



**CSI2110 Fall 2018**

**Final Exam**

**Prof. Lucia Moura and Prof. Robert Laganriere**

**December 10, 14:00 – 3 hours**

**40 points (40% of the final mark)**

LAST NAME: \_\_\_\_\_

First Name: \_\_\_\_\_

Student Number: \_\_\_\_\_

Signature \_\_\_\_\_

Instructions :

**Closed book exam. No calculators allowed. Please fill all multiple choice questions on scantron paper.** There are ??? pages in this exam. In all the questions, when asked for a big-Oh, please provide the best possible value. All the logarithms are given in base 2.

QUESTION	MARKS OBTAINED
Q1-15 multiple choice	/15
Q16	/ 2
Q17	/ 2
Q18	/ 6
Q19	/ 4
Q20	/ 3
Q21	/ 2
Q22	/ 1
Q23	/ 2
Q24	/ 3
TOTAL	/40

**Question 1 [1 point]**

Consider the following piece of code:

```
int mycode(int n) {  
    int i, j, k = 0;  
    for (i = n/2; i <= n; i++)  
        for (j = 2; j <= n; j = j * 2)  
            k = k + n/2;  
    return k;  
}
```

The **RETURN VALUE** of function mycode(n) is

A)	$\Theta(n^2)$
B)	$\Theta(n^2 \log n)$
C)	$\Theta(n^3)$
D)	$\Theta(n^3 \log n)$
E)	None of the above

**Question 2 [1 point]**

Consider the following piece of code that uses a graph G implemented with adjacency lists data structure.

```
int count=0;  
  
for each (v: G.vertices()) { // run through all vertices v of the graph  
    for each (e: G.incidentEdges(v)) // run through all edges incident to v  
        count++;  
}
```

If n is the number of vertices and m is the number of edges, the function that most precisely describes the running time of this piece of code is:

A)	$O(n^3)$
B)	$O(n^2)$
C)	$O(n*m)$
D)	$O(n+m)$
E)	None of the above

**Question 3 [1 point]**

Which of the first 4 statements is FALSE?

- A)  $2^{\log n}$  is  $O(2^n)$
- B)  $n^2$  is  $O(2^n)$
- C)  $2^{\log n}$  is  $O(n^2)$
- D)  $n^2 + 2^n$  is  $\Omega(n)$
- E) None of the above is false.

**Question 4 [1 point]**

For each of the following data structures: Heap, AVL tree, Hash Table, associate the Abstract Data Type that it implements:

	Heap	AVL tree	Hash table
A)	Priority queue	Map	Map
B)	Priority queue	Priority queue	Map
C)	Map	Map	Queue
D)	Queue	Priority queue	Map
E)	None of the above.		

**Question 5 [1 point]**

Which of the following alternatives is a true completion of the phrase:

The worst case running time of searching for a key in an AVL tree is (i) \_\_\_\_\_ but the worst case running time of searching for a key in a binary search tree is (ii) \_\_\_\_\_

- A) (i)  $\Theta(\log n)$  (ii)  $\Theta(n)$
- B) (i)  $\Theta(\log n)$  (ii)  $\Theta(n \log n)$
- C) (i)  $\Theta(n)$  (ii)  $\Theta(\log n)$
- D) (i)  $\Theta(n \log n)$  (ii)  $\Theta(n)$
- E) None of the above.

**Question 6 [1 point]** The worst case running times of the following sort algorithms are:

	Insertion sort	Mergesort	Quicksort
A)	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$
B)	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n \log n)$
C)	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n \log n)$
D)	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n^2)$
E)	None of the above.		

**Question 7 [1 point]** We use an array representation for a binary tree which stores keys:

I)	1	3	2	8	10	4	5	9
II)	10	70	80	110	90	-	-	120
III)	5	7	8	10	11	14	15	12

Which of the above represents a **min heap**?

- A) I only
- B) I and II only
- C) I and III only
- D) II and III only
- E) None of the above is the correct answer.

**Question 8 [1 point]**

Consider the **max heap** that uses the following array  $a[] = [10, 5, 3, 1, 4, 2]$  to store its elements. Which is the array after one `removeMax()` operation?

- A) [5, 3, 1, 4, 2, -]
- B) [2, 5, 3, 1, 4, -]
- C) [5, 4, 3, 1, 2, -]
- D) [5, 2, 3, 1, 4, -]
- E) [5, 4, 3, 2, 1, -]

**Question 9 [1 point]**

Consider the **max heap** that uses the following array  $a[] = [10, 5, 3, 1, 4, 2]$  to store its elements. Which is the array after inserting key 9 into the heap?

