Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

CSI2110/CSI2510

Algorithmes et Structures de données

Examen Final

Durée: 3 heures **Décembre, 2013**

Professeur: Robert Laganière **Page 1 of 15**

Page	Nombre de Points
PAGE 2	sur 4
PAGE 3	sur 5
PAGE 4	sur 4
PAGE 5	sur 3
PAGE 6	sur 4
PAGE 7	sur 2
PAGE 8	sur 5
PAGE 9	sur 3
PAGE 10	sur 6
PAGE 11	sur 2
PAGE 12	sur 6
PAGE 13	sur 3
PAGE 14	sur 3
TOTAL	50

Prénom: _____

Nom: _____

Numéro d'étudiant: _____

Signature: _____

Aucune documentation permise

Répondre sur le questionnaire.

Aucun appareil électronique permis,
sauf une simple calculatrice non-programmable.

Question 1 [2 points] Quelle est la complexité d'exécution des algorithmes décrits ci-dessous en termes de la notation *Grand O*? Toujours donner le meilleur estimé en fonction de n, le nombre d'entrées.

Algorithm IAMDUMMY(A)
Soit A un tableau de dimension n
for i $\leftarrow$ 0 to $(n-1)$ **do**
    for j $\leftarrow$ 1 to 3^{i} **do**
        A[i] $\leftarrow$ j*j
    end for
end for

a) $O(n \log n)$ *b)* $O(n^3)$ *c)* $O(n^6)$ *d)* $O(3^n)$ *e)* $O(j \bmod n)$ *f)* $O(n2^n)$

Algorithm AmIDUMMY(A)
Soit A un tableau de dimension n
for i $\leftarrow$ 0 to $(n * n * n)$ **do**
    for j $\leftarrow$ 0 to (i/2) **do**
        A[j mod n] $\leftarrow$ j
    end for
end for

a) $O(n^3 \log n)$ *b)* $O(n^3)$ *c)* $O(n^6)$ *d)* $O(3^n)$ *e)* $O(j \bmod n)$ *f)* $O(n2^n)$

Question 2 [2 points] Quel est la complexité au pire cas en termes *Grand O* de l'insertion d'un élément dans la structure de données suivante (en fonction de la taille n de cette structure):

1. Une séquence non-triée (implémentation utilisant un tableau)
   a) O(1) b) O(log n) c) O(n) d) O(n log n)
2. Une file implémentée en utilisant une liste doublement chainée avec un pointeur tête et un pointeur de fin de liste:
   a) O(1) b) O(log n) c) O(n) d) O(n log n)
3. Un arbre binaire de recherche:
   a) O(1) b) O(log n) c) O(n) d) O(n log n)
4. Un arbre 2-4:
   a) O(1) b) O(log n) c) O(n) d) O(n log n)

Question 3 [3 points]

1. (2 pt) Dessiner tous les **arbres binaires de recherche** possibles pouvant stocker les clés 1, 2, 3, et indiquer lesquels seraient des arbres AVL:

2. (1 pt) Dessiner tous les **arbres 2-4** possibles pouvant stocker les clés 1, 2, 3, 4, 5, 6.

Question 4 [2 points] Soit le tableau A suivant représentant un monceau-max:

Index i:	1	2	3	4	5	6	7	8
A[i]:	20	18	19	4	2	3	5	1

a) Quel sera le contenu de A après avoir effectué le retrait de l'élément maximum (**removeMax()**).

Index i:	1	2	3	4	5	6	7
A[i]:							

b) Quel sera le contenu de A après avoir effectué l'insertion de la clé **25** sur l'arbre original A tel que montré plus haut.

Index i:	1	2	3	4	5	6	7	8	9
A[i]:									

Question 5 [1 point] Dessiner l'arbre binaire dans lequel chaque noeud contient 5 lettres qui lorsque affiché en suivant l'ordre in-ordre produit la séquence "ABCDE" et la séquence "ACBED" pour un parcours post-ordre.

