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 Fall 2018, Prof. L. Moura and Prof. R. Laganière

Midterm Exam

November 4, 10h00 - 120 minutes

30 points (30% of the final mark)

Closed book

LAST NAME: _____SOLUTION_____

First Name: _____ Student Number: _____

Signature _____

Instructions :

Closed book exam. No calculators allowed. There are 10 pages and 14 questions in this exam.

In all the questions, when asked for a big-Oh, please provide the best possible value.

All the logarithms where the base is not shown are given in base 2.

QUESTION	MARKS OBTAINED
Questions 1-7 (multiple choice)	/7
Question 8	/3
Question 9	/3
Question 10, 11	/4
Question 12	/5
Question 13	/3
Question 14	/5
TOTAL	/30

Question 1 [1 point] The following code uses a stack implemented using a singly linked list. Give the **time complexity** of the following piece of code in terms of big-Oh as a function of N.

```
Stack<Integer> myStack = new Stack<Integer>();

for (int i=0; i< N; i++) {
     myStack.push(i);
}
int tot=0;
for (int i=0; i< N; i++) {
     tot += myStack.pop();
}
```

A) O(1) B) O(log N) C) O(N) D) O(N log N) E) $O(N^2)$

Question 2 [1 point] The following code uses a sorted array `arr` with `N` elements and uses the well-known **binary search** method that returns the index of the array that contains the `key` or –1 if the `key` is not found. Give the **time complexity** of the following piece of code in terms of big-Oh as a function of `N`.

```
int j; int count=0;
int N = arr.length;

for (int i=0; i< N/2; i++) {
     int key=2*i;
     j=binarySearch(arr,key,0,N-1);
     if (j!=-1) count++;
}
```

A) O(1) B) O(log N) C) O(N) D) O(N log N) E) $O(N^2)$

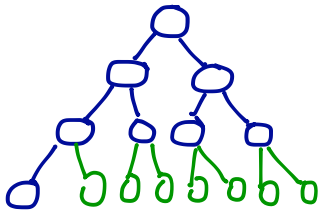
Question 3 [1 point] Give the **time complexity** of the following piece of code in terms of big-Oh as a function of `N`.

```
int i=1;
while (i< N){
    i = i*2;
    print(i);
}
```

A) O(1) B) O(log N) C) O(N) D) O(N log N) E) $O(N^2)$

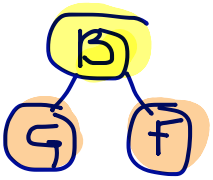
Question 4 [1 point]
What is the minimum and maximum number of nodes in a **complete binary tree** with height 3 ?

A) min= 4, max= 8
B) min= 4, max= 15
C) min= 8, max= 15
D) min= 8, max= 16
E) None of the above.



Question 5 [1 point]
Consider a binary tree containing letters, such that the **in-order** traversal prints G,C,A,D,B,F,E and its **pre-order** traversal prints B, G, A, C, D, F, E. What does a **post-order** traversal prints?

A) G, A, C, B, E, F, D
B) D, C, F, G, A, B, E
C) E, F, D, C, A, G, B
D) C, D, A, G, E, F, B
E) None of the above.



Question 6 [1 point]
Consider a **priority queue** with `n` elements, implemented using a **min-heap**.
Which of the following answers give the time complexity of the operations
`insert` (insertion of an arbitrary element into the priority queue)
`removeMin` (deletion of the smallest element in the priority queue)
`min` (return the smallest element without modifying the priority queue), respectively:

A) Θ (1), Θ (l), Θ (1)
B) Θ (log n), Θ (log n), Θ (1)
C) Θ (log n), Θ (log n), Θ (log n)
D) Θ (n), Θ (n), Θ (n)
E) None of the above.

Question 7 [1 point]

In which of the following data structures containing `n` elements, **deleting the element with minimum key** has worst-case complexity `O(log n)`?

A) a sorted array (sorted in increasing order)
B) an unsorted doubly linked list
C) a binary search tree
D) a queue
E) none of the above.

all take O(n)

Question 8 [3 points]
Consider a MAP abstract data type and various different data structures that implement it.

Assume we employ the most efficient known algorithm for each operation in the given data structure.

The MAP operations are detailed below:

get(k): searches for key k and returns the value v associated to key k, if such entry exists; otherwise returns null.

put(k, v): searches for key k, and if M does not have an entry with key k, then adds (k,v) and returns null; otherwise it replaces with v the value of the entry with key equal to k and returns the old value.

remove(k): searches for key k and removes from M the entry (v,k) and returns its value v; if M has no such entry, then it returns null.