- A) [10,5,3,1,4,2,9]
- B) [10,9,5,1,4,2,3]
- C) [10,9,1,4,5,2,3]
- D) [10,5,9,1,4,2,3]
- E) none of the above

**Question 10 [1 point]** Consider the following Hash table where insertions are done using the hash function  $h(k) = k \bmod 7$ , and collisions are resolved with **quadratic probing**.

0	1	2	3	4	5	6
	15	1	8		5	

Regarding the ordering of insertion in this table, which answer is CORRECT?

- A) Key 1 was the last key to be inserted.
- B) Key 8 was the last key to be inserted.
- C) Key 15 was the last key to be inserted.
- D) It is impossible to determine which was the last key among 8 and 5.
- E) None of the above is correct.

**Question 11 [1 point]** Consider the following Hash table where insertions are done using the hash function  $h(k) = k \bmod 7$ , and collisions are resolved with **linear probing**.

0	1	2	3	4	5	6
	15	1	8		5	

What is the **average number of probes A** for a search in this table?

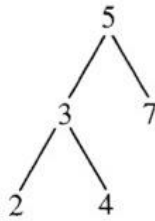
- A)  $A=1$
- B)  $1 < A < 2$
- C)  $A=2$
- D)  $A > 2$
- E) None of the above is correct.

**Question 12**

Suppose you **insert element 2** in the table given in Question 11. After this, you search for element 3. How many table positions must be probed until you conclude element 3 is not in the table?

- A) 2
- B) 3
- C) 4
- D) 5
- E) 6 or more

**Question 13 [1point]** Consider an algorithm printed the nodes of the tree above in the order: 5 3 7 2 4. Which algorithm this may be?



- A) Preorder traversal of the tree.
- B) Postorder traversal of the tree.
- C) Depth first search starting in node 5
- D) Breadth first search starting in node 5
- E) None of the above is correct.

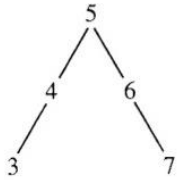
**Question 14 [1 point]** Consider the torus graph, a graph where every vertex has exactly 4 adjacent vertices. The graph has **N vertices** and is represented using **adjacency list data structure**. Calculate the number of edges **M** as a function of **N**. The time complexity of running Depth-First Search (DFS) as a function of **N** is

- A)  $\Theta(\log N)$
- B)  $\Theta(N)$
- C)  $\Theta(N \log N)$
- D)  $\Theta(N^2)$
- E) None of the above is correct.

**Question 15 [1 point]** Consider procedure `stableSort(String [] A, int i)` which sorts a global array of strings with 4 letters by simply using the letter at position *i* of each string *S* as the key for sorting in increasing order of *S[i]*. Moreover we know that the algorithm is **stable**, that is, if elements have the same key, their order in the output will be the same order they arrived in the input. Which of the following algorithms sorts the strings in array *A* in alphabetical order?

- A) `for (i=0 ; i<=4 ; i++) stableSort(A,i)`
- B) `for (i=3 ; i>=0 ; i--) stableSort(A,i)`
- C) `for (i=0 ; i<=4 ; i++)`  
     `for (j=0 ; j<i ; j++)`  
         `stableSort(A,i)`
- D) `for (i=0 ; i<=log n ; i++) stableSort(A,i)`
- E) None of the above.

**Question 16 [ 2 marks]** Consider the following AVL tree where dummy nodes exist but have not been drawn.



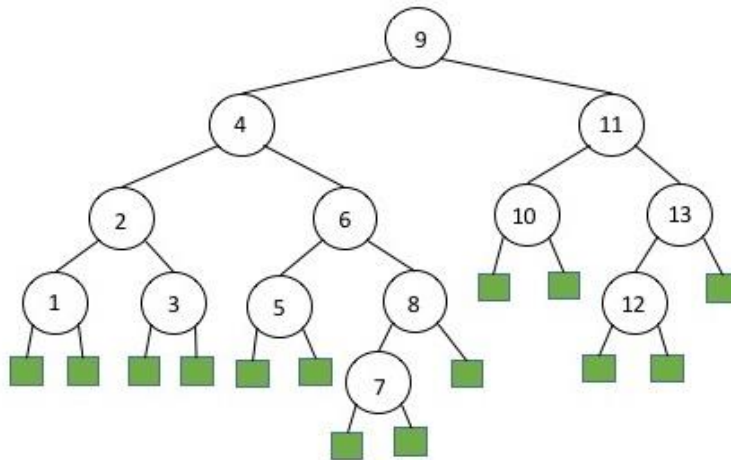
The following insertions are done, one at a time in this order, into the AVL tree above:  
Insert 9, Insert 10, Insert 11, Insert 8. Show the tree after each insertion:

