COP 3502

UNIVERSITY OF CENTRAL FLORIDA (UCF)

Department of Computer Science

COP 3502 Computer Science I

Common Final Examination

Total points: 100   Total Time: 2 hours and 30 minutes

SOLUTIONS

Instructions:

1. This is a **closed-book** and **closed-neighbor** exam. No notes, textbooks, or outside assistance are permitted.
2. **Calculators, smartwatches, and electronic devices** (phones, tablets, laptops, earbuds, etc.) are strictly prohibited.
3. If you would like to use scratch paper, please come up to the front of the room and request some after the exam has started.
4. Write **clearly and legibly**. If the instructor cannot read your handwriting, the answer may not receive credit.
5. Manage your time wisely

A. Multiple Choice/Fill in the Blanks/Short Answer Questions [1 x 5 = 5 pts]

1. Consider a heap is represented by the following array. The first item is stored at index 1. Answer the following questions (**please answer the data not the index!!!**)

index	1	2	3	4	5	6	7	8	9	10	11
data	6	10	12	15	17	18	23	20	19	34	

a) What value is stored in the right child of node storing 12? 23 **(Grading 1 pt if both correct)**

b) What value is stored in the parent node of the node storing 19? 15

2. In the array-based representation of a queue, we must store the elements in an array, the index to the front of the queue, the actual size of the array, and what one other value?

Number of elements actually in the queue **(Grading 1 pt)**

3. Which of the following two applications most likely utilize a stack instead of a queue:

(a) Undo operation (b) Ticketing center (c) Tracking the jobs in a network printer

(d) Job scheduling by operation system (e)Tracking function calls in a program

(a), (e) **(Grading: 1 pt if both correct, if both 1 and 3 are half credit give one point)**

4. In one way, a trie is more efficient (uses less memory) in storing words from a dictionary than an array of strings, but in another way it's less efficient (uses more memory). Explain how the trie comparatively saves memory AND how it uses extra memory.

Answer: More efficient: Letters aren't stored and shared prefixes amongst all words are stored only once.
Less efficient: Each node (essentially letter link) also stores 26 pointers, many of which are unused.
Grading: 1 pt if about half of this is here

5. Which data structure is used to maintain priority queue efficiently? binary heap **(Grading 1 pt)**

B. Summations [2 + 5 = 7 pts]

1. The code segment below stores the sum of several terms in a variable called sum. Write, but do not solve, the nested summation that corresponds to what the variable sum is equal to.

```
int sum = 0;

for (int i=1; i<100; i++)
    for (int j=1; j<=i; j++)
        sum += (2*i+j*j);

printf("sum = %d.\n");
```

Answer: $\sum_{i=1}^{99}(\sum_{j=1}^{i}(2i + j^2))$, **Grading: 1 pt if incorrect but on right path, 2 pts if correct, if outer sum is to 100, give 1 pt.**

2. Determine a closed-form solution for the summation shown below, in terms of n. Express your answer in standard polynomial form ***an***^{2} + ***bn*** + ***c***, where a, b and c are constants. *[You must show the steps of splitting and/or shifting when applicable to receive full credit]*

$$\sum_{i=1}^{n}\left(\sum_{j=1}^{3i+7} 2\right) = \sum_{i=1}^{n}\big(2(3i+7)\big) = \sum_{i=1}^{n}(6i+14) = \left(\sum_{i=1}^{n} 6i\right) + \left(\sum_{i=1}^{n} 14\right)$$

$$= \frac{6n(n+1)}{2} + 14n = 3n(n+1) + 14n = 3n^2 + 3n + 14n = 3n^2 + 17n$$

Grading: 1 pt to do inner sum, 1 pt to get to split sums, 1 pt to evaluate each sum, 1 pt for the algebra to get to the answer.

