CMPT125, Fall 2018

Midterm Exam - Solutions
October 30, 2018

Name_____

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

Problem 1	
Problem 2	
Problem 3	
Problem 4	
Problem 5	
TOTAL	

Instructions:

1. Write your name and SFU ID **clearly**
2. This is a closed book exam, no calculators, cell phones, or any other material.
3. The exam contains five (5) problems.
4. Each problem is worth 20 points.
5. Write your answers in the provided space.
6. There is an extra page in the end of the exam. You may use it if needed.
7. Explain all your answers.

Good luck!

Problem 1 [20 points]

a) [4 points] What will be the output of the following program?

```
#include <stdio.h>

int foo(int* x, int* y, int z) {
    y = x;
    z = 7;
    *y = z;
}

int main() {
    int a = 0, b = 1, c = 2;
    foo(&a, &b, c);
    printf("a = %d, b = %d, c = %d", a, b, c);
    return 0;
}
```

ANSWER: a = 7, b = 1, c = 2

b) Consider the following function.

```
void bar(int n) {
  int i = 1, sum = 0;
  while(sum < n*(n+1)/2) {
    printf("%d ", i);
    sum += i;
    i++;
  }
}
```

[4 points] What will it print on input n = 3? Show your intermediate computation if needed.

ANSWER: 1 2 3

[4 points] Use the big-O notation to express the running time of bar(n) as a function of n.

ANSWER: O(n).
Explanation: sum increases by 1, then by 2, then by 3.
After n iterations, sum = 1+2+3+...+n= n*(n+1)/2, and the while loop terminates.
Therefore, the total number of iterations is n, and the running time is O(n)

c) [4 points] Will the code below compile?
If yes, what will be the output? If no, explain why.

```c
#include <stdio.h>

int main() {
  int b[5] ={1,2,3,4,5};
  int* a = b;
  printf("a[2] = %d\n", a[2]);
  return 0;
}
```

ANSWER: Yes. It will print "a[2] = 3"

d) [4 points] Will the code below compile?
If yes, what will be the output? If no, explain why.

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
  int* a = (int *) malloc(5 * sizeof(int));
  for (int i = 0; i < 5; i++)
     a[i] = i;
  int b[5];
  b = a;
  printf("b[2] = %d\n", b[2]);
  return 0;
}
```

ANSWER: No, we cannot change b to point anywhere else.
The variable b is a constant pointer.

Problem 2 [20 points]
In this problem represent a Linked List of ints using `LLnode_t`:

```
struct node {
  int data;
  struct node* next;
};
typedef struct node LLnode_t;
```

a) Consider the following function

```
void fun_list(LLnode_t* head)
{
  if(head == NULL) {
     printf("\n");
     return;
  }

  printf("%d  ", head->data);
  if(head->next != NULL)
    fun_list(head->next);
  printf("%d  ", head->data);
}
```

[6 points] What will be the output of fun_list() on input 1→ 2→ 3→ 4→ 5 ?

ANSWER:

1 2 3 4 5 5 4 3 2 1

[4 points] Use big-O notation to express the running time of fun_list()?

ANSWER: O(length of the list)
Explanation: we access each node in the list exactly twice.

b) [8 points] Write a function in C that gets a *sorted* linked list, and removes duplicates.
For example, for input 1→ 1→ 1→ 2→ 2→ 2→ 5→ 6→ 6→ 6→ 7, the list will become
1→ 2→ 5→ 6→ 7.
You need to free the memory used by the nodes removed from the list.

```
void remove_duplicates(LLnode_t* head) {
```

Explanation: we run through the list with a pointer called current.
In each step if next element has same value as current, we remove the next element from the list.
Otherwise we set current = current->next

```
    LLnode_t* current = head;
    LLnode_t* remove_me;
    while(current != NULL) {
        if (current->next &&(current->data == current->next->data))
        {
          remove_me = current->next;
          current->next = remove_me->next;
          free(remove_me);
        }
        else  // current->next is either NULL or different data
           current = current->next;
    }
}
```

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

ANSWER: O(length of the list)
Explanation: we access each node in the list exactly twice.

Problem 3 [20 points]

a) [12 points] Write a function that creates a copy of a stack, i.e., it gets a stack and creates another stack with the same elements in the same order.
**In the end on the function, the original stack must be returned to its initial state.
**If you allocate memory for any temporary variables, you need to release them.

You may assume that the functions below are implemented, but you cannot make assumptions about how they are implemented.

