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, 2015**

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

Page	Nombre de Points
PAGE 2	sur 4
PAGE 3	sur 6
PAGE 4	sur 6
PAGE 5	sur 4
PAGE 6	sur 2
PAGE 7	sur 4
PAGE 8	sur 4
PAGE 9	sur 3
PAGE 10	sur 4
PAGE 11	sur 2
PAGE 12	sur 3
PAGE 13	sur 5
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 de données d'entrées.

Algorithm AlgoI(A)
Soit A un tableau de dimension n
for i $\leftarrow$ 0 to $(n-1)$ **do**
    for j $\leftarrow$ 0 to $n/2$ **do**
        A[i] $\leftarrow$ j*j
    end for
end for

a) $O(n \log n)$ *b*) $O(n^2)$ *c*) $O(n^3)$ *d*) $O(2^n)$ *e*) $O(n)$ *f*) $O(\log n)$

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

a) $O(n \log n)$ *b*) $O(n^2)$ *c*) $O(n^4)$ *d*) $O(2^n)$ *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 nouvel élément dans la structure de données suivante (en fonction de n le nombre d'éléments dans cette structure):

1. Une liste doublement chaine non trié
   a) O(1) b) O(log n) c) O(n) d) O(n log n)
2. Un arbre binaire de recherche AVL
   a) O(1) b) O(log n) c) O(n) d) O(n log n)
3. Une table de hachage avec sondage lináire:
   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 [6 points]

1. (3 points) Insérer, dans cet ordre, les clés $1, 2, 3, 4, 5$ dans un arbre AVL initialement vide. Dessiner la structure de l'arbre après chaque insertion.

2. (3 points) Insérer, dans cet ordre, les clés $1, 2, 3, 4, 5$ dans un arbre 2-4 initialement vide. Dessiner la structure de l'arbre après chaque insertion.

Question 4 [6 points] Soit le tableau A suivant représentant un monceau-max:

Index i:	1	2	3	4	5	6	7	8
A[i]:	20	18	16	4	10	3	5	3

1. (2 points) Dessiner la structure de l'arbre binaire correspondant puis donner le parcours préordre, inordre, postordre.

   Dessin de l'arbre:

   Parcours préordre:

   Parcours inordre:

   Parcours postordre:

2. (2 points) 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]:							

3. (2 points) Quel sera le contenu de A après avoir effectué l'insertion de la clé **19** 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 [2 points] Une table de hachage de taille 15 a été construite en utilisant un hachage avec sondage quadratique en appliquant la formule suivante $h_j(k) = (k+j^2) mod\ 15$ où j représente le $j$ième sondage (débutant à $j = 0$). 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		24	10			28	44

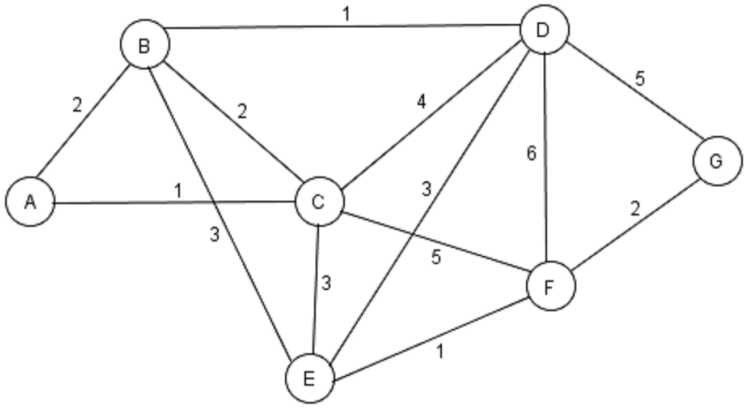
Question 6 [2 points] Une table de hachage de taille 13 a été construite en utilisant la fonction de hachage $h(k) = k\ mod\ 13$ et un sondage linéaire pour la résolution des collisions.

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

1. Quel est le facteur de charge de cette table de hachage?
2. Selon vous, est-ce que la clé `57` a été insérée avant ou après la clé `58`? Justifier votre réponse.

Question 7 [5 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 et en partant du sommet A,

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



1. (1 point) donner, dans l'ordre, les sommets qui seront visités par une recherche en profondeur;

2. (1 point) donner, dans l'ordre, les sommets qui seront visités par une recherche en largeur.