C. Recurrence Relations [1 + 6 = 7 pts]

1. Consider the following recursive function. Write out the corresponding recurrence relation T(n) such that T(n) equals the value that compute(n) returns. ***(You do not need to solve the recurrence relation)***

```
int compute(int n) {
    if (n == 1)
         return 2;
    else
        return 2 * compute(n - 1) + 3;
}
```

Answer: T(n) = 2T(n-1)+3, for n > 1, T(1) = 2 **(Grading: 1 pt have to get everything to get point.)**

2. Use the **iteration technique** to determine the Big-Oh run-time of the following recurrence relation. You may find the following formula useful while solving this: $a^{\log_{a^k} n} = \sqrt[k]{n}$

[Rubric (6 pts): 2 iterations 2 pts, general form 1 pt, getting value of k and apply k to the general form 1 pt, further simplification 1 pt, final bigOh 1 pt]

$T(n) = 2T\left(\frac{n}{4}\right) + 5$ $\quad\quad T(1) = 1$

Iterate the recurrence three times:

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{4}\right) + 5 \\ &= 2\left(2T\left(\frac{n}{16}\right) + 5\right) + 5 \\ &= 4T\left(\frac{n}{16}\right) + 3(5) \\ &= 4\left(2T\left(\frac{n}{64}\right) + 5\right) + 3(5) \\ &= 8T\left(\frac{n}{64}\right) + 7(5) \end{aligned}$$

After k iterations, we have:

$$T(n) = 2^kT\left(\frac{n}{4^k}\right) + (2^k - 1)(5)$$

Since we know T(1), solve for k in $\frac{n}{4^k} = 1$. This means $n = 4^k$ and that $(2^2)^k = n$, and using our exponent rule we get that $(2^k)^2 = n$. Taking square root of both sides we find that $2^k = \sqrt{n}$. While it's not necessary, the desired value of k to substitute is $log_4 n$. Substituting appropriately for $2^k$and 4^k for our chosen value of k we get:

$$T(n) = \sqrt{n}T(1) + \left(\sqrt{n} - 1\right)(5) = 6\sqrt{n} - 5 = \boldsymbol{O}(\sqrt{\boldsymbol{n}})$$

Grading: Written above, 1 pt for 2nd iteration, 1 pt for 3rd iteration, 1 pt general form, 1 pt plug in k, 2 pts to simplify to the Big-Oh answer.

D. Sorting Algorithms (3 + 2 + 4 + 3 = 12 pts)

1. Consider running a Merge Sort on the array shown below. What does the array look like right before the **last call** to the Merge function executes:

Original Array:

index	0	1	2	3	4	5	6	7	8	9
value	37	15	9	17	22	64	2	25	6	13

Array right before last call to Merge:

index	0	1	2	3	4	5	6	7	8	9
value	9	15	17	22	37	2	6	13	25	64

Grading: 3 pts correct, 2 pts only if incorrect but right 5 values on left and right, 1 pt if pairs are sorted instead, 0 otherwise

2. Quick Sort has an average running time of **O(nlogn),** but in the worst case it is **O(n²).** Briefly explain what type of **input** and **quick sort implementation** can cause Quick Sort to run in its worst case, and **why** Merge Sort does **not** suffer from the same worst-case behavior.

Type of input and quick sort implementation (1 to 3 lines):
Sorted input with a pivot selection strategy of simply selecting the leftmost item as the pivot.

Merge sort discussion (1 to 3 lines):
The recursive call always splits the array into two roughly equal sizes, so the run-time is dictated by the recurrence relation T(n) = 2T(n/2) + O(n), which ensures a reasonably efficient sort regardless of the original input.

Grading: 1 pt each, give the point if it's close...

3. The following function tried to implement insertion sort to sort only the numbers between the left index to the right index (both inclusive) passed to the function. However, it has **four** bugs. Correct the bugs directly on top/side of the line.

```
void insertionSort(int arr[], int left, int right) {
    int i, hand, j;
    for (i = left+1; i <= right; i++)  {
          hand = arr[i];
          for(j=i-1; j>=left; j--) {
             if(arr[j]>hand)
                 arr[j+1] = arr[j];
             else
                 break;
          }
          arr[j+1] = hand;
    }
 }
```

Grading: 1 pt for each fix (I think this one's hard...)

