CMPT 225 D100, Fall 2025

Final Exam - SOLUTIONS
December 5, 2025

Name_____

SFU ID: |__|__|__|__|__|__|__|__|__|

Problem 1	
Problem 2	
Problem 3	
Problem 4	
TOTAL	

Instructions:

1. The duration of the exam is 180 minutes.
2. Write your full name and SFU ID **clearly**.
3. This is a closed book exam, no calculators, cell phones, or any other material.
4. The exam consists of four (4) problems, each worth 25 points
5. Write your answers in the provided space.
6. There is an extra page at the end of the exam. You may use it if needed.
7. You may write helper functions if needed.
8. Explain all your answers.
9. Really, explain all your answers.

Good luck!

Problem 1 (Java syntax, Big-O notation) [25 points]

A. (5 points) Prove that the running time of the function fun_A(n) is O(n log(n)).

```
int fun_A (int n) {
    int sum = 0, i, j;
    for (int i=0; i<n; i++)
        for (int j=1; j<i; j=j*2)
            sum += 1;
    return sum;
}
```

Answer: The outer loop has n iterations.
In each iteration of the outer loop, the inner loop makes $\log_2(i) < \log_2(n)$ iterations, each iteration running in O(1) time.
Therefore, the total running time is $n^* \log_2(n)^*O(1) = O(n \log(n))$

B. (5 points) Prove that the running time of the function fun_B(n) is O(n).

```
int fun_B (int n) {
    int sum = 0, i, j;
    for (int i=1; i<n; i*=2)
        for (int j=1; j<i; j++)
            sum += j;
    return sum;
}
```

Answer: The outer loop has log(n) iterations: i=1,2,4,8,16…2^k for k such that up 2^k<n.
In each iteration of the outer loop, the inner loop runs for i steps.
Therefore, the total running time is 1+2+4+8+16+... + 2^k + up to n.
This is a geometric series, and the sum is at most 2^{k+1}-1 < 2n = O(n)

Consider the function fun_C(a,b).

```
static void printAB(int a, int b) {
   for (int i = a; i <= b; i++)
       System.out.print(i + " ");
   System.out.println();
}
static void fun_C(int a, int b) {
   if (a < b) {
       printAB(a, b);
       int q = (b-a) / 2;
       fun_C(a, a+q);
       fun_C(a+q+1, b);
   }
}
```

C. (7 points) What will be the output of fun_C(2,6)? List all recursive calls of the program.

Answer:

- **2 3 4 5 6**
- **2 3 4**
- **2 3**
- **5 6**

List of recursive calls:

We start with fun_C(2,6). It computes q =(6-2)/2 = 2

- prints **2 3 4 5 6**
- makes recursive calls: ***fun_C(2,4)*** and ***fun_C(5,6)***
    - fun_C(2,4) - computes q = (4-2)/2 = 1.
        - prints **2,3,4**
        - makes recursive calls ***fun_C(2,3)*** and ***fun_C(4,4)***
            - fun_C(2,3)
                - prints **2,3**
                - calls *fun_**C(2,2)**, **fun_C(3,3)*** - exit immediately
            - fun_C(4,4) exits immediately
    - fun_C(5,6) - computes q = (6-5)/2, which is 0 after casting into int.
        - prints **5,6**
        - recursive calls ***fun_C(5,5)***, ***fun_C(6,6)*** - both exit immediately

D. (8 points) What is the running time of fun_C(0,n) as a function of n?
Use big-O notation to express your answer.

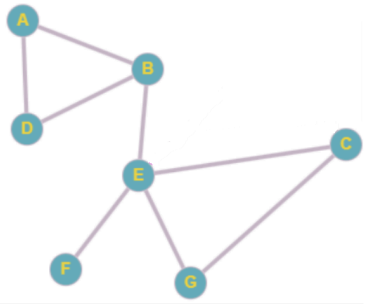
Answer: For general inputs (a,b) define n = b-a+1.
Then the running time on input (a,b) is T(n) = O(n) + 2T(n/2).
By Master Theorem we have T(n) = O(n log(n))

Problem 2 (Graphs) [25 points]

A. (5 points) Draw the graph defined by this adjacency matrix.

	A	B	C	D	E	F	G
A	0	**1**	0	**1**	0	0	0
B	**1**	0	0	**1**	**1**	0	0
C	0	0	0	0	**1**	0	**1**
D	**1**	**1**	0	0	0	0	0
E	0	**1**	**1**	0	0	**1**	**1**
F	0	0	0	0	**1**	0	0
G	0	0	**1**	0	**1**	0	0

ANSWER:



B. (20 points) A map is given as a nXn 2d array of ints. The cells of the map are connected *horizontally/vertically* (not diagonally).

