CMPT 125, Fall 2019 - (incomplete) solutions

Midterm Exam
October 28, 2019

Name_____

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

Problem 1	
Problem 2	
Problem 3	
Problem 4	
TOTAL	

Instructions:

1. Duration of the exam is 90 minutes.
2. Write your 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 problem is 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. Explain all your answers.
8. Really, explain all your answers.

Good luck!

Problem 1 [25 points]

a) [6 points] What will be the output of the following program?

```
#include <stdio.h>
enum week {MON, TUE, WED, THUR, FRI, SAT, SUN};

          x gets ptr to a, y gets 1,  z gets ptr to c
void foo(int* x,     int y, int* z) {
    x = z;                    // x points to c
    y = *x;                   // the local variable y changes to 2
    *z = THUR;                // c = 3
}

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

//ANSWER: a = 0, b = 1, c = 3

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

```
#include <stdio.h>

int main() {
  char str[5] ={'a','b','c','d',0};
  char* ptr = str;
  printf("%s\n", ptr);
  return 0;
}
```

//ANSWER: the function prints: abcd

c) Consider the following function.

```
int bar(int n) {
  if (n <= 0)
    return 0;
  else {
    int i = 0, sum = 0;
    while(sum <= n*(n-1)/2) {
      i++;
      sum += i;
    }
    return i + bar(n-1);
  }
}
```

**Below you may need the following fact: 1+2+3+...+n = n*(n+1)/2.

[4 points] What does bar(n) return on input n = 3? Show some intermediate computation if needed.

When the while loops exists we have i = n
Therefore, bar(3) returns 3+bar(2) = 3+2+bar(1) = 3+2+1+bar(0) = 6

ANSWER: 6

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

Denote the runtime by T(n). Then T(n) = O(n) + T(n-1)
That is, T(n) = T(n-1) + Cn for some constant C
We have T(n) = T(n-1) + Cn = T(n-2) + C(n-1) + Cn = = T(n-3) + C(n-2) + C(n-1) + Cn = …
= C(1+2+3+...(n-1)+n) = $O(n^2)$

[6 points] Explain in words what bar(n) returns, and write a function with the same functionality as bar(n) whose running time is O(1).

The function returns 1+2+3+...+n. We can compute it using the following code:

```
int bar(int n) {
  return n*(n+1)/2;
}
```

Problem 2 [25 points]

a) [15 points] Recall the MergeSort algorithm.

```
void merge_sort(int* A, int n) {
  if (n >= 2) {
      int mid = n/2;
      merge_sort(A, mid);
      merge_sort(A+mid, n-mid);
      merge(A,n,mid);
  }
}
```

Implement the merge function that gets an array A of length n, and an index mid, and it is guaranteed that the part A[0,...mid-1] is sorted in ascending order, and A[mid...n-1] is sorted in ascending order. The function merges the two halves in time in A in time O(n).
You may assume all elements are distinct.
Remember to use malloc/free if you need to use a new array.

```
void merge(int* A, int n, int mid) {

    // temporary array, to be released in the end
    int* tmp = (int*) malloc(n*sizeof(int));

    int ind1 = 0, ind2 = mid;
    int i = 0;

    //  adding elements to the new array one at a time
    while (i < n) {
      if (ind1==mid || (ind2<n && A[ind2]<A[ind1])) {
        tmp[i] = A[ind2];
        ind2++;
      }
      else { // (ind2 == n || A[ind1] < A[ind2])
        tmp[i] = A[ind1];
        ind1++;
      }
      i++;
    }
  //  copy from tmp back to A
  for (int i = 0 ; i < n; i++)
    A[i] = tmp[i];

  // free the unused memory
   free(tmp);



