

CMPT225, Spring 2021

Midterm Exam

Name Nathan Esau

SFU ID: | 3 | 0 | 1 | 1 | 9 | 7 | 5 | 6 | 8 |

Problem 1	
Problem 2	
Problem 3	
Problem 4	
TOTAL	

Instructions:

1. You should write your solutions directly in this word file, and submit it to Coursys. Submitting a pdf is also ok.
2. Submit your solutions to Coursys before March 12, 23:59.
No late submissions, no exceptions
3. Write your name and SFU ID on the top of this page.
4. This is an open book exam.
You may use textbooks, calculators, wiki, stack overflow, geeksforgeeks, etc.
If you do, specify the references in your solutions.
5. Discussions with other students are not allowed.
Posting questions online asking for solutions is not allowed.
6. The exam consists of four (4) problems. Each problem is worth 25 points.
7. Write your answers in the provided space.
8. You may use all classes in standard Java, and everything we have learned.
9. Explain all your answers.
10. **Really, explain all your answers.**

Good luck!

Problem 1 [25 points]

A. (3 pts each) For each sentence decide whether it is True or False.

Write a brief explanation.

1) Let $T(n) = 10n^4 + 5n + 3$. Then $T = \Omega(n^3)$.

True. This statement is saying that $T(n)$ takes at least n^3 time. As n gets large n^4 grows faster than n^3 . So $T(n)$ will take at least n^3 time.

2) Let $T(n) = 10^n$. Then $T = \Theta(2^n)$.

False. There is no constant such that $10^n \leq k * 2^n$ for big n .

3) For all positive integers n , the function $\text{foo}(n)$ will return 0.

```
public int foo(int n) {  
    if (n > 0)  
        foo(n-1);  
    return 0;  
}
```

True. For $n = 1$, $\text{foo}(1) = \text{foo}(0) = 0$. For $n = 2$, $\text{foo}(2) = \text{foo}(1) = 0$. And so on. We always get to the 0 case. In fact, the function $\text{foo}(n)$ will return 0 for negative integers as well.

4) For all integers n (positive or negative), the function $\text{bar}(n)$ will return 0.

```
public int bar(int n) {  
    if (n > 0)  
        return bar(2*n);  
    return 0;  
}
```

True. At first glance function runs forever. But actually there will be integer overflow. For instance $\text{bar}(1) = \text{bar}(1073741824) = \text{bar}(-2147483648) = 0$. So large integers eventually overflow and we will get to the 0 case.

B. (5 pts) Use big-O notation to express the running time of $\text{foo}()$ on an array of length n .

Explain your answer.

```
public void foo(int array[]) {  
    fooRec(array, 0, array.length-1);  
}
```

```
public static void fooRec(int array[], int start, int end) {  
    if (start == end)  
        array[start]++;  
    else {  
        int mid = (end + start)/2;
```

```

        fooRec(array, start, mid);
        fooRec(array, mid+1, end);
        fooRec(array, mid+1, end);
    }
}

```

Here $T(n) = 3T(n/2) + O(1)$. Here $c = \log_2(3) = 1.58$ and $d = 1$. Since $d < c$ we have Master Theorem case 1. $T(n) = \Theta(n^{1.58})$.

- C. (4 pts) Use big-O notation to express the running time of `bar()` on an array of length n . **Explain your answer.**

```

public void bar(int array[]) {
    barRec(array, 0, array.length-1);
}

public static void barRec(int array[], int start, int end) {
    if (start <= end) {
        for (int i = start; i <= end; i++)
            a[i] += 1;
        barRec(a, start+1, end-1);
    }
}

```

The function is $O(N^2)$.

Following is an equivalent iterative implementation of `barRec`:

```

public static void barRec(int[] array) {
    for (int start = 0; start <= array.length / 2; start++) {
        int end = (array.length - 1) - start;
        for (int i = start; i <= end; i++) {
            array[i] += 1;
        }
    }
}

```

This is a nested loop. Outer loop goes until $N/2$. Inner loop goes from outer loop index to end. This is $O(N^2)$.

- (4 pts) Rewrite `bar()` so that it has the same functionality, but the running time is $O(n)$. **Explain your answer.**

```

public static void barRec(int[] array) {
    int start = 0;

```

```

        int end = array.length - 1;
        int mid = (start + end) / 2;
        for (int i = 0; i < array.length; i++) {
            int incr = (i <= mid) ? i + 1 : (end - start - i + 1);
            array[i] += incr;
        }
    }
}

```

Example: `int[] array = {5,4,7,1,3}` becomes `{6,6,10,3,4}`. Middle entries of array have larger values added to them. First entry += 1. Second entry += 2. This pattern continues until middle entry. Last entry += 1. Second last entry += 2. This pattern continues until middle entry.

