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

Data Structures and Algorithms

Final Exam

Duration: 3 hours
Professors: L. Moura and R. Laganière

December 12th, 2015
Page 1 of 15

Family name: ________________

Name: ________________

Student number: ________________

Signature: ________________

Closed Books.

Please answer in the space provided (in this questionnaire).

No electronic devices allowed, except for a simple non-programmable calculator.

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

Throughout this exam, unless specified, every logarithm is in base 2.

Question 1 [2 points] What are the running time complexities of the algorithms in the following pseudo-code fragments (in big-Oh notation)? Choose the best big-Oh estimate as a function of n, the size of the input data.

```
**Algorithm** AlgoI(A)
Let A be an array of size $n$
for i ← 0 to $(n-1)$ do
    for j ← 0 to $n/2$ do
        A[i] ← 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)
Let A be an array of size $n$
for i ← 0 to $(n * n)$ do
    for j ← 0 to i do
        A[j mod $n$] ← 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] What is the worst case big-Oh complexity of inserting an element in the following data structures (as a function of the number n of elements stored in the data structure):

1. An unsorted doubly linked list:
   a) O(1) b) O(log n) c) O(n) d) O(n log n)
2. An AVL binary search tree:
   a) O(1) b) O(log n) c) O(n) d) O(n log n)
3. A Hash table where collisions are resolved with linear probing:
   a) O(1) b) O(log n) c) O(n) d) O(n log n)
4. A 2-4 tree:
   a) O(1) b) O(log n) c) O(n) d) O(n log n)

Question 3 [6 points]

1. (3 points) Insert, in this order, the keys 1, 2, 3, 4, 5 into an initially empty AVL tree. Draw the tree after each insertion.

2. (3 points) Insert, in this order, the keys 1, 2, 3, 4, 5 into an initially empty 2-4 tree. Draw the tree after each insertion.

Question 4 [6 points] You are given the following array A which represents a max-Heap:

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

1. (2 points) Draw the corresponding binary tree, and then give the sequence of keys in pre-order, in-order and post-order traversals.

   Draw the tree:

   pre-order traversal: ______________________________

   in-order traversal: ______________________________

   post-order traversal: ______________________________

2. (2 points) Write the content of the array A after performing the deletion of the maximum element (**removeMax()**).

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

3. (2 points) Write the content of the array A after performing the insertion of key **19** to the **original** max-Heap A, which is given on the top of this page.

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

Question 5 [2 points] A hash table of size 15 has been constructed using quadratic hashing by applying the formula $h_j(k) = (h(k) + j^2) \bmod 15$, where the j represents the j-th probe (attempt) and j starts with 0. Insert the key **33** and show the table after this 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] A hash table of size 13 has been constructed using $h(k) = k \bmod 13$ and linear probing for collision resolution.

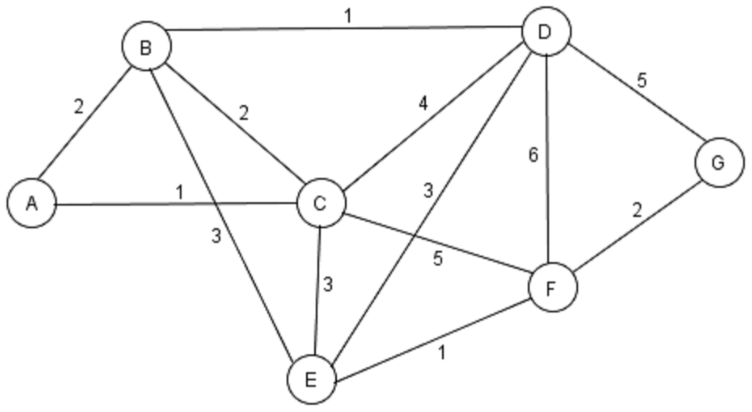
Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Value		27	14			31	58	57	34	135			

1. What is the loading factor of this hash table ?

2. Determine if the key `57` was inserted before or after the key `58`. Justify your answer.

Question 7 [6 points] Consider the graph below and its adjacency list representation. Taking into account **the order of the vertices in each adjacency list** (which follows a decreasing order of edge weights) and starting from vertex A,

Vertex	list of adjacent vertices
A	B C
B	E A C D
C	F D E B A
D	F G C E B
E	B C D F
F	D C G E
G	D F



1. (1 point) give, in order, the list of visited vertices in the case of a **depth-first search**;

2. (1 point) give, in order, the list of visited vertices in the case of a **breath-first search**.

3. (4 points) Use Dijkstra's algorithm to obtain the shortest path tree starting from vertex A for the graph on the previous page. Fill the following table to show the updates of the distances associated to the vertices along its execution. You may add rows to the table.

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

Question 8 [4 points]

Suppose you are given an algorithm that performs a **Partition** of the elements of a list `S` using a pivot value `P` that produces three lists `L`, `E` and `G`, such that the list `L` contains the elements of `S` that are strictly smaller than the pivot `P`, the list `E` contains the elements of `S` that are equal to the pivot `P`, and the list `G` contains the elements of `S` that are strictly greater than the pivot `P`.

You use this **Partition** algorithm inside the **Quicksort** algorithm to sort the list:

$$[1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9]$$

The pivot is randomly chosen at each call to **Partition**. Assume that recursive calls to **Quicksort** stop when the list contains 0 or 1 elements.

1. How many calls to **Partition** are required in *the worst case* in order to sort this list? Justify your answer well.

2. How many calls to **Partition** are required in *the best case* in order to sort this list ? Justify your answer well.

Question 9 [3 points]

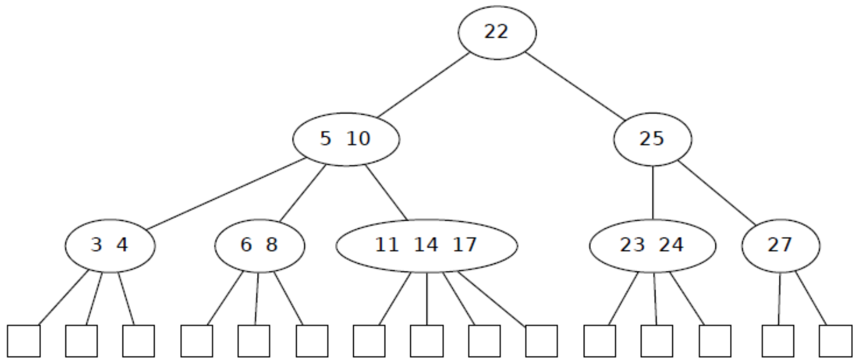
Consider the following array `A` with `n=10` elements:

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

This array is sorted with the **Bubblesort algorithm** given below. Show the state of the array at the end of each **while** iteration.

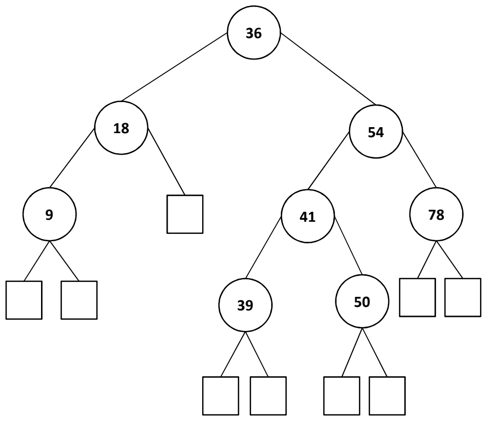
```
j = 0
swapped = true