}
```

b) [6 points] Consider the ***QuickSort*** algorithm that uses as a pivot the first element (i.e., A[0]). List all recursive calls made by the algorithm on input A = [2, 8, 6, 1, 5, 3].
Show some intermediate steps of the computation.

TIn first rearrangement the algorithm swaps(8,1) and gets A = [2,1,6,8,5,3].
The we swap 2 to the correct position: = [1,2,6,8,5,3]
We make a two recursive calls: [1] and [6, 8, 5, 3]

- [1] doesn't do anything
- [6, 8, 5, 3] choose 6 as the pivot. Then we swap (8,3), and get [6, 3, 5, 8].
  Then we swap the pivot to the middle: [5, 3, 6, 8]
  - We make two more recursive calls: [5, 3] and [8]
  - [5,3] chooses 5 as pivot, then swaps(5,3) and gets [3,5]. Then it makes a recursive call on [3], and returns.
  - [8] make no more recursive calls.

c) [4 points] Consider the **Binary Search** algorithm. How many comparisons will it make on input A = [2, 4, 6, 8, 10, 12, 14] when searching for 6.

Compare 6 to 8, then go to [2, 4, 6]
Compare 6 to 4, then go to [6]
Compare 6 to 6, and return the index

Problem 3 [25 points]

a) [6 points] Consider the following sequence of operations on a stack. What will be the state of the stack in the end? Show some intermediate steps of the computation.

```
stack_t* s = stack_create();
push(s, 1);          [1
push(s, 2);          [1,2
push(s, 3);          [1,2,3
push(s, 4);          [1,2,3,4
pop(s);              [1,2,3

push(s, 5);          [1,2,3,5
push(s, 1);          [1,2,3,5,1
pop(s);              [1,2,3,5
push(s, 1);          [1,2,3,5,1
push(s, 3);          [1,2,3,5,1,3
pop(s);              [1,2,3,5,1
```

ANSWER: [1,2,3,5,1

b) [6 points] Consider the following sequence of operations on a queue. What will be the state of the queue in the end? Show some intermediate steps of the computation.

```
queue_t* q = queue_create();
enqueue(q, 1);           --> [1]
enqueue(q, 2);           --> [2,1]
enqueue(q, 3);           --> [3,2,1]
enqueue(q, 4);           --> [4,3,2,1]
dequeue(q);              --> [4,3,2]

enqueue(q, 5);           --> [5,4,3,2]
enqueue(q, 1);           --> [1,5,4,3,2]
dequeue(q);              --> [1,5,4,3]

enqueue(q, 1);           --> [1,1,5,4,3]
enqueue(q, 3);           --> [3,1,1,5,4,3]
dequeue(q);              --> [3,1,1,5,4]
```

ANSWER: TAIL -> [3,1,1,5,4] -> HEAD

c) Consider the following function.

```
void do_what(queue_t* q) {
   // Base case
   if (queue_is_empty(q));
       return;
   // Dequeue current item (from the head)
   int data = dequeue(q);
   // apply recursion
   do_what(q);
   // Enqueue current item (to the tail)
   enqueue(q, data);
}
```

[8 points] If the input to the function is a queue in the state q = [1,2,3] (1 is the head, 3 is the tail), what will be the state of q when the function returns? Explain your answer.

The function returns the queue in the reversed order.

The state of q will be [3,2,1].

[5 points] Use the big-O notation to express the running time of do_what. Explain your answer.

ANSWER: O(n), where n is the size of the queue.

Explanation: each element in the queue is removed once and added once.

Problem 4 [25 points]

In this problem we 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

```
int fun_list(LLnode_t* head)  {
  if (head == NULL) {
     printf("\n");
     return 1;
  }
  else  {
    int w = fun_list(head->next);
    printf("%d  ", head->data);
    return w * head->data;
  }
}
```

[4 points] What will fun_list() *return* on input 1→ 2→ 3→ 2→ 5? Explain your answer.

The function returns 1*2*3*2*5 = 60.

[4 points] What will fun_list() *print* on input 1→ 2→ 3→ 2→ 5? Explain your answer.

The function skips a line and the prints ”5 2 3 2 1 “.

b) [12 points] Write a function in C that gets a linked list and a number, and returns the first node containing the number. If the number is not in the list, the function returns `NULL`.
The linked list is represented as a pointer to the first node in the list `LLnode_t* head`.

The state of the list on return should not change.

```
LLnode_t* find(LLnode_t* head, int number) {
  if (head == NULL)
    return NULL;
  if (head->data == number)
    return head;
  return find(head->next, number);




}
```

[5 points] Use big-O notation to express the running time of your function?

ANSWER: O(size of list) because each item is touched at most once

Extra page