3. (4 points) Utiliser l'algorithme de Dijkstra afin de trouver l'arbre des plus courts chemins à partir du noeud A pour le graphe de la page précédente. Remplir le tableau ci-dessous afin de montrer l'évolution des valeurs de distances associées aux noeuds en cours d'execution. Vous pouvez ajouter des rangées à ce tableau.

New vertex	A	B	C	D	E	F	G	New Edge
A	0	2	1	∞	∞	∞	∞	–

Question 8 [4 points]

Soit un algorithme qui effectue une Partition des éléments d'une liste `S` en utilisant une valeur pivot `P` et produisant trois listes `L`, `E` et `G` pour lesquelles les éléments de `L` sont les éléments de `S` strictement plus petits que le pivot `P`, les éléments de `E` sont les éléments de `S` égaux au pivot `P`, et les éléments de `G` sont les éléments de `S` strictement plus grands que le pivot `P`.

Vous utilisez Partition à l'intérieur de l'algorithme Quicksort (le tri rapide) afin de trier la liste $[1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9]$. Le pivot est choisi au hasard pour chaque appel à Partition. En supposant que les appels récursifs à Quicksort s'arrêtent lorsque la liste à trier contient 0 ou 1 élément...

1. combien d'appels à Partition seront requis *au pire cas* afin de trier cette liste? Bien justifier votre réponse.

2. combien d'appels à Partition seront requis *au meilleur cas* afin de trier cette liste? Bien justifier votre réponse.

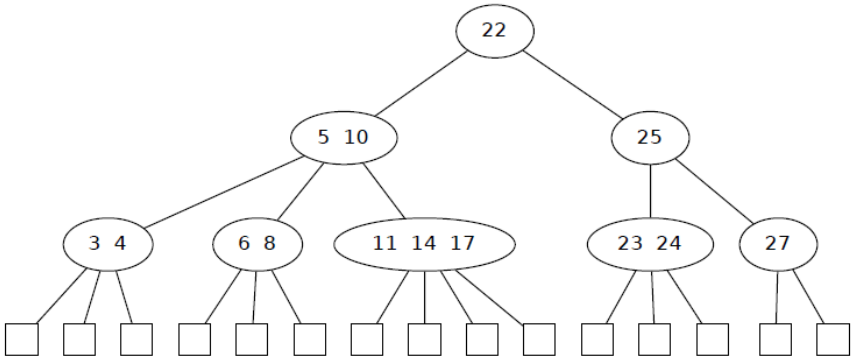
Question 9 [3 points] Soit le tableau `A` suivant avec $n = 10$ éléments:

Index	0	1	2	3	4	5	6	7	8	9
Clé	2	4	58	10	19	17	22	77	54	67

Ce tableau est trié avec l'algorithme de *tri à bulle* (bubble sort) montré ci-dessous. Montrer l'état du tableau à la fin de chaque tour de la boucle `while`.

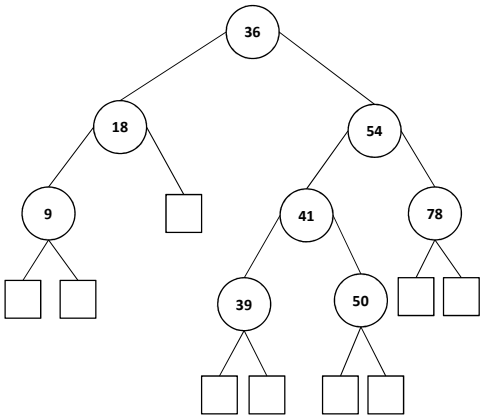
```
j = 0
swapped = true
while (swapped) {
  swapped = false
  j = j+1
  for i=0 to n-j-1 {
     if A[i].key > A[i+1].key {
        tmp= A[i]
        A[i]= A[i+1]
        A[i+1]= tmp
        swapped = true
     }
  }

  // ** montrer l'etat du tableau ici
}
```

Question 10 [4 points] Les questions qui suivent sont indépendantes; les opérations demandées doivent donc être effectuée sur l'arbre 2-4 original tel montré ici.