D.

Problem 2 [25 points]

- A. (15 pts) Write a class `StackReverse` that supports the following operations, with running time $O(1)$ for each operation.

```

public class StackReverse<T> {

    public StackReverse() - a constructor, creates an empty stack

    public void push(T item) - adds an element to the stack

    public T pop() - removes an element from the stack

    public void reverse() - reverses the order of the elements. That is,
        the element that was the last in the stack becomes the first,
        and vice versa

    public int size() - returns the number of elements in the stack

    public boolean isEmpty() - checks if the stack is empty

}

```

The running time of each operation must be $O(1)$.

Before writing code, explain your answer.

Implement stack using linked list.

- Keep reference to top and bottom of stack and have a variable for the direction.
- When reversing, toggle the direction.
- Each node needs to have next and previous reference.
- Different logic for push and pop depending on direction. If forward, change top. Else change bottom.
- Keep a counter in the class for keeping track of size.

- For empty, we can check if size > 0.

```
public class StackReverse<T> {
    class Node {
        T item;
        Node prev = null;
        Node next = null;
        Node(T item) { this.item = item; }
    }

    int length;
    Node top;
    Node bottom;
    Boolean forward;

    public StackReverse() {
        this.length = 0;
        this.top = null;
        this.bottom = null;
        this.forward = true;
    }

    public void push(T item) {
        var node = new Node(item);

        if (top == null) {
            top = node;
            bottom = node;
        }
        else if (forward) {
            top.prev = node;
            node.next = top;
            top = node;
        }
        else { // backward
            bottom.next = node;
            node.prev = bottom;
            bottom = node;
        }
        length += 1;
    }

    public T pop() { // throw Exception if empty
        T item = null;
        if (forward) {
            item = top.item;
            top = top.next;
            if (top == null) bottom = null;
            else top.prev = null;
        }
        else {
            item = bottom.item;
```

```

        bottom = bottom.prev;
        if (bottom == null) top = null;
        else bottom.next = null;
    }

    length -= 1;
    return item;
}

public void reverse() {
    forward = !forward;
}

public Boolean isEmpty() {
    return length > 0;
}

public int size() {
    return length;
}
}

```

- B. (10 pts) Write an algorithm that gets an expression as a String in prefix notation and returns a String with the expression in postfix notation.

For example, on input `"/ * 2 + 3 4 - 18 16"` the method returns `"2 3 4 + * 18 16 - /"`.

You may assume the input is always valid

Explain your answer.

```

public static String prefix2Postfix(String prefix) {
    var ops = java.util.Arrays.asList("+", "-", "/", "*");

    // prefix2Infix
    var s1 = new java.util.Stack<String>();
    var t1 = prefix.split(" ");
    for (int i = t1.length - 1; i >= 0; i--) {
        if (ops.contains(t1[i])) {
            s1.push(String.format("( %s %s %s )", s1.pop(), t1[i], s1.pop()));
        }
        else {
            s1.push(t1[i]);
        }
    }
    String infix = s1.pop();

    // infix2Postfix
    var s2 = new java.util.Stack<String>();
    var t2 = infix.split(" ");
    String postfix = "";
    for (int i = 0; i < t2.length; i++) {
        if (ops.contains(t2[i]) || t2[i].equals("(")) {
            s2.push(t2[i]);
        }
    }
}

```

```

        else if (t2[i].equals("(")) {
            while (!s2.peek().equals("(")) {
                postfix += String.format("%s ", s2.pop());
            }
            s2.pop();
        }
        else {
            postfix += String.format("%s ", t2[i]);
        }
    }
    return postfix.stripTrailing();
}

```

Explanation: Convert prefix to infix then infix to postfix. More details below.

For provided example:

prefix = / * 2 + 3 4 - 18 16

infix = ((2 * (3 + 4)) / (18 - 16))

postfix = 2 3 4 + * 18 16 - /

prefix to infix.

process right to left. push numbers onto stack. when encounter operand, add "(" + stack.pop() + operand + stack.pop() + ")" to stack. At end return top of stack.