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
         }
     }
   // *** Show the state of the array here ****
}
```

Question 10 [4 points] The questions bellow are independent; the operations requested must be done on the original 2-4 tree shown here.

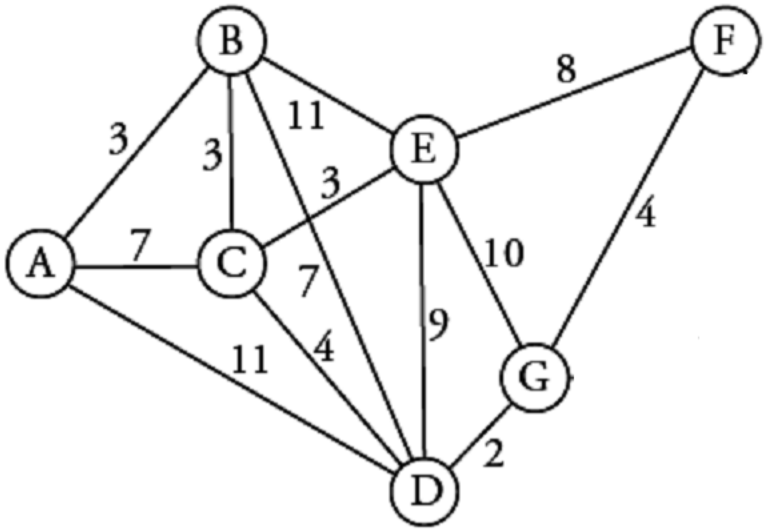


1. (1 point) Insert 2 in the tree of the picture and show the resulting tree.
2. (1 point) Insert 15 in the tree of the picture and show the resulting tree.
3. (1 point) Delete 27 from the tree of the picture and show the resulting tree.
4. (1 point) Delete 22 from the tree of the picture and show the resulting tree.

Question 11 [2 points]
Insert key 37 into the AVL tree below using the algorithm seen in class and show the resulting tree.



Question 12 [3 points] Use **Kruskal's algorithm** in order to find the **Minimum Spanning Tree** of the graph below. Please refer to the last page of the exam for a drawing reminding you of the different minimum spanning tree algorithms.



1. Give the list of edges in the order they are chosen to be part of the minimum spanning tree.

2. What is the total weight of this spanning tree ?

3. What is the complexity (worst-case running time) of Kruskal's algorithm as a function of the number n of vertices and the number m of edges ?
   (Note: different data structures used along the way may affect the running time of any algorithm; this question is concerned with the use of the best data structures to implement this algorithm, explained in this course).

Question 13 [5 points]

True or False:

1. A minimum spanning tree of a connected graph G connects all the vertices of G.
   TRUE **FALSE**
2. A minimum spanning tree with root vertex u represents the shortest paths between u and all the other vertices of the graph.
   TRUE **FALSE**
3. Kruskal's algorithm to find a minimum spanning forest of a graph requires that the graph be connected.
   TRUE **FALSE**
4. A regular graph consisting of n vertices of degree 4 has $4n$ edges.
   TRUE **FALSE**
5. During breadth-first search of a connected graph, the vertices marked "VISITED" and edges marked "DISCOVERY" form a spanning tree of the graph.
   TRUE **FALSE**

Question 14 [3 points] Below, you are given the pseudo-code of the Depth-First Seach (DFS) algorithm, and you must modify it in order to compute the **number of connected components** of the graph. Please make your changes on top of the pseudo-code given below.

Pseudocode (algorithm to modify):