Question 6 [1 points] Soit une table de hachage ayant une taille de 31 entrées. Son facteur de charge courant est de 0.74. Afin de réduire ce facteur de charge à 0.5, vous décidez de re-créer une nouvelle table et d'y insérer les éléments de la table originale. Quel devrait être la taille minimale de la nouvelle table afin d'assurer un facteur de charge de 0.5 tout en évitant les collisions le plus possible.

Question 7 [1 point] Une table de hachage de taille 15 a été construite en utilisant un hachage cubique en appliquant la formule suivante $h_j(k) = (k + j^3) mod 15$. Insérer la clé `33` et montrer l'état de la table après cette insertion.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Value	21	16		48	79		36	81		43	10			28	44

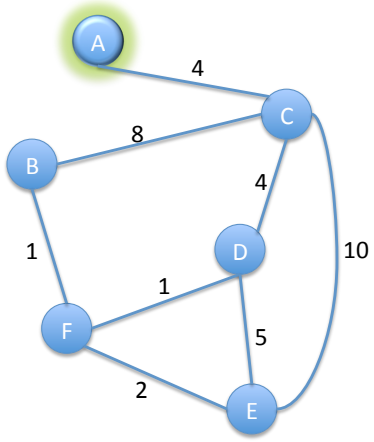
Question 8 [1 point] Une table de hachage de taille 13 a été construite en utilisant le modulo 13 et un sondage linéaire pour la résolution des collisions. En supposant qu'aucun élément n'a été retiré de cette table, donner l'ordre dans lequel les élément aurait pu avoir été insérés.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Value		27	14			31	58	57	34	135			

Question 9 [3 points] Soit le graphe ci-dessous et soit la liste d'adjacence. Considérant l'ordre dans lequel les noeuds apparaissent dans cette liste d'adjacence (suivant l'ordre décroissant du poids de leurs arêtes correspondantes) et en partant du noeud A,

- donner, dans l'ordre, les noeuds qui seront visités par une recherche en profondeur;
- donner, dans l'ordre, les noeuds qui seront visités par une recherche en largeur.

Node	list of adjacent nodes
A	C
B	C F
C	E B D
D	E C F
E	C D F
F	E D B



Question 10 [2 points]

Vous désirez remplacer la valeur associée à chacun des noeuds d'un tore (graphe dans lequel tous les noeuds ont un degré égal à 4) par la somme des poids associés à ses arêtes incidentes. Vous effectuer cette tâche en visitant chacun des noeuds et en additionnant les poids de ses arêtes. Donner la complexité de cet algorithme en spécifiant le nombre **exact** d'additions à effectuer en fonction du nombre de noeuds dans le graphe. *Note:* additionner $a + b + c + d$ requière trois additions.

Question 11 [2 points]

Soit le tableau suivant:

Index	0	1	2	3	4	5	6	7	8	9
Key	6	5	1	3	8	11	2	4	9	10

Appliquer la partition de ce tableau en utilisant une valeur de 6 comme pivot (utiliser l'algorithme montré ci-dessous) et montrer le résultat dans le tableau ci-dessous

Index	0	1	2	3	4	5	6	7	8	9
Key										
Key										

```
private  static void partition(int[] a, int pivot)
{
    int i =0 , j = a.length -1;

    while(true) {

            while(i<a.length && a[i] <= pivot) i++;

            while( j >0 && a[j] > pivot) j--;

            if(i >= j) {
                break;
            } else {
                swap(a,i,j); // echanger les valeur de a aux indices i et j
            }
    }
}
```

Question 12 [1 point] Soit un tableau trié en ordre croissant. Vous désirez maintenant trier ce tableau en ordre décroissant.Quel sera la complexité de ce tri en termes *Grand O* si:

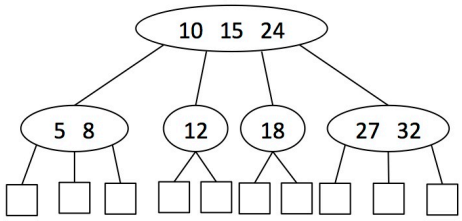
- le tri rapide (Quicksort) est utilisé (avec comme pivot le premier élément de chaque séquence).
- le tri par insertion est utilisé
- le tri par sélection est utilisé
- le tri fusion est utilisé

Question 13 [1 point] Vous désirez trier des nombres se trouvant dans l'intervalle $[0, 999]$ en utilisant un tri lexicographique sur des 3-uplets basé sur les trois chiffres de ces nombres (en base 10). Dans chacune des trois itérations du tri lexicograpique, une implémentation du tri fusion garantissant un tri stable est utilisé.