/ * 2 + 3 4 - 18 16

stack = ["16"], s = ""

stack = ["18", "16"], s = ""

stack = [], s = "(18 - 16)"

stack = ["(18 - 16)"], s = ""

stack = ["4", "(18 - 16)"], s = ""

stack = ["3", "4", "(18 - 16)"], s = ""

stack = ["(3 + 4)", "(18 - 16)"], s = ""

stack = ["2", "(3 + 4)", "(18 - 16)"], s = ""

stack = ["(2 * (3 + 4))", "(18 - 16)"], s = ""

stack = ["((2 * (3 + 4)) / (18 - 16))"], s = ""

s = "((2 * (3 + 4)) / (18 - 16))"

infix to postfix.

process left to right. add numbers to postfix string. Push left parenthesis and operands to stack. When encounter right parentheses, pop and add to postfix string until encountering a left parenthesis and remove the left parenthesis from the stack. At end return postfix string.

```

(( 2 * ( 3 + 4 ) ) / ( 18 - 16 ) )
stack = ["(", s = ""
stack = ["(", "(", s = ""
stack = ["(", "(", s = "2"
stack = ["*", "(", "(", s = "2"
stack = ["*", "(", "(", s = "2"
stack = ["*", "(", "(", s = "2 3"
stack = ["+", "(", "(", s = "2 3"
stack = ["+", "(", "(", s = "2 3 4"
stack = ["*", "(", "(", s = "2 3 4 +"
stack = ["(", s = "2 3 4 + *"
stack = ["/", "(", s = "2 3 4 + *"
stack = ["(", "/", "(", s = "2 3 4 + *"
stack = ["(", "/", "(", s = "2 3 4 + * 18"
stack = ["-", "(", "/", "(", s = "2 3 4 + * 18"
stack = ["-", "(", "/", "(", s = "2 3 4 + * 18 16"
stack = ["/", "(", s = "2 3 4 + * 18 16 -"
stack = [], s = "2 3 4 + * 18 16 - /"

```

C.

Problem 3 [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 pts) Write the method `equals(Object other)` for the class `BinaryTree`. The method returns true if the argument is a `BinaryTree` with the same data. You should compare the data in different nodes using `equals()` method in the class `T`.

The running time should be $O(n)$ in the worst case, where n is the size of the smaller tree. For example, if one tree has n vertices, and the other has n^2 vertices, the running time should be $O(n)$. **Explain your algorithm and running time.**


```

public boolean equalsHelper(BTNode<T> node1, BTNode<T> node2) {
    if (node1 == null && node2 == null) {
        return true;
    }

    if (node1 != null && node2 != null) {
        return node1.getData().equals(node2.getData()) &&
            equalsHelper(node1.getLeftChild(), node2.getLeftChild()) &&
            equalsHelper(node1.getRightChild(), node2.getRightChild());
    }

    return false;
}

@Override
public boolean equals(Object other) {
    if (!other.getClass().equals(getClass())) {
        return false;
    }

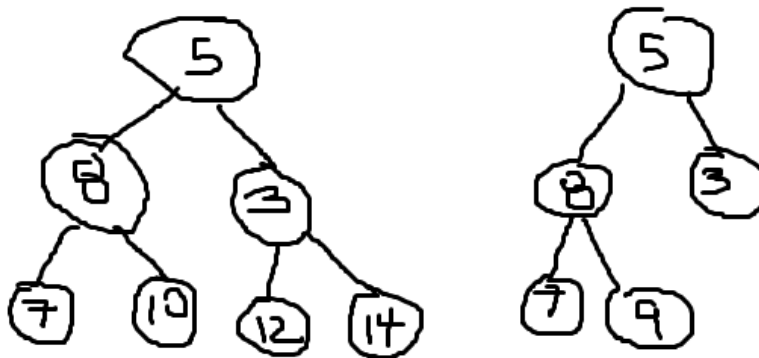
    var otherTree = BinaryTree.class.cast(other);
    var otherRoot = BTNode.class.cast(otherTree.getRoot());
    return equalsHelper(root, otherRoot);
}

```

Explanation

Traverse both trees together (pre-order) at same time. If tree1 node is null before the tree2 node, that means trees are different size and not equal. If tree1 node is not null and tree2 node is not null, make sure the nodes are same. If tree1 node is null and tree2 node is null that means they have same number of nodes and all previous nodes were equal.

Suppose we have following trees:



First we check $5 == 5$, then we check $8 == 8$, then we check $7 == 7$, then we check $10 == 9$. Then we are done (not equal). Suppose tree1 has $n1$ nodes and tree2 has $n2$ nodes. We will check at most $\min(n1, n2)$ nodes.