```
typedef struct {
        ... // not known
} stack_t;
stack_t* stack_create(); //create empty stack
void push(stack_t* s, int item); //adds item to the stack
int pop(stack_t* s); //removes the top of the stack and returns it
bool is_empty(stack_t* s); //checks if the stack is empty
void stack_free(stack_t* s); //free the memory used by the stack

// returns a copy of orig
stack_t* stack_copy(stack_t* orig) {

    stack_t* ret = stack_create();
    stack_t* tmp = stack_create();

    while (!is_empty(orig))
        push(tmp, pop(orig));

    int item;
    while (!is_empty(tmp))
    {
        item = pop(tmp);
        push(orig, item);
        push(ret, item);
    }

    stack_free(tmp);
    return ret;



}
```

[3 points] What is the running time of you function stack_copy()?

ANSWER: There are two loops each runs in time O(size of orig)
Therefore, the total running time is O(size of orig)

b) Consider the following sequence of operations on a stack:

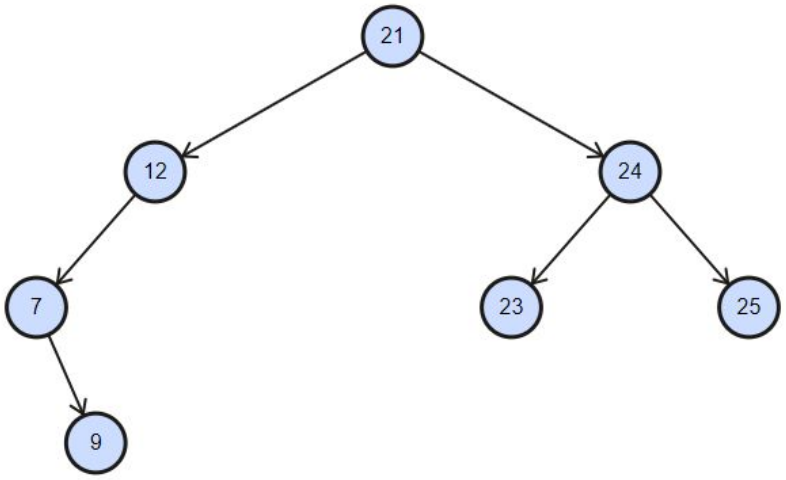
```
stack_t* s = stack_create();

push(s, 1);
push(s, 2);
push(s, 3); // stack = [1,2,3]

printf("%d ", pop(s));  // stack = [1,2]
push(s, 4);
push(s, 5);
push(s, 6);  // stack = [1,2,4,5,6]

printf("%d ", pop(s));
printf("%d ", pop(s)); // stack = [1,2,4]