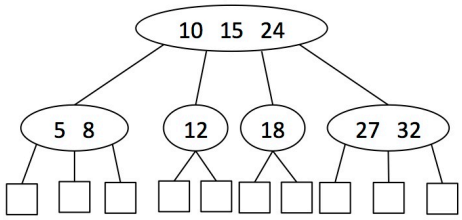
- Afin de procéder à ce tri, devriez-vous débuter par les chiffres des unités ou par celui des centaines? Justifier votre réponse.
- Quel sera la complexité en termes de *Grand O* de l'algorithme décrit ici?
- Serait-il une bonne idée d'utiliser ici le tri rapide (Quicksort) au lieu du tri fusion stable? Expliquer.

Question 14 [3 points] Soit l'arbre 2-4 suivant:

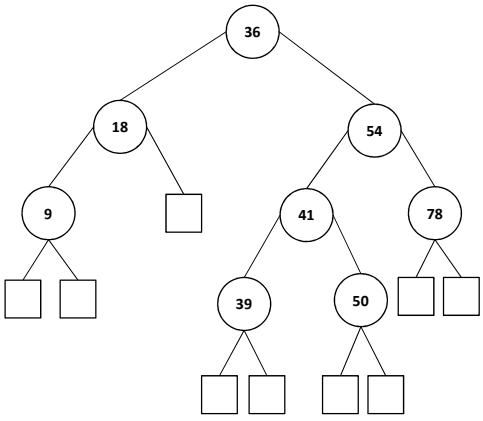
1. (1 point) Insérer 2 dans cet arbre et montrer l'arbre résultant.



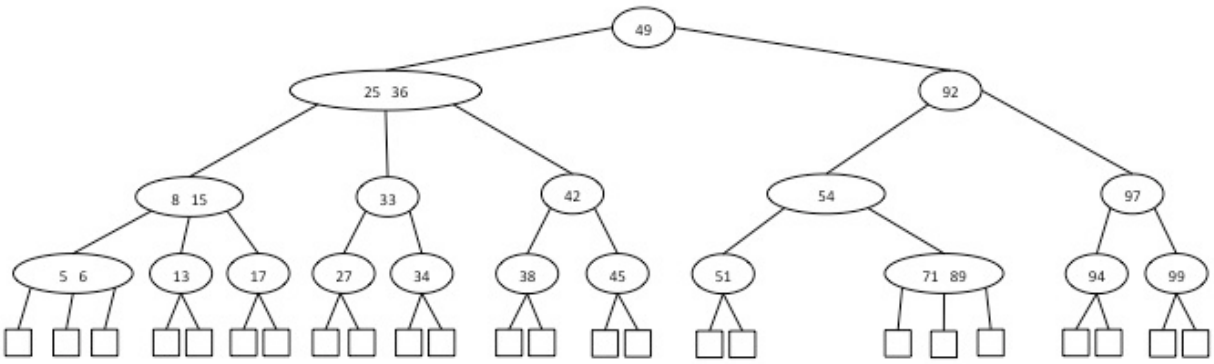
2. (2 point) Retirer 12 de l'arbre ci-dessous et montrer l'arbre résultant.