- B. (10 pts) Write a method for the class Binary Tree that returns the depth of the shallowest leaf in the tree. That is, among all leaves, you need to return the smallest depth of a leaf. What is the running time of your algorithm

Explain your algorithm and running time.

```
public int depthOfShallowestLeaf() { // assume root is not null
    var q = new ArrayList<Map.Entry<BTNode<T>, Integer>>();
    q.add(new AbstractMap.SimpleEntry<BTNode<T>, Integer>(root, 1));
    while (!q.isEmpty()) {
        Map.Entry<BTNode<T>, Integer> entry = q.remove(0);
        BTNode<T> node = entry.getKey();
        Integer depth = entry.getValue();

        if (node.getLeftChild() == null &&
            node.getRightChild() == null) {
            return depth;
        }

        if (node.getLeftChild() != null) {
            q.add(new AbstractMap.SimpleEntry<BTNode<T>, Integer>(
                node.getLeftChild(), depth + 1));
        }

        if (node.getRightChild() != null) {
            q.add(new AbstractMap.SimpleEntry<BTNode<T>, Integer>(
                node.getRightChild(), depth + 1));
        }
    }

    return 0;
}
```

Use a level order traversal. Keep track of depth of each node. Use Map.Entry for pair of node and depth. Once we encounter a node with no children we are done. Return the depth of that node.

In the worst case, we visit all the nodes of the tree so the running time is $O(N)$.

- C. (5 pts) Write a definition of a Ternary Tree in Java. Each node has data of a generic type, a pointer to the parent, and at most three children: left, middle, and right.

```
public class TernaryTree<T> {

    public class TNode<T> {
        public T data;
        public TNode<T> leftChild;
        public TNode<T> middleChild;
        public TNode<T> rightChild;
        public TNode<T> parent;
    }
}
```

```

        public TTreeNode(T data) {
            this.data = data;
            this.leftChild = null;
            this.middleChild = null;
            this.rightChild = null;
            this.parent = null;
        }
    }

    private TTreeNode<T> root;

    public TernaryTree(TTreeNode<T> root) {
        this.root = root;
    }
}

```

Only difference from binary tree is the addition of a middle child. Class would provide getters and setters for leftChild, middleChild, rightChild and parent as well as additional functions such as equal as needed.

Problem 4 [25 points]

A. (6 pts) Let $A = [4, 5, 10, 6, 7, 12, 14, 8, 15, 2]$.

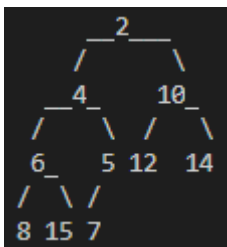
- Apply build-minHeap algorithm on A using the linear time algorithm we saw in class.
- Draw the tree representation of the heap.
- Draw the array representing the heap.
- **Draw the intermediate steps.**

Recall the buildHeap algorithm:

Treat the array as a complete binary tree
 For each vertex v starting from the bottom
 Apply $\text{heapify}(v)$

Final Heap

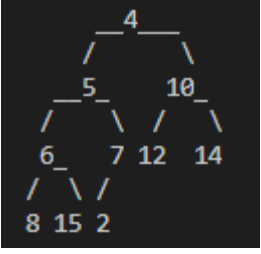
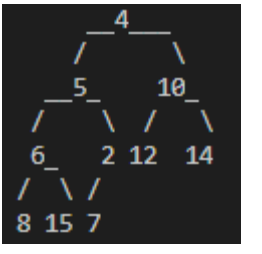
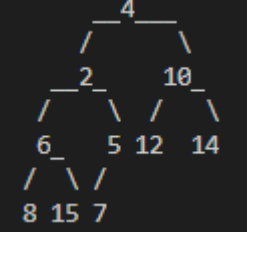
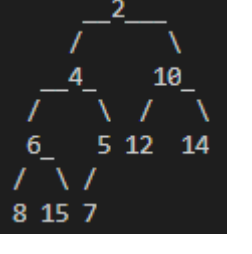
Tree representation



Array representation

[2, 4, 10, 6, 5, 12, 14, 8, 15, 7]. Note: this is the level-order traversal of heap tree.

Intermediate steps

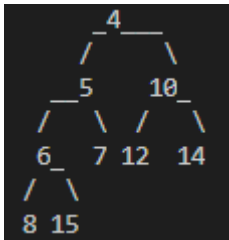
Initial	Heapify(7)	Heapify (5)	Heapify (4)
			

- B. (3 pts) Apply removeMin() on the heap obtained in part A, and draw the resulting heap.

Remove 2 and put 7 to top of heap. Then propagate 7 down.