```
Algorithm DFS(G)
Input: graph G
Output: labeling of the edges of G as DISCOVERY edges or BACK edges

       for all vertices u of G
                setLabel(u, UNEXPLORED)
       for all edges e of G
              setLabel(e, UNEXPLORED)
       for all vertices v of G
              if  getLabel(v) == UNEXPLORED
                      DFS(G, v)


Algorithm DFS(G, v)
Input: graph G and a start vertex v of G
Output: labeling of edges of G starting from vertex v

       setLabel(v, VISITED)
       for all edges e of G.incidentEdges(v)
              if  getLabel(e) ==  UNEXPLORED {
                     w = opposite(v,e)
                     if getLabel(w) == UNEXPLORED {
                          setLabel(e, DISCOVERY)
                          DFS(G, w)
                      }
                     else
                          setLabel(e, BACK)
               }
```

Appendix

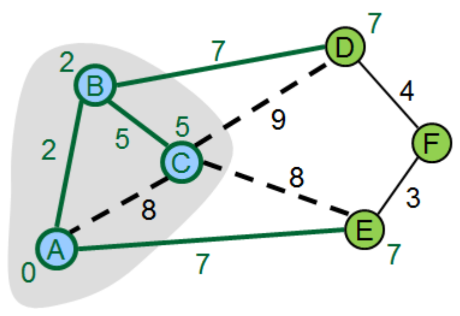


Figure 1: Prim-Jarnik Algorithm in action

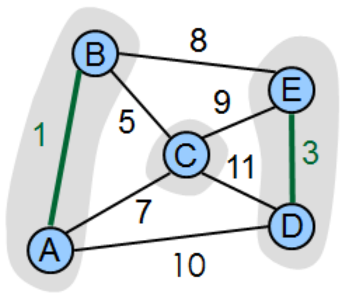


Figure 2: Kruskal's Algorithm in action

Pseudocode for Dijkstra (provided for reference only)

```
Algorithm DijkstraShortestPath(G, v):

    Input: A weighted graph G and a vertex v of G.
    Output: A label D[u], for each vertex that u of G, such that D[u]
    is the length of a shortest path from v to u in G.

    initialize D[v] = 0 and D[u] = INFINITY for each vertex u != v

    let Q be a priority queue that contains all of the vertices of G
    using the D labels as keys.

    while !Q.isEmpty () do
        u = Q.removeMinElement()
        for each vertex z adjacent to u such that z is in Q do
            if D[u] + w((u, z)) < D[z] then
                D[z] = D[u] + w((u, z))
                Change the key value of z in Q to D[z]

    return the label D[u] of each vertex u.
```