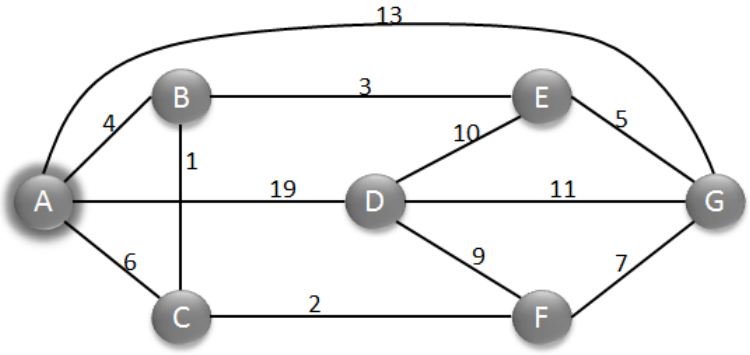
Question 15 [2 points] Insérer la clé 53 dans l'arbre AVL ci-dessous en utilisant l'algorithme discuté en classe et montrer l'arbre résultant..



Question 16 [3 points] Retirer la clé 99 de l'arbre 2-4 suivant et montrer l'arbre résultant.



Question 17 [3 points] Soit le graphe montré ci-dessous. Utiliser l'algorithme de Dijkstra afin de trouver l'arbre des plus courts chemins à partir du noeud A.



Remplir le tableau ci-dessous afin de montrer l'évolution des valeurs de distances associées aux noeuds en cours d'execution.

New vertex	A	B	C	D	E	F	G	New Edge
A	0	4	6	19	∞	∞	13	–

Question 18 [3 points] Pour le graphe de la question 17, utiliser l'algorithme de Kruskal afin de trouver l'arbre couvrant minimal (voir le rappel sur les algorithmes ACM à la fin de l'examen).

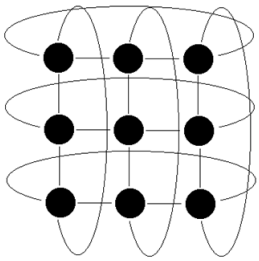
1. Donner la liste des arêtes dans l'ordre où elles seront choisies afin de faire partie de l'arbre couvrant.

2. Quel est le poids total de cet arbre couvrant?

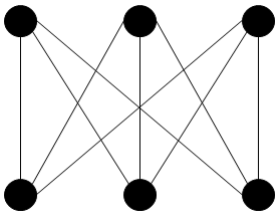
3. Quelle est la complexité de l'algorithme en fonction du nombre de noeuds et du nombre d'arêtes?

Question 19 [2 points] Dans cette question, les graphes sont implémentés en utilisant une liste d'adjacence.

1. Pour un tore 2D (voir la figure ci-dessous), quel est la complexité, en termes de *Grand O* de l'algorithme de Prim-Jarnik pour le calcul de l'arbre couvrant minimum (voir le rappel sur les algorithmes ACM à la fin de l'examen). Exprimer cette complexité en fonction du nombre d'arêtes seulement.



2. Pour un graphe bi-partite complet et balancé (voir la figure ci-dessous), quel est la complexité, en termes de *Grand O* de l'algorithme de Kruskal pour le calcul de l'arbre couvrant minimum (voir le rappel sur les algorithmes ACM à la fin de l'examen). Exprimer cette complexité en fonction du nombre d'arêtes seulement. A noter que pour un tel graphe, les noeuds sont divisés en deux ensembles égaux où chaque noeud d'un ensemble est connecté à tous les noeuds de l'autre ensemble.



Question 20 [3 points] L'algorithme de Boyer-Moore (basé sur les sauts) est utilisé pour trouver un motif P dans une chaine T. Dans le tableau ci-dessous, indiquer les 6 prochaines comparaisons qui seront effectuées par l'algorithme (la première étant déjà indiquée).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
T =	a	-	c	a	n	a	r	y	-	o	n	-	a	-	b	i	n	a	r	y	-	t	r	e	e
P =	b	i	n	a	r	y																			

Comparison #	i	j	$T[i]$	$P[j]$
1	5	5	a	y
2				
3				
4				
5				
6				
7				

Question 21 [3 points]

Vrai ou Faux:

1. L'arbre couvrant d'un graphe G contient tous les sommets de G.
   VRAI **FAUX**

2. Un arbre couvrant minimum avec comme racine le sommet u représente le plus court chemin entre u et tous les autres sommets du graphe.
   VRAI **FAUX**

3. L'algorithme de Kruskals pour trouver une forêt couvrante minimum suppose que le graphe est connecté
   VRAI **FAUX**

Questions courtes:

1. Combien de composantes connectés compte un graphe connecté?

2. Pour un texte de longueur n et un motif de longueur $n/4$, quelle est la complexité en termes de *Grand O* de l'approche *force brute* en fonction de n?

