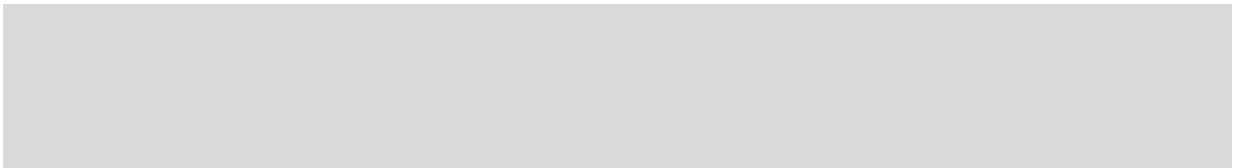
2,5 pts

Question 5: algorithmes probabilistes

Il n'est pas nécessaire d'évaluer vos résultats numériquement; des expressions symboliques suffisent.

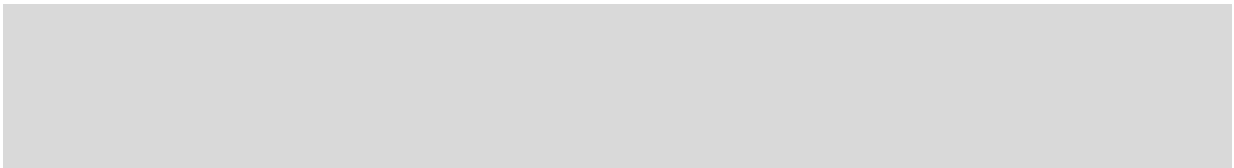


(a)



3 pts

(b)



7 pts

Annexe:

Fiches récapitulatives

5. Algorithmes récursifs et approche diviser-pour-régner

Diviser-pour-régner

- A) découper en sous-problèmes disjoints
- B) obtenir solutions récursivement
- C) s'arrêter aux cas de base (souvent triviaux)
- D) combiner solutions pour obtenir solution globale
- Exemple: tri par fusion $\mathcal{O}(n \log n)$

Réurrences linéaires

- Cas homogène: $\sum_{i=0}^d a_i \cdot t(n-i) = 0$
- Polynôme caractéristique: $\sum_{i=0}^d a_i \cdot x^{d-i}$
- Forme close: $t(n) = \sum_{i=1}^d c_i \cdot \lambda_i^n$ où les λ_i sont les racines
- Constantes c_i : obtenues en résolvant un sys. d'éq. lin.
- Cas non homo.: si $= c \cdot b^n$, on multiplie poly. par $(x-b)$
- Exemple:
 - Récurrence: $t(n) = 3 \cdot t(n-1) + 4 \cdot t(n-2)$
 - Poly. carac.: $x^2 - 3x - 4 = (x-4)(x+1)$
 - Forme close: $t(n) = c_1 \cdot 4^n + c_2 \cdot (-1)^n$

Autres méthodes

- Substitution: remplacer $t(n), t(n-1), t(n-2), \dots$ par sa déf. jusqu'à deviner la forme close
- Arbres: construire un arbre représentant la récursion et identifier le coût de chaque niveau

Quelques algorithmes

- Hanoi: $src[1:n-1] \rightarrow tmp, src[n] \rightarrow dst, tmp[1:n-1] \rightarrow dst$ $\mathcal{O}(2^n)$
- Exp. rapide: exploiter $b^n = (b^{n \div 2})^2 \cdot b^{n \bmod 2}$ $\mathcal{O}(\log n)$
- Mult. rapide: calculer $(a+b)(c+d)$ en 3 mult. $\mathcal{O}(n^{\log 3})$
- Horizon: découper blocs comme tri par fusion $\mathcal{O}(n \log n)$

Théorème maître (allégé)

- $t(n) = c \cdot t(n \div b) + f(n)$ où $f \in \mathcal{O}(n^d)$:
 - $\mathcal{O}(n^d)$ si $c < b^d$
 - $\mathcal{O}(n^d \cdot \log n)$ si $c = b^d$
 - $\mathcal{O}(n^{\log_b c})$ si $c > b^d$

6. Force brute

Approche

- Exhaustif: essayer toutes les sol. ou candidats récursivement
- Explosion combinatoire: souvent # solutions $\geq b^n, n!, n^n$
- Avantage: simple, algo. de test, parfois seule option
- Désavantage: généralement très lent et/ou avare en mémoire

Techniques pour surmonter explosion

- Élagage: ne pas développer branches inutiles
- Contraintes: élaguer si contraintes enfreintes
- Bornes: élaguer si impossible de faire mieux
- Approximations: débiter avec approx. comme meilleure sol.
- Si tout échoue: solveurs SAT ou d'optimisation

Problème des n dames

- But: placer n dames sur échiquier sans attaques
- Algo.: placer une dame par ligne en essayant colonnes dispo.