After Insert 9	
After Insert 9, Insert 10	

<p>After</p> <p>Insert 9, Insert 10, Insert 11</p>	
<p>After</p> <p>Insert 9, Insert 10, Insert 11, Insert 8.</p>	



**Question 17 [2 marks]** Consider the following AVL tree.



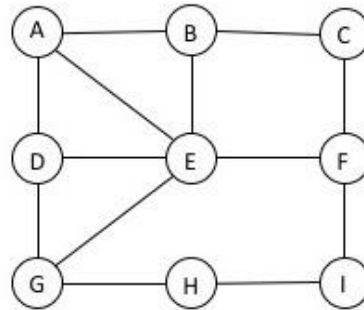
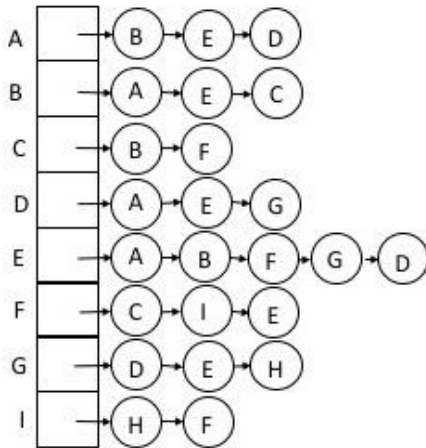
Show the relevant steps (tree transformations) after the operation delete 11.  
Whenever a rebalancing operation is performed, identify the relevant nodes and sub-trees involved in the rebalancing operation (before and after the rebalancing take place)

(page intentionally left blank)

**Question 18 [6 points=3+3]** Consider the graph given below and consider the given adjacency list representation of the graph.

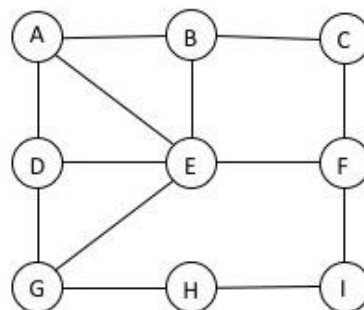
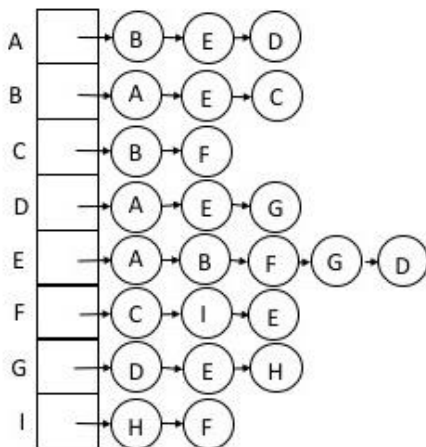
1) Taking into account the order of the vertices in each adjacency list, apply **depth-first search** starting from **vertex A**, draw on the graph:

- The order in which the nodes are visited (mark the number beside the vertex)
- Mark each edge DISCOVERY (solid) or BACK (dashed) showing the direction in the edge when it was labelled.



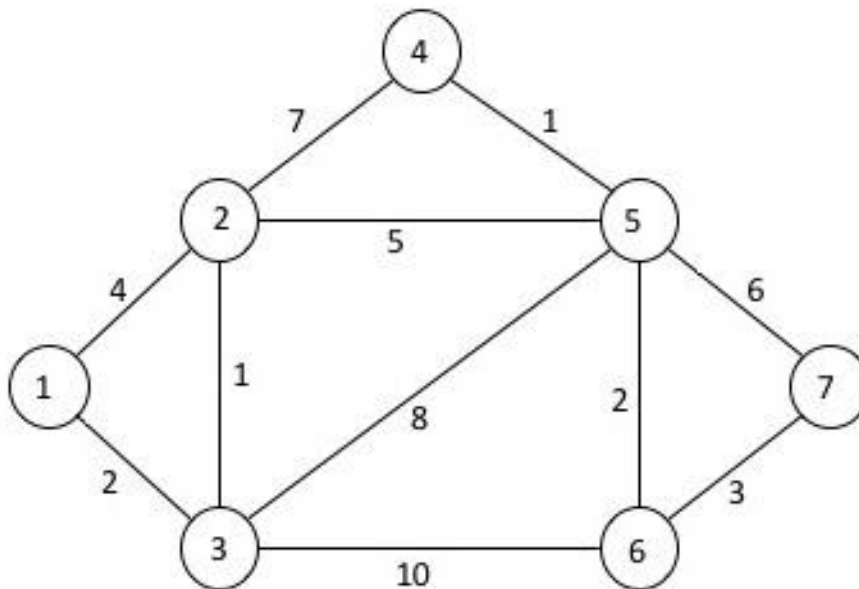
2.) Taking into account the order of the vertices in each adjacency list, apply **breadth-first search** starting from **vertex A**, and draw on the graph:

- the order in which the nodes are visited (mark the order number beside the vertex)
- Mark each edge DISCOVERY (solid) or CROSS (dashed) showing the direction in the edge when it was labelled.