3. Pour un graphe avec n sommets et $2n$ arêtes, quelle est la complexité en termes de *Grand O* de l'algorithme de Dijkstra pour trouver l'arbre des plus courts chemins en fonction de n?

Question 22 [3 points] Soit l'algorithme de Knuth-Morris-Pratt (basé sur l'utilisation des préfixes et suffixes) pour trouver un motif P dans une chaine T.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
T =	b	a	c	c	b	c	a	a	b	a	c	c	b	a	c
P =	b	a	c	c	b	a	c								

- Calculer la fonction de défaillance (*failure function*) pour le motif P en complétant le tableau ci-dessous:

j	0	1	2	3	4	5	6
P[j]	b	a	c	c	b	a	c
F[j]							

- Lorsque le premier désaccord (*mismatch*) se produit entreT[i] et P[j], quelles seront les valeurs de:

i= j= T[i]= P[j]=

- Et pour la comparaison qui suivra juste après, quels seront les valeurs de:

i= j= T[i]= P[j]=

Question 23 [3 points] L'algorithme `estConnecté?` vérifie si un graphe est connecté ou non et retourne un booléen. A l'intérieur de l'algorithme, l'algorithme de Dijkstra pour les plus courts chemins est appelé. Le résultat obtenu par `Dijkstra` devrait ensuite être utilisé afin de déterminer si le graphe est connecté.

L'algorithme `Dijkstra` prend un graphe G et un sommet v comme paramètre d'entrée et retourne une séquence D spécifiant la distance de v à chaque autre sommet dans le graphe.

Compléter l'algorithme `estConnecté`.

Pseudocode (algorithme à compléter)

```
Algorithme estConnecte (G):

    Entree: un graphe pondere G.
    Sortie:  Vrai si le graphe est connecte, faux autrement.

    Soit v un sommet du graphe G.
    connecte = vrai
    D = Dijkstra(G, v)
    // inserer votre code ici









    retourner connecte;
```

Appendice

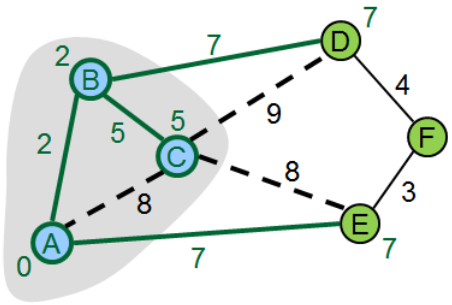


Figure 1: Algorithme de Prim-Jarnik en action

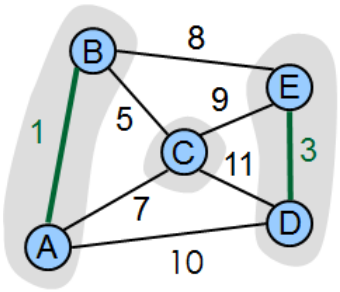


Figure 2: Algorithme de Kruskal en action

Pseudocode pour Dijkstra (pour référence seulement)

```
Algorithme Dijkstra(G, v): // plus courts chemins

    Entree: un graphe pondere G et un sommet v.
    Sortie: une etiquette D[u], pour chaque sommet u de G, tel que D[u]
    est la longueur du plus court chemin de de v a u dans G.

    D[v] = 0 et D[u] = INFINI pour chaque sommet u != v

    Soit Q une file a priorite utilisant D comme cles.

    tant que !Q.estVide () faire
        u = Q.retireMinElement()
        pour chaque sommet z adjacent a u tel que z est dans Q faire
            si D[u] + w((u, z)) < D[z] alors
                D[z] = D[u] + w((u, z))
                Change la cle de z dans Q pour D[z]

    retourner les etiquette D[u] de chaque sommet u.
```