For each data structure, indicate the big-Oh (worst case) complexity of each of the MAP operations as a function of *n*, the number of elements in the MAP:

	get(k)	**put(k,v)**	**remove(k)**
unsorted linked list	$O(n)$	$O(n)$	$O(n)$
sorted array	$O(\log n)$	$O(n)$	$O(n)$
Binary Search Tree	$O(n)$	$O(n)$	$O(n)$

Question 9 [3 points] Give an asymptotic upper bound using **big-Oh** for the following functions. Use the tightest and simplest upper bound. Briefly justify your answers.

a) $f(n) = 1 + n + 3n^2 + 3n^3$ $\leq 1n^3 + 1n^3 + 3n^3 + 3n^3 = 8n^3$

$n \geq 1$

$O(n^3)$

b) $f(n) = log_2(2^n \cdot n^3)$ $= \log_2 2^n + \log_2 n^3$

$= n + 3 \log n \leq 1n + 3n = 4n$

$n \geq 1$

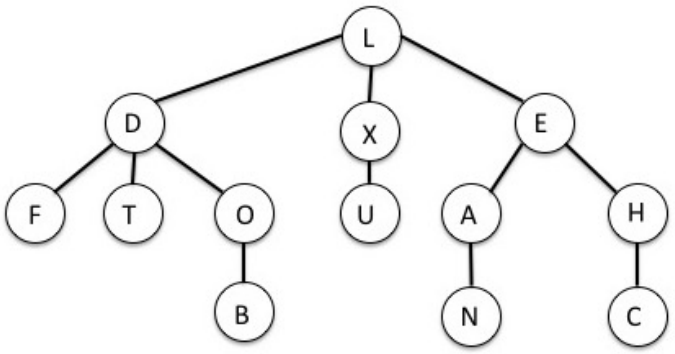
$O(n)$

c) $f(n) = \sum_{i=0}^{n} 2^i = 1 + 2 + 4 + \cdots + 2^n$ $= 2^{n+1} - 1 = 2 \cdot 2^n - 1$

$\leq 2 \cdot 2^n$ $n \geq 1$

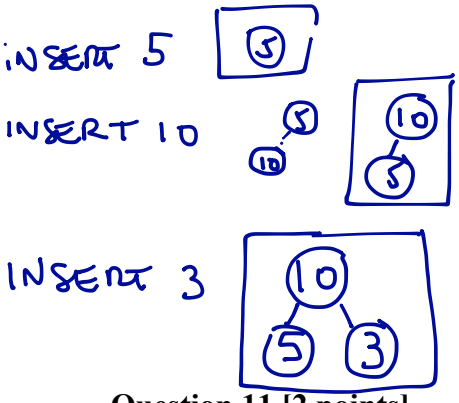
$O(2^n)$

Question 10 [2 points] Print values for the pre-order and post-order traversal for the tree:



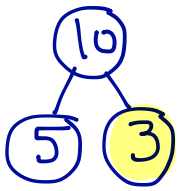
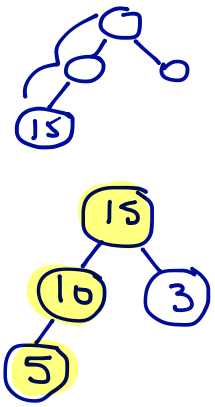
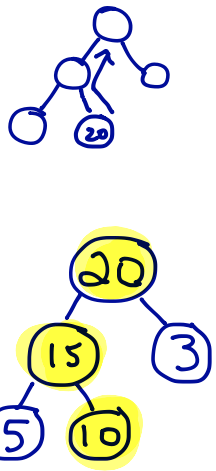
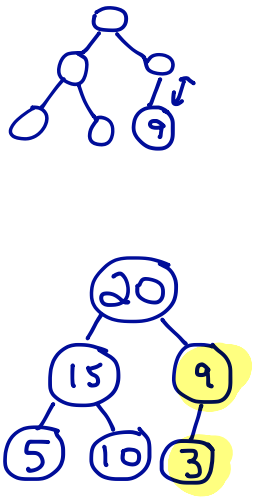
pre-order : L D F T O B X U E A N H C

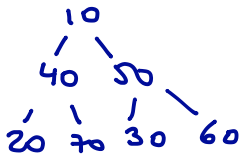
post-order: F T B O D U X N A C H E L



Question 11 [2 points]

Draw the **max-heap** obtained after **inserting** each of the following keys (in this order) into an empty heap : 5, 10, **3, 15, 20, 9 (**show the heap after each of the bold numbers are inserted)



Question 12 [5 points] Given the following array:

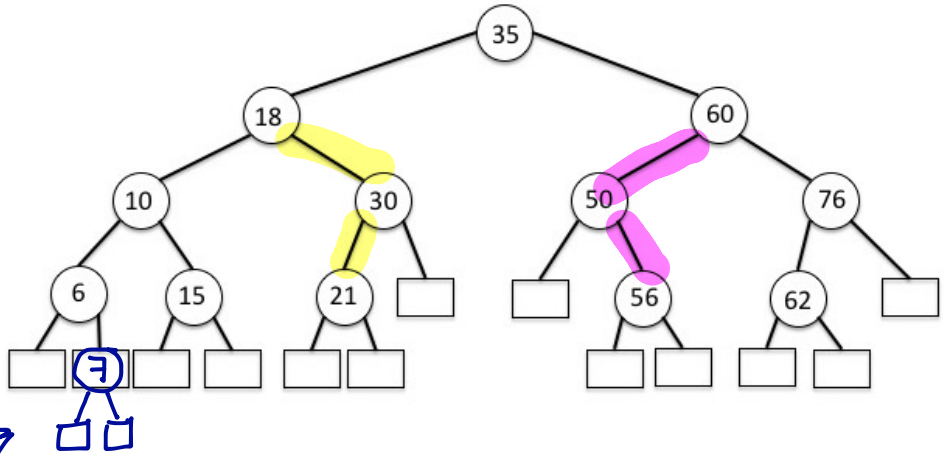
0	1	2	3	4	5	6
10	40	50	20	70	30	60

Sort this array **in-place** in increasing order using the **HeapSort algorithm**.
Phase 1: Bottom up maxheap contruction: **display array after each downheap operation.**
Phase 2: Phase that incrementally builds sorted array, reducing the heap part:
underline the sorted part at each step (the sorted part size must increase by 1 at each new array shown, while the heap part decreases by 1), do not show intermediate steps.
You may need more or less arrays than the ones provided below.
Show the array after each BIG STEP of each phase (not at each simple swap!).
Draw a line separating Phase 1 from Phase 2.

0	1	2	3	4	5	6
10	40	60	20	70	30	50
10	70	60	20	40	30	50
70	40	60	20	10	30	50
60	40	50	20	10	30 |	70
50	40	30	20	10 |	60	70
40	20	30	10 |	50	60	70
30	20	10 |	40	50	60	70
20	10	30 |	40	50	60	70
10 |	20	30	40	50	60	70

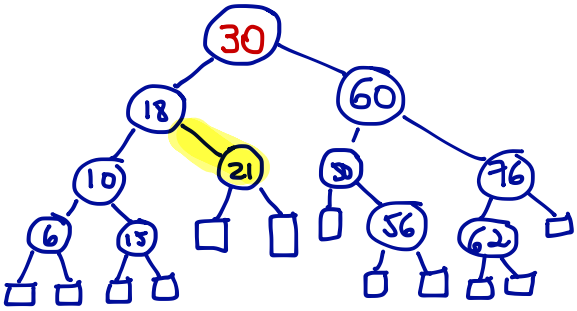
Question 13 (3 points =1+2)

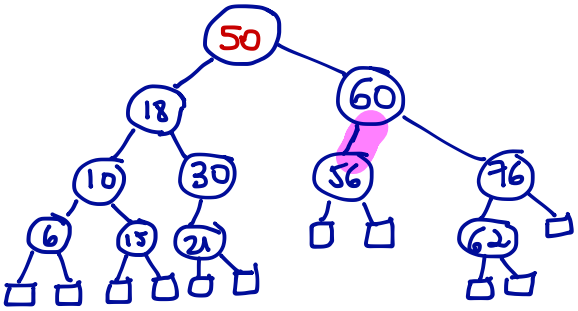
Consider the **binary search tree** below :



a) Show the tree after inserting 7. (To save time you can draw on top of the figure above)

b) Show the tree after deleting key 35 from the original tree. TWO OPTIONS





Question 14 (5 points) Consider a **binary search tree** that provides the following methods :

`root()`	returns the root node of the tree or null if the tree is empty
`parent(p)`	returns the parent node of node p or null if p is the root
`leftChild(p)`	returns the left child node of node p or null if p has no left child.
`rightChild(p)`	returns the right child node of node p or null if p has no right child
`key(p)`	returns the key (integer) stored in node p
`isInternal(p)`	returns true if and only if p is an internal node
`isExternal(p)`	returns true if and only if p is an external node
`size()`	returns the number of nodes stored in the tree

You may give the required algorithm in pseudocode or Java program excerpt. You may use the methods listed above and design any auxiliary method you wish.

a) (3.5 points) Give a pseudocode or Java code for a method that computes the maximum value of a key stored in the binary search tree.
Example : for the tree in page 8 this method would return 76.

```
int maxKey() { // complete the code here.
```

SOLUTION WITH DUMMY

```
p=root();
if (right(p)==null)
    return -MAXINT
while (right(p)!=null)
    p=right(p);
return key(parent(p))
```

SOLUTION WITHOUT DUMMY:

```
p=root();
if (p=null)
    return -MAXINT
while (right(p)!=null)
    p = right(p);
return key(p);
```

b) (1.5 point) If the tree has n nodes and height $h=2 \log n$, what is the big-Oh running time of your `maxKey()` algorithm as a function of n ? O (log n)

(page intentionally left blank)