**Question 19 [4 points]**

Use Dijkstra's algorithm to obtain a tree of shortest paths for the graph below starting from vertex 1 :



Display below:

- Vertices in the order they enter the cloud (tree of shortest paths),
- Edges in the order they enter the tree of shortest paths (tree edges),
- Final array with distances **dist**, where **dist[v]** shows the distance between origin (vertex 1) and vertex v

Vertices (in order)

Edges (in order)

	dist
1	
2	
3	
4	
5	
6	
7	

**Question 20 [3 points] Mergesort**

Simulate a full execution of the in-place Mergesort algorithm for the given array below. Suppose that at the end of the “merge” step, the algorithm prints a line with the current contents of the full array.

Show each printout of this algorithm, highlight the part of the array where the merge has been done. The number of blank arrays below may be more or less than the amount you need to show (use the back if more space is needed).

30	10	50	20	15	7	2	12
----	----	----	----	----	---	---	----

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

**Question 21 [ 2 points]** The following program is an implementation of the Quicksort algorithm.

```
private static void doQuickSort(int[] data, int low, int high) {

    int i = low;
    int j = high;

    // beginning of Partition
    int pivot = data[low];

    while (i <= j) {
        while (data[i] < pivot) {
            i++;
        }
        while (data[j] > pivot) {
            j--;
        }
        if (i <= j) {
            swap(data, i, j);
            i++;
            j--;
        }
    }
    // *** end of Partition ***
    if (low < j)
        Sort.doQuickSort(data, low, j);
    if (i < high)
        Sort.doQuickSort(data, i, high);
}
```

The most important part of the quicksort strategy for sorting is the partition of the array into two parts using a pivot element. The value of the chosen pivot determines the efficiency of the sorting. Consider the following array:

18	35	12	20	15	7	2	50
----	----	----	----	----	---	---	----

- a) Show what will be the state of the array at the end of the first partition (i.e. at the point indicated in the source code by the \*\*\*).

--	--	--	--	--	--	--	--

- b) Now if you change the line `int pivot = data[low];` by the following:

```
int pivot = data[high];
```

Will you get the same result? If not draw the state of the new array

--	--	--	--	--	--	--	--

**Question 22 [1 points]**

Consider an array of four equal-value elements. [X,X,X,X]

- A) how many comparisons will be performed if you try to sort this array using insertion sort? \_\_\_\_\_
- B) How many comparisons will be performed if you try to sort this array using Selection sort? \_\_\_\_\_
- C) How many merge operations will be performed if you try to sort this array using Merge sort? \_\_\_\_\_

**Question 23 [2 points]**

A programmer wrote a wrong implementation of a binary search method for a sorted array data. He/she tested the function and it finds the right value. However the method seems to be slower than expected.

- a) Examine the code and evaluate the worst case complexity of the implemented algorithm.  
The algorithm runs in  $O(\text{_____})$
- b) Binary search in a sorted array should be  $O(\log N)$ , modify the code below such that it now correctly implements this algorithm. Note: you have the right to change a maximum of three lines in this code.

```
// search if a given integer `value` in a sorted array `data` of size N>0
public int search(int value, int[] data, int N) {

    int inf=0, sup= N-1;
    int mid= (inf+sup)/2;

    while (data[mid]!=value && inf<sup) {

        if (data[mid]<value) {
            inf++;
        } else {
            sup--;
        }

        mid= (inf+sup)/2;
    }

    if (data[mid]==value)
        return mid;
    else
        return -1;
}
```

**Question 24 [3 points]**

Suppose you are given an array A of student names and grades (each grade is an integer number between 0 and 10). Write an algorithm to sort this array in increasing order of grades in time  $O(N)$ , where N is the number of students.

At the end of the algorithm the input array must be sorted.

You may and should use other auxiliary memory to store the buckets and its contents.

Hint – implement the BucketSort method.

Reminder: BucketSort creates an array of lists indexed by the values the key can take, insert each record on the list indexed by the value of its key, then use these lists to produce a sorted output.

Write your algorithm in Java-like pseudocode or Java code.

```
public interface Student {  
    int grade(); // key used for sorting  
    String StudentName();  
}  
void BucketSort (Student[] A) {
```



(page intentionally left blank)