Sac à dos

- But: maximiser valeur sans excéder capacité
- Algo.: essayer sans et avec chaque objet
- Mieux: élaguer dès qu'il y a excès de capacité
- Mieux++: élaguer si aucune amélioration avec somme valeurs

Retour de monnaie

- But: rendre montant avec le moins de pièces
- Algo.: pour chaque pièce, essayer d'en prendre 0 à # max.

7. Programmation dynamique

Approche

- *Principe d'optimalité*: solution optimale obtenue en combinant solutions de sous-problèmes qui se chevauchent
- *Descendante*: algo. récursif + mémorisation (ex. Fibonacci)
- *Ascendante*: remplir tableau itér. avec solutions sous-prob.

Retour de monnaie

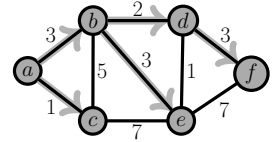
- *Sous-question*: # pièces pour rendre j avec pièces 1 à i ?
- *Identité*: $T[i, j] = \min(T[i-1, j], T[i, j-s[i]] + 1)$
- *Exemple*: montant $m = 10$ et pièces $s = [1, 5, 7]$

	0	1	2	3	4	5	6	7	8	9	10
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	1	2	3	4	5	6	7	8	9	10
2	0	1	2	3	4	1	2	3	4	5	2
3	0	1	2	3	4	1	2	1	2	3	2

Sac à dos

- *Sous-question*: val. max. avec capacité j et les objets 1 à i ?
- *Identité*: $T[i, j] = \max(T[i-1, j], T[i-1, j-p[i]] + v[i])$

Plus courts chemins



- *Déf.*: chemin simple de poids minimal
- *Bien défini*: si aucun cycle négatif
- *Approche générale*: raffiner distances partielles itérativement
- *Dijkstra*: raffiner en marquant sommet avec dist. min.
- *Floyd-Warshall*: raffiner via sommet intermédiaire v_k
- *Bellman-Ford*: raffiner avec $\geq 1, 2, \dots, |V| - 1$ arêtes
- *Sommaire*:

	Dijkstra	Bellman-Ford	Floyd-Warshall
Types de chemins	d'un sommet vers les autres	paires de sommets	
Poids négatifs?	✗	✓	✓
Temps d'exécution	$\mathcal{O}(V \log V + E)$	$\Theta(V \cdot E)$	$\Theta(V ^3)$
Temps ($ E \in \Theta(1)$)	$\mathcal{O}(V \log V)$	$\Theta(V)$	$\Theta(V ^3)$
Temps ($ E \in \Theta(V)$)	$\mathcal{O}(V \log V)$	$\Theta(V ^2)$	$\Theta(V ^3)$
Temps ($ E \in \Theta(V ^2)$)	$\mathcal{O}(V ^2)$	$\Theta(V ^3)$	$\Theta(V ^3)$

8. Algorithmes et analyse probabilistes

Modèle probabiliste

- *Modèle*: on peut tirer à pile ou face (non déterministe)
- *Aléa*: on peut obtenir une loi uniforme avec une pièce
- *Idéalisé*: on suppose avoir accès à une source d'aléa parfaite (en pratique: source plutôt pseudo-aléatoire)

Algorithmes de Las Vegas

- *Temps*: varie selon les choix probabilistes
- *Valeur de retour*: toujours correcte
- *Exemple*: tri rapide avec pivot aléatoire
- *Temps espéré*: dépend de $\mathbb{E}[Y_x]$ où $Y_x = \#$ opér. sur entrée x

Algorithmes de Monte Carlo

- *Temps*: borne ne varie pas selon les choix probabilistes
- *Valeur de retour*: pas toujours correcte
- *Exemple*: algorithme de Karger
- *Prob. d'erreur*: dépend de $\Pr(Y_x \neq \text{bonne sortie sur } x)$

Coupe minimum: algorithme de Karger

- *Coupe*: partition (X, Y) des sommets d'un graphe non dirigé
- *Taille*: # d'arêtes qui traversent X et Y
- *Coupe min.*: identifier la taille minimale d'une coupe
- *Algorithme*: contracter itérativement une arête aléatoire en gardant les multi-arêtes, mais pas les boucles



- *Prob. d'erreur*: $\leq 1 - 1/|V|^2$ (Monte Carlo)
- *Amplification*: on peut réduire (augmenter) la prob. d'erreur (de succès) arbitrairement (en général: avec min., maj., $\vee$, etc.)

Temps moyen

- *Temps moyen*: $\sum(\text{temps instances de taille } n) / \# \text{ instances}$
- *Attention*: pas la même chose que le temps espéré
- *Hypothèse*: entrées distribuées uniformément ($\pm$ réaliste)
- *Exemple*: $\Theta(n^2)$ pour le tri par insertion