CMPT125, Fall 2020

Final Exam
December 12, 2020

Name________________________

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

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 before December 12, 10:00pm.
   Submitting a pdf is also ok.

2. 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. Explain all your answers.

9. Really, explain all your answers.

Good luck!

Problem 1 [25 points]

a) Consider the following function.

```
int foo(unsigned int x) {
  if (x==0)
    return 0;
  if (x==1)
    return x+1;
  unsigned int half = x/2; // rounded down
  return foo(half) + foo(x-half);
}
```

[4 points] What will be the return value of foo(8)? Show some intermediate steps of the recursion.

[5 points] Use big-O notation to express the running time of foo()? Explain your answer.

[4 points] Explain the functionality of the function foo.

[4 points] Write an equivalent function (i.e., a function with the same functionality) more efficiently without recursion.

For items b and c below use the following struct

```
typedef struct {
  int x,y;
} point;
```

b) Consider the following code

```
void bar(point* p1, point* p2) {
  point p={p1->x, p2->y};
  p.x = 5;
  p.y = 6;
  *p2 = p;
  p1 = &p;
}

int main() {
  point p0 = {0,0};
  point p1 = {1,2};
  bar(&p0, &p1);
  return 0;
}
```

[4 points] Will the code above compile properly? If yes, what will be the values of p0 and p1 in the end of main()? Explain your answer.

c) Consider the following code.

```
point* create_point(int x,int y) {
  point p={x,y};
  point* ptr = &p;
  return ptr;
}

int main() {
  point* p1 = create_point(1,2);
  point* p2 = create_point(6,7);
  printf("%d, %d, ", p1->x, p1->y);
  printf("%d, %d \n", p2->x, p2->y);
  return 0;
}
```

[4 points] Will the code compile properly? If yes, what will be the output of the following code? Explain your answer.

Problem 2 [25 points]

A node in a linked list of ints is a struct containing an int and pointer to the next element in the list.

```
struct LL_node {
  int data;
  struct LL_node* next;
};
typedef struct LL_node LL_node_t;
```

In all questions below a linked list is represented by a pointer to the first element (head) of the list.

a) [5 points] Write a function in C that gets a linked list of ints and a boolean predicate on ints, and returns the number of elements in the list on which pred outputs true.

```
int count(LL_node_t* head, bool (*pred)(int)) {







}
```

[2 points] What is the running time of your function?

b) [8 points] Write a function in C that gets a linked list of ints and checks if all even numbers in the list come before all odd numbers. The running time of the function must be O(length of list).

For example, the function returns true on the following inputs:

2 → 4 → 10 → 0 → -11
4 → 3 → 5
-2 → -4
1
-6

The function needs to return false on the following inputs:

-3 → 4 → 6 → -8

7 → 10
1 → 0 → 1 → 0 → 1

```
bool evens_before_odds(LL_node_t* head) {











}
```

c) [10 points] Implement in C the ***rearrange*** function on a Linked List.
The function gets a pointer to the head of the list, and a pivot.
It rearranges the list so that all elements < pivot appear before all elements >=pivot.
The function returns a pointer to the new head of the list.

For example, if the input list is 5→2→1→3→3→4 and pivot = 4, then the output can be any of the following:
(1) 1 → 2 → 3 → 3 → 4 → 5 or
(2) 2 → 3 → 1 → 3 → 5 → 4 or
(3) 3 → 3 → 2 → 1 → 4 → 5

*You don't have to necessarily implement the rearrange algorithm for quickSort we saw in class.

```
LL_node_t* rearrange(LL_node_t* head, int pivot) {









}
```

Problem 3 [25 points]

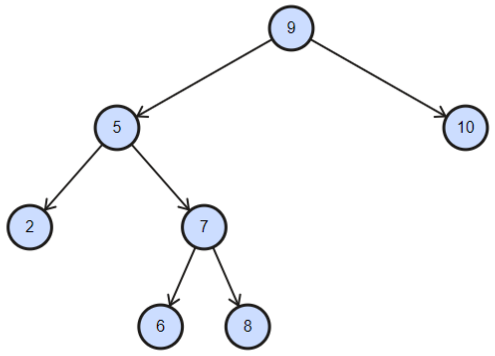
In this problem use the following struct for Binary Tree of ints.

```c
struct BTnode {
  int value;
  struct BTnode* left;
  struct BTnode* right;
};
typedef struct BTnode BTnode_t;
```

a) [10 points] Write a function in C that gets a pointer to the root of a Binary Tree, and counts the number of nodes in even layers and the number of nodes in odd layers. In the tree below there are 3 nodes in even layers (9,2,7) and 4 nodes in odd layers (5,10, 6, 8).

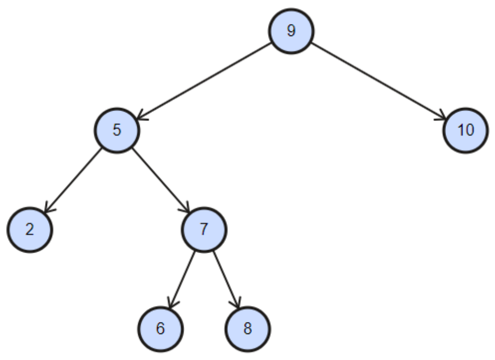
The output is stored in struct

```c
typedef struct {
  int evens;
  int odds;
} pair;
```



```c
pair count_even_odd_layers(BTnode_t* root) {







}
```

b) [7 points] Write a function in C that gets the root of a Binary Tree and computes the sum of all numbers in all leaves of the tree. For example, for the tree below the function should output: `2+6+8+10=26`



```
int sum_in_leaves(BTnode_t* root} {







}
```

b) [8 points total]

[4 pts] Draw the Binary Search Tree obtained from the following sequence of insertions: 7,2,5,8,3,4,9,6,1.

[2 pts] Remove 7 from the resulting tree, and draw the new tree.

[2 pts] Remove 2 from the resulting tree, and draw the new tree.

Problem 4 [25 points]

BoundedMemoryStack is a variant of stack where the capacity of the stack is bounded by some parameter specified when creating the stack.
If the stack is full, and a new element is added, then the element that was added the earliest (the farthest in the past) is “forgotten”, and the new item is added.

For example, suppose we have a stack with capacity 3. Consider the following sequence of operations

```
push(0)  // stack is [0]
push(1)  // stack is [1,0]
push(2)  // stack is [2,1,0]
push(3)  // stack is [3,2,1] // 0 is forgotten
pop()    // -- stack is [2,1] -- returns 3
push(4)  // stack is [4,2,1]
pop()    // stack is [2,1] -- returns 4
pop()    // stack is [1] -- returns 2
pop()    // stack is [] -- returns 1
```

Implement this ADT in C++. Specifically, you need to implement all the methods below.
The methods push(), pop(), isEmpty() must run in O(1) time.

```cpp
class BoundedMemoryStack{
  public:

    // constructor
    BoundedMemoryStack(int capacity) {}

    // copy constructor
    BoundedMemoryStack(const BoundedMemoryStack &bmStack) {}

    void push(int item) {}

    int pop() {}

    bool isEmpty() {}

    ~BoundedMemoryStack() {}

private:
 // implement me
};
```