Zeros correspond to *water* and positive numbers correspond to *land*.

Each island consists of connected cells of land (positive numbers).

For example, suppose grid = {

{5,0,0,0,0,0},
{4,0,2,5,0,0},
{0,0,1,0,0,0},
{0,3,2,4,0,0},
{0,6,0,0,2,0},
{0,0,0,1,8,0}};

This map has 3 islands, colored with blue, red, and green. Note that the green and red islands contain cells that touch on a diagonal, and we do not consider it as connected.

Write a function that gets a nXn map, and coordinates (x,y), where 0≤x,y<n.

- If map[x][y] == 0, the function returns 0.
- Otherwise, it returns the sum of all numbers on the island containing (x,y).

In the example above,

- The sum on the blue island is 5+4=9,
- The sum on red island is 2+5+1+3+2+4+6=23
- The sum on the green island is 2+1+8=1

For full marks the running time of the function should be O(size of the island)

Explain your answer before writing code.

```
public int sumIsland(int[][] map, int n, int x, int y) {
```

IDEA: Consider the graph whose vertices are nxn points. (each inner vertex has 4 neighbours)
Run BFS from (x,y). For each visited vertex, add the value to the sum. Every visited vertex turns into a negative number to mark as visited. When done, return all to positive numbers.

```
    if (map[x][y] == 0)
        return 0;

    Queue<int[]> q = new LinkedList<>();
    int sum = addToQueue(map, x, y, q);

    while (!q.isEmpty()) {
        int[] p = q.remove(); // explore next vertex
        int i = p[0];
        int j = p[1];

        // add the 4 neighbours to the queue
        if (i > 0 && map[i - 1][j] > 0)
            sum += addToQueue(map, i - 1, j, q);

        if (i < n - 1 && map[i + 1][j] > 0)
            sum += addToQueue(map, i + 1, j, q);

        if (j > 0 && map[i][y1 - 1] > 0)
            sum += addToQueue(map, i, j - 1, q);

        if (j < n - 1 && map[i][j + 1] > 0)
            sum += addToQueue(map, i, j + 1, q);
    }

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (map[i][j] < 0)
                map[i][j] *= -1; // back to positive numbers

    return sum;
}


private int addToQueue(int[][] map, int i, int j, Queue<int[]> q) {
    int ret = map[i][j];
    q.add(new int[]{i, j});
    map[i][j] *= -1; // mark as visited
    return ret;
}
```

Problem 3 (Binary Trees) [25 points]

In this problem use the following definition of Binary Tree. You may assume the classes have the standard getters/setters.

```
public class BTNode<T> {
      private T data;
      private BTNode<T> leftChild;
      private BTNode<T> rightChild;
      private BTNode<T> parent;
   }

public class BinaryTree<T> {
      private BTNode<T> root;
   }
```

A. (10 points) Write the method equals() for the class Binary Tree. The method returns true if **this** has the exact same tree structure as the `other`, with the same data in the corresponding nodes. Compare if two nodes contain the same data using ==. **Explain the idea of the algorithm and its running time**.

```
public boolean equals(BinaryTree<T> other) {
  if (this == other)
    return true;
  if (!(other instanceof BinaryTree<?>))
    return false;
  BinaryTree<T> o = (BinaryTree<T>) other;
  return equalsNodes(this.root, o.root); // recursive helper function
}

private boolean equalsNodes(BTNode<T> a, BTNode<T> b) {
  if (a == b) return true;
  if (a == null || b == null) return false;
  if (a.getData() != b.getData()) return false;
  return equalsNodes(a.getLeftChild(), b.getLeftChild())
              && equalsNodes(a.getRightChild(), b.getRightChild());
   }
```

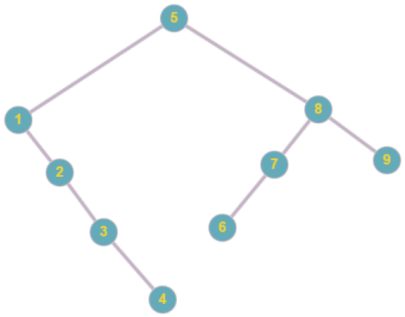
B. (10 points) Write the method nextInorder(). The method gets a node in a binary tree and returns the next node in the inorder traversal of the tree. The running time must be at most O(height of the tree)
Explain the idea of the algorithm and its running time.

```
public BTNode<T> nextInorder(BTNode<T> node) {

    if (node.getRightChild() != null) {
        // node has right child --
        // right and then left all the way
        BTNode<T> cur = node.getRightChild();
        while (cur.getLeftChild() != null)
            cur = cur.getLeftChild();
        return cur;
    } else { // node doesn't have right child --
             // go up as long as cur is right child of its parent
        BTNode<T> cur = node;
        while (cur.getParent() != null &&
                    cur.getParent().getRightChild() == cur)
            cur = cur.getParent();
        return cur.getParent();
    }



}
```

