CMPT 125, Fall 2022

Final Exam - Solution
December 13, 2022

Name_________________________

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

Problem 1	
Problem 2	
Problem 3	
Problem 4	
TOTAL	

Instructions:

1. Duration of the exam is 180 minutes.
2. Write your full name and SFU ID NUMBER **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) Consider the following function.

```
char* foo(int n) {
   char* str = malloc(n+1);
   str[0] = 0;
   for (int i=0; i<n; i++)
     strcat(str, "*");
   return str;
}
```

[6 points] Explain the functionality of foo().

Answer: The function returns a string consisting of n asterisks

[6 points] What is the running time of foo() as a function of n? Use big-O notation to express your answer. Explain your answer.

Answer: strcat(dest, src) works by reading the dest until the end of the string, and then concatenates src to the end of it. The running time in the i'th iteration is O(i). Therefore, the total running time is $O(1+2+3+...n) = O(n^2)$

b) [7 points] Let T(n) be given using the recursive formula T(n) = T(n-1) + O(n), T(1) = 1. Use big-O notation to express the rate of growth of T as a function of n.

Answer: T(n) = O(n^2)

Proof: T(n) = T(n-1) + Cn
= T(n-1) + C(n-1) + Cn
= T(n-2) + C(n-2) + C(n-1) + Cn
= …
= T(1) + C*1 + C*2 + C*3 + ... + C(n-2) + C(n-1) + Cn = O(n^2)

c) [6 points] Let T(n) be given using the recursive formula T(n) = T(n-1) + O(n), T(1) = 1. Write a recursive function whose running time is expressed as T(n).

Answer:

```
  void foo(int n) {
      if (n<=0)
        return;
      for (int i=0; i<n; i++)
        sprintf( "%d ", i);
      foo(n-1);
  }
```

Problem 2 [25 points]

a) [6 points] Write Insertion Sort in C.

```
void insertion_sort(int* A, int n) {
```

Answer: there are many variations possible

```
     int i, j, tmp;
     for (i=1; i<n; i++) {
       j = i;
       while (j>0 && A[j]>A[j-1]) {
         tmp = A[j];
         A[j] = A[j-1];
         A[j-1] = tmp;
         j--;
       }
     }

}
```

b) [8 points: 4 points each]
What is the worst case running time of Insertion Sort? Explain your answer.

Answer: there are two nested loops. For a fixed i the while loop will have at most i iterations. Therefore, the total running time is O(1+2+3…+n) = O(n^2)

What is the running time of Insertion Sort on a sorted array? Explain your answer.

Answer: if the array is sorted, then the while loop will have 1 iteration every time. Therefore, the total running time is O(n)

c) [5 points] Consider the *QuickSort* algorithm that given an array A, chooses A[0] as the pivot. How many **swaps** will it perform on the array A = [10, 0, 2, 4, 6, 8]? Explain your answer. Write some intermediate steps of the algorithm when necessary.

Answer:
First 10 is the pivot, after rearranging the array remains the same. **10 is swapped with 8**. The new array becomes [8, 0, 2, 4, 6, 10]

Recursive call on the subarray [8, 0, 2, 4, 6]. 8 is the pivot, after rearranging the array remains the same. **8 is swapped with 6**. The new array becomes [6, 0, 2, 4, 8]

Recursive call on the subarray [6, 0, 2, 4]. 6 is the pivot, after rearranging the array remains the same. **6 is swapped with 4**. The new array becomes [4, 0, 2, 6]

Recursive call on the subarray [4, 0, 2]. 4 is the pivot, after rearranging the array remains the same. **4 is swapped with 2**. The new array becomes [2, 0, 4]

Recursive call on the subarray [2, 0]. 2 is the pivot, after rearranging the array remains the same. **2 is swapped with 0**. The new array becomes [0, 2]

Recursive call on the subarray [0] - base case

Total number of swaps: 5

d) [6 points] Explain what the function qsort() does. Write the signature of the function, and explain each of the parameters. Give an example of how qsort() is used.

Answer: qsort() gets an array of length n, size of each item, and the compare function
void qsort(void *ar, int n, int size_of_item, int (*compare)(const void *, const void*))
// actually, the types of n and size_of_item are both size_t, but that's not very important.

Example of usage:

```
  int compare(const void* a, const void * b) {
     int n1 = (int*)a;
     int n2 = (int*)b;
     return n1-n2;
  }

int main()  {
    int ar[5] = {5,4,3,2,1};
    qsort(ar, 5, sizeof(int), compare);
    // print ar
    //
}
```

Problem 3 [25 points]

In the problem a Linked List of ints is represented as follows.

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

typedef struct {
  LL_node_t* head;
  LL_node_t* tail;
} LL_t;
```

a) [10 points] Write a function that gets a Linked List of ints and reverses it.
The running time of the function must be O(length of list).
For example, if the linked list is *1 → -3 → 10 → 0 → 1 → 11*, then after applying the function it becomes *11 → 1 → 0 → 10 → -3 → 1*.