push(s, 7); // stack = [1,2,4,7]
```

[5 points] What will be the state of the stack in the end?
Show the intermediate steps of the computation.

__
_7_
_4_
_2_
_1_

Problem 4 [20 points]

Consider the following Binary Search Tree



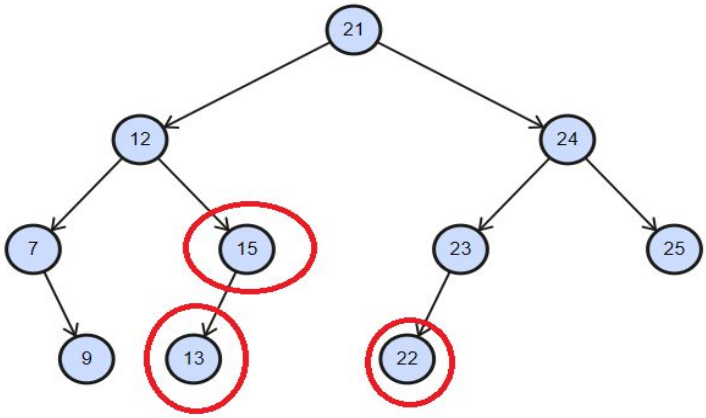
a) [1 point] What is the depth of this tree?

ANSWER: 3

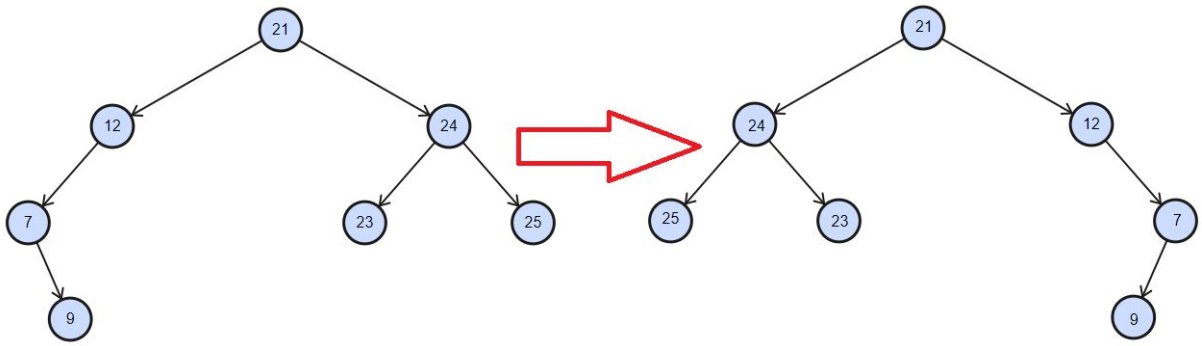
b) [3 points] Write the InOrder traversal for this tree.

ANSWER: 7 9 12 21 23 24 25

c) [6 points] Add the elements 15, 13, 22 to the Binary Search Tree.



d) [10 points] Write an algorithm that gets a Binary Tree and convert it into its mirror reverse. For example




```
struct BTnode {
 int value;
 struct BTnode* left;
 struct BTnode* right;
 struct BTnode* parent;

};
typedef struct BTnode BTnode_t;

void mirror_tree(BTnode_t* root) {

    if (root != NULL)
    {
      BTnode_t* tmp;
      tmp = root->left;
      root->left = root->right;
      root->right = tmp;
      mirror_tree(root->left);
      mirror_tree(root->right);
    }




}
```

Problem 5 [20 points]

a) [8 points] How many **swaps** will Insertion Sort perform on the input [9, 6, 2, 1, 4]?

ANSWER: 8 swaps

Insert 9: 0 swaps. Result [9, 6, 2, 1, 4]

Insert 6: 1 swap: (6,9). Result [6, 9, 2, 1, 4]

Insert 2: 2 swaps: (2,9), (2,6). Result [2, 6, 9, 1, 4]

Insert 1: 3 swaps: (1,9) (1,6), (1,2). Result [1, 2, 6, 9, 4]

Insert 4: 2 swaps: (4,9), (4,6). Result [1, 2, 4, 6, 9]

b) [8 points] List all recursive calls made by *Merge Sort* on input [9, 6, 7, 2, 1, 4]?

ANSWER:
[9 6 7], [2 1 4]
[9], [6 7]
[6], [7]
[2], [1 4]
[1], [4]

Answers may vary, e.g. [9 6 7] can be split as [9 6] [7]

c) [6 points] What is the running time of *Selection Sort* on a sorted array of length n? Use big-O notation to express the running time.

ANSWER: $O(n^2)$
Explanation: the running time of Selection sort is always $O(n^2)$.
In each iteration we are looking for the minimal element regardless of the array.
Finding the minimal element in the first iteration takes n-1 comparisons
Finding the minimal element in the second iteration takes n-2 comparisons
And so on...
The total number of comparisons is $1+2+3+...+ (n-1) = O(n^2)$regardless of the array