Final Heap

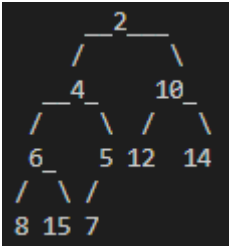
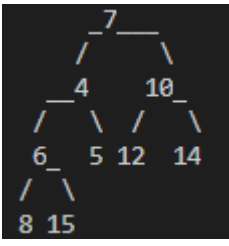
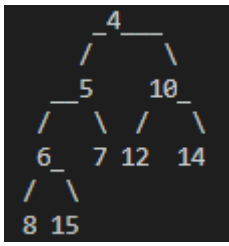
Tree representation



Array representation

[4, 5, 10, 6, 7, 12, 14, 8, 15]. Note: this is the level-order traversal of heap tree.

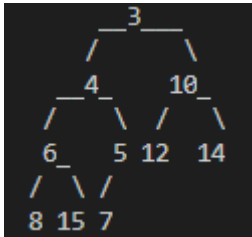
Intermediate Steps

Initial	remove 2 from top	propagate 7 down
		

- c. (3 pts) Add 3 to the heap obtained in part B, and draw the resulting heap.

Add 3 to bottom right of heap. Then propagate up.

Tree representation



Array representation

[3, 4, 10, 6, 5, 12, 14, 8, 15, 7]. Note: this is the level-order traversal of the heap tree.

Intermediate Steps

Initial	add 3 to bottom	propagate 3 up
<pre> graph TD 4 --- 5 4 --- 10 5 --- 6 5 --- 7 10 --- 12 10 --- 14 6 --- 8 6 --- 15 </pre>	<pre> graph TD 4 --- 5 4 --- 10 5 --- 6 5 --- 7 5 --- 3 10 --- 12 10 --- 14 6 --- 8 6 --- 15 </pre>	<pre> graph TD 3 --- 4 3 --- 10 4 --- 5 4 --- 10 5 --- 6 5 --- 7 10 --- 12 10 --- 14 6 --- 8 6 --- 15 </pre>

- D. (13 pts) Write a function that gets an array A of length n of integers, and $0 \leq k \leq n$, and returns an array B of length k containing the smallest k elements in A. In the end A must be in the same state as in the beginning.

The **running time** must be $O(n \log(k))$ and **extra space used** should be $O(k)$.

For example, on input $A = [4, 1, 5, 7, 2, 3, 1, 3]$ and $k = 4$ the output should be $B = [1, 1, 2, 3]$. The order of the elements in B is not important

Explain your idea before writing the code.

Since calling heapify on A is not an option, we need to be creative here.

- Build a max heap using first k elements. This takes $O(k)$ time and $O(k)$ space.

- For the remaining $n - k$ numbers, compare each to the root of the max heap. If the current number is smaller than the root, remove the root of the heap and add the current number to the heap. At the end, the root of the max heap will be k th largest number and all the nodes below the root will be smaller than the root. Therefore, we will have the k smallest numbers. This takes $(n - k) * \log(k)$ time and no additional space (still $O(k)$ from initial build of max heap).
- Total time is $O(k + (n - k) * \log(k)) = O(n * \log(k))$.

For the provided, example, the max heap will be (at each step):

[4,1,5,7] => [7,4,5,1]

[2,4,5,1] => [5,4,2,1]

[3,4,2,1] => [4,3,2,1]

[3,3,2,1] => [3,3,2,1]

[1,3,2,1] => [3,1,2,1]

We will return [3,1,2,1] as the k smallest numbers.

Re. max heap. Max heap is equivalent to min heap where we make each number negative. See function `heapify`.

```
public int[] kSmallest(int[] array, int k) {

    class MaxHeap {
        private int[] array;

        public MaxHeap(int[] array, int k) {
            this.array = new int[k];

            // copy first k numbers
            for(int i = 0; i < k; i++) {
                this.array[i] = array[i];
            }

            // initial heapify
            for (int i = k / 2 - 1; i >= 0; i-- 1)
                heapify(i);
        }

        public int[] getArray() {
            return array;
        }

        public int getMax() {
```

```

        return array[0];
    }

    public void replaceMax(int value) {
        array[0] = value;
        heapify(0);
    }

    private void heapify(int i) {
        if (i < array.length) {
            int left = 2 * i + 1;
            int right = 2 * i + 2;
            int j = i;

            if (left < array.length && -array[left] < -array[j]) {
                j = left;
            }

            if (right < array.length && -array[right] < -array[j]) {
                j = right;
            }

            if (j != i) {
                int tmp = array[i];
                array[i] = array[j];
                array[j] = tmp;
                heapify(j);
            }
        }
    }

    }

    MaxHeap heap = new MaxHeap(array, k);
    for (int i = k; i < array.length; i++) {
        if (array[i] < heap.getMax())
            heap.replaceMax(array[i]);
    }

    return heap.getArray();
}

```