```
void reverse(LL_t* list) {
```

// no magic, just make it work

```
     LL_node_t* prev_node = NULL;
     LL_node_t* cur_node = list->head;
     LL_node_t* next_node = NULL;

     list->tail = cur_node;
     while(cur_node) {
       next_node = cur_node->next;
       cur_node->next = prev_node;
       prev_node = cur_node;
       cur_node = next_node;
     }
     list->head = prev_node;









}
```

b) [15 points: 5 points each] Implement the following standard functions on a Linked List with pointers to head and tail, using the struct above.

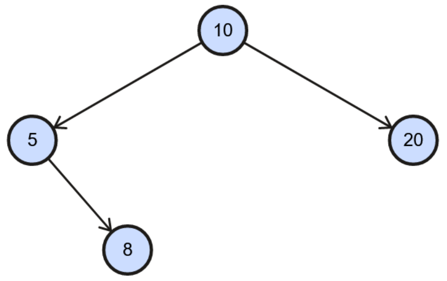
```c
// adds a node with the given value to the head of the list
// the running time is O(1)
void add_to_head(LL_t* list, int val) {

     LL_node_t* new_node = malloc(sizeof(LL_node_t));
     new_node->data = val;
     new_node->next = list->head;
     list->head = new_node;
     if (list->tail == NULL)
       list->tail = new_node;

}
// applies f to the value in each node of the list
// the running time is O(length of the list)
void map(LL_t* list, int (*f)(int)) {


     LL_node_t* cur = list->head;
     while(cur) {
        cur->data = f(cur->data);
        cur = cur->next;
     }

}
// appends all nodes of src to the end of dest, and deletes src.
// after the application of the function, src cannot be used
// the running time is O(1)
void LL_cat(LL_t* dest, LL_t* src) {

     if (dest->tail != NULL) { // dest is not empty
          dest->tail->next = src->head;
          dest->tail = src->tail;
     }
     else { // dest is empty
          dest->head = src->head;
          dest->tail = src->tail;
     }
     free(src);



}
```

Problem 4 [25 points]

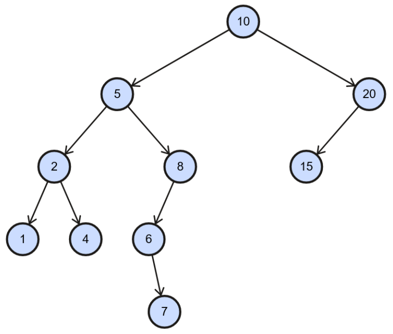
In this problem use the following struct for Binary Tree of ints.

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

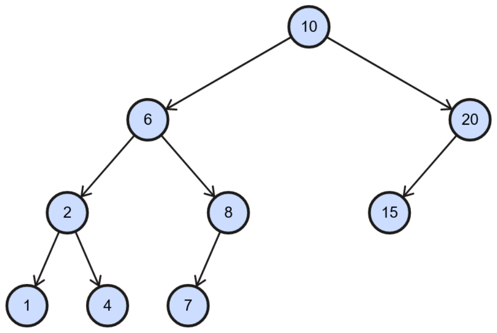
a) Consider the following Binary Search Tree.



[5 points] Add the list of numbers to the Binary Search Tree in the given order: 2, 1, 6, 4, 7, 15. What will be the result in the end? Draw the resulting tree with the ten nodes.

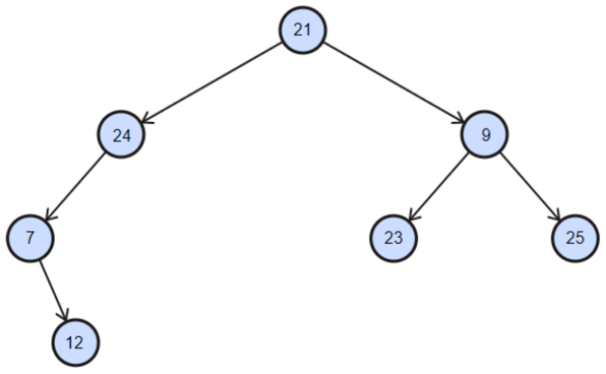


[5 points] Remove 5 from the BST obtained above (tree with values 1,2,4,5,6,7,8,10,15,20). Draw the resulting BST.



b) [15 points] Write a function that gets a node in a binary tree (not necessarily the root), and returns the next node in the *post-order traversal order*.
The running time must be O(depth of three).
For example, in the tree below the post order traversal is [12,7,24,23,25,9,21]

- on input 12 the output should be 7
- on input 7 the output should be 24
- on input 23 the output should be 25
- on input 9 the output should be 21
- on input 21 the output should be NULL



Explain your answer before writing the code.

```
BTnode_t* next_postorder(BTnode_t* node) {
      // given a node if we want its next postorder
      // we need to go first up the tree
      BTnode_t* par = node->parent;

      if (par == NULL)  // node is the root
            return NULL;

      else if (par->right == node)  // the easy case
            return par;

      else {  // the interesting case of parent->left == node
            if (par->right == NULL)
                  return par;
            else {  // go right, and then all the way to the left
                  BTnode_t* cur = par->right;  // it is not null
                  while (cur->left || cur->right)  go down the subtree
                        cur = cur->left ? cur->left : cur->right;
                  return cur;
            }
      }
}
```

Extra page

Empty page