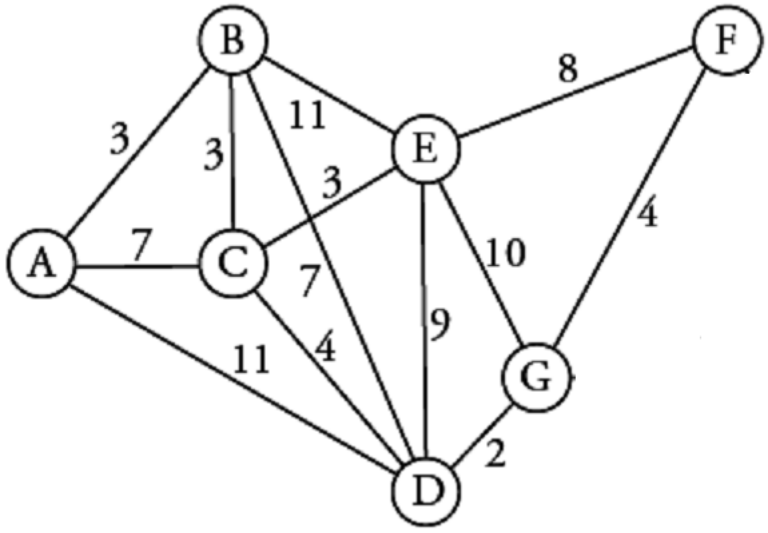


1. (1 point) Insérer 2 dans cet arbre et montrer l'arbre résultant.
2. (1 point) Insérer 15 dans cet arbre et montrer l'arbre résultant.
3. (1 point) Retirer 27 de l'arbre ci-dessus et montrer l'arbre résultant.
4. (1 point) Retirer 22 de l'arbre ci-dessus et montrer l'arbre résultant.

Question 11 [2 points] Insérer la clé 37 dans l'arbre AVL ci-dessous en utilisant l'algorithme montré en classe et montrer l'arbre résultant.



Question 12 [3 points] Pour le graphe ci-dessous, 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 de Kruskal en fonction du nombre de sommets n et du nombre d'arêtes m?

Question 13 [5 points]

Vrai ou Faux:

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

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

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

4. Un graphe régulier constitué de n sommets de degré 4 posséde $4n$ arêtes
   VRAI **FAUX**

5. Dans le parcours en largeur d'un graphe connexe, les sommets et arêtes explorés (i.e. marqués VISITED et EXPLORED) forment un arbre couvrant.
   VRAI **FAUX**

Question 14 [3 points] On veut utiliser l'algorithme de recherche en profondeur `DFS` afin de compter le nombre de composantes connexes dans un graphe. Le pseudo-code ci-dessous prsente le code d'un algorithme classique de recherche en profondeur. Modifier cet algorithme de facon à ce qu'il retourne le nombre de composantes connectées.

Pseudocode (algorithme à modifier)

```
Algorithm DFS(G)
    Entre graphe G
    Sortie etiquetage des aretes de G comme aretes decouvertes ou de retour

    Pour tous les sommets u de G
       setLabel(u, UNEXPLORED)
    Pour toutes les aretes e de G
       setLabel(e, UNEXPLORED)
    Pour tous les sommets v de G
       Si getLabel(v) = UNEXPLORED
          DFS(G, v)

Algorithm DFS(G, v)
    Entre graphe G et un sommet de debut v de G
    Sortie etiquetage des aretes de G a partir de v

    setLabel(v, VISITED)
    Pour toutes les aretes e de G.incidentEdges(v)
    Si getLabel(e) = UNEXPLORED {
        w= opposite(v,e)
        Si getLabel(w) = UNEXPLORED {
            setLabel(e, DISCOVERY)
            DFS(G, w)
        } sinon
            setLabel(e, BACK)
    }
```

Appendice

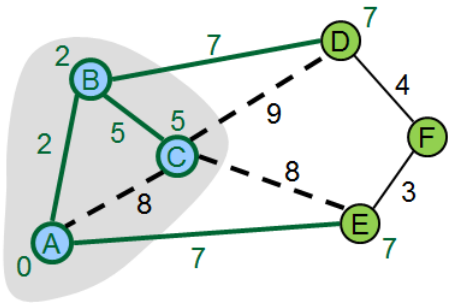


Figure 1: Algorithme de Prim-Jarnik en action

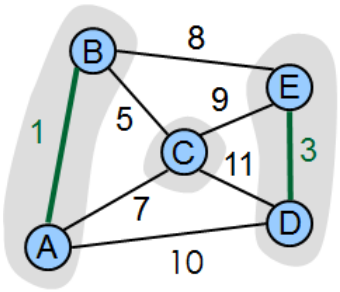


Figure 2: Algorithme de Kruskal en action

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.
```