C. (5 points) Draw a ***Binary Search Tree*** whose preOrder traversal is [5,1,2,3,4,8,7,6,9].

Answer: root is 5. This means
- [1,2,3,4] is preorder of left subtree -- easy to see the it's just a string
- [8,7,6,9] is preorder of right subtree
    - 8 is the root, [7,6] on the left,[9] on the right

The tree is



Problem 4 (Heaps) [25 points]

A. (10 points) A min-heap is typically represented using an array. Write the following functions for min-heap. Explain your answer (e.g., by drawing a correspondence between the tree and the array)

1.
```
// returns the index in the array containing the root
public int getRoot() {

  return 0;

}
```

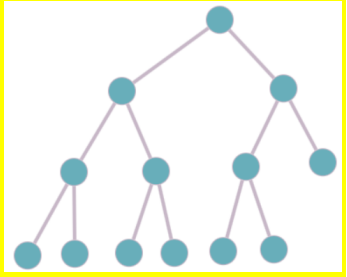
2.
```
// returns the index in the array containing the left child of
the node i
public int getLeftChild(int i) {

  return 2*i + 1;

}
```

3.
```
// returns the index in the array containing the right child of
the node i
public int getRightChild(int i) {

  return 2*i + 2;

}
```

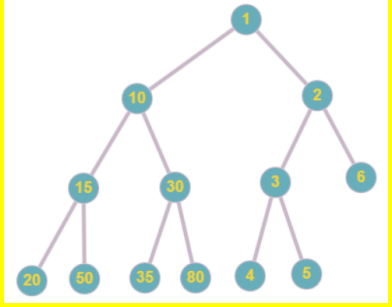
4.
```
// returns the index in the array containing the parent of the
node i
public int getParent(int i) {

  return (i-1) / 2;

}
```

B. (8 points) Consider the array of integers A= [1,10,15,20,50,30,35,80,2,3,4,5,6].
Is there a *min-heap* whose preOrder traversal is A?
If yes, draw the min-heap. If not, explain why.

Answer: Yes, such a min-heap exists.
The total number of nodes in the heap is 13 = 1+2+4+6. This means we have 3 full layers and 6 more nodes in the last layer. Start with the tree structure with no values.



Run the preorder traversal on the nodes of the empty tree, and add the values from the array in that order. We have root = 1, left child of root is 10, then 15, then 20… And so on



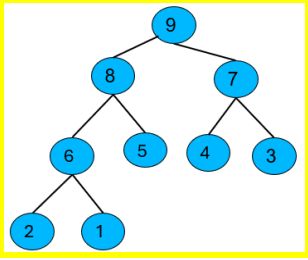
Finally, we check that it satisfies the min-heap property.

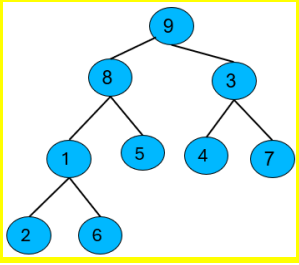
C. (7 points) Apply **buildMinHeap** on the array [9,8,7,6,5,4,3,2,1].
Show some intermediate steps by drawing the corresponding tree and the array.
Recall buildHeap works as follows:

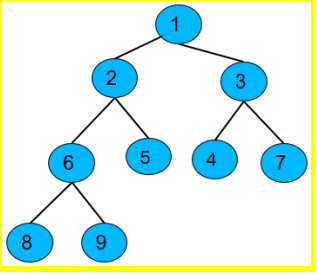
```
public void buildHeap(int[] ar) {
   for (int i=ar.length-1; i>=0; i--)
      heapify(ar,i);
}
```

Answer:

We start with this | After heapify 6 and 7 | End result after heapify 8 and 9







Extra page

Master Theorem

Let $T(n) = a\,T(n/b) + f(n)$, $T(1) = O(1)$

Define $c = \log_b(a)$

- If $f(n) = O(n^d)$ for $d<c$, then $T(n) = \Theta(n^c)$
- If $f(n) = \Omega(n^d)$ for $d>c$, then $T(n) = \Theta(n^d)$
- If $f(n) = \Theta(n^c)$, then $T(n) = \Theta(n^c \log(n))$