4. The array shown below has been partitioned exactly once (first function call in a quicksort of an array). Observe the following partitioned array and answer which element was the partition element. Why do you think that is the partition element (Mention three reasons that really validate that your chosen item is the partition element.

index	0	1	2	3	4	5	6	7	8
data	17	20	14	13	10	27	50	34	48

a) Partition Element Index: **5**   b) Value of Partition Element: **27**

c) Reason this is the only possible partition element: It's the only index in the array such that all values to the left of it are smaller than its contents (27) and all items to the right of it are greater than the contents of the index.

Grading: 1 pt each slot (pretty sure this will be 0 or 3 for all papers...)

COP 3502

E. Binary Trees (5 + 3 + 2 = 10 pts)

1. Write a **recursive** function in C that **identifies and prints** out the value all nodes in a **Binary Search Tree (BST)** that have **two children**, in numeric order.

```
typedef struct treenode {
    int data;
    struct treenode *left;
    struct treenode *right;
} treenode;

void twoChildren(treenode *root) {
    if (root == NULL) return;                              // Grading: 1 pt
    twoChildren(root->left);                               // Grading: 1 pt
    if (root->left != NULL && root->right != NULL)         // Grading: 1 pt
        printf("%d ", root->data);                         // Grading: 1 pt
    twoChildren(root->right);                              // Grading: 1 pt
}
```

2. Given the following code:

```
int mistery(struct treenode* root, int k) {
  if (root == NULL)
     return 0;
  int s = mistery(root->left, k) + mistery(root->right, k);
  if (root->data > k)
     s += root->data;
  return s;
}
```

```
        50
       /  \
     20    80
    / \    / \
  10   30  70 90
       / \
     25  40
```

What would be returned from this function if we pass the root of the tree shown on the right side and k=30?

Answer: **330** **(Grading: 3 pts correct, 2 pts for 415, 1 pt 50, 0 pts otherwise)**

3. Consider the following binary search tree. Redraw the tree after deleting 55. (Note: There are two correct answers to this question.)

```
        55
      /    \
    20      75
   /  \    /  \
 10   30  70   80
     /  \   \
   25   35  72
        /
      32
```

Answer #1

```
      35
    /    \
  20      75
 /  \    /  \
10   30 70  80
    /  \  \
  25   32 72
```

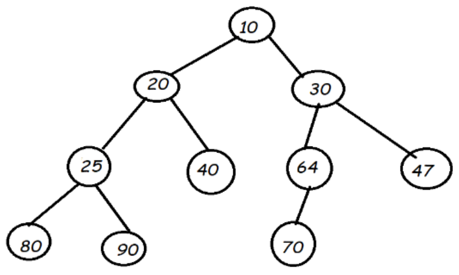
Answer #2

```
      70
    /    \
  20      75
 /  \    /  \
10   30  72  80
    /  \
  25   35
       /
     32
```

Grading: 1 pt root, 1 pt rest

F. Binary Heaps (1 + 5 + 2 + 2 = 10 pts)

1. Consider the following tree. Is this a valid binary minheap? Before answering carefully observe if it fulfills all the properties of a binary heap. Justify your answer. *Just saying yes/no has no credit without justification.*



No, this isn't a valid heap since 40 has no kids but 64 does, so it's not a complete binary tree. **(Grading: 1 pt)**

2. Consider a min heap is stored in an integer array ***int heaparray[100]***, which is globally declared. There are currently ***heapsize*** number of nodes in the heap, and ***heapsize*** is also globally declared. The first item of the heap is stored at index 1. Assume the following functions are already provided to you:
void percolateUp(int arr[], int upIndex); //it performs percolate up starting from index upIndex.

void percolateDown(int arr[], int downIndex); //it performs percolate down starting from index downIndex.

Write a *heapInsert*() function that takes an item, and heap array in the parameters, and insert the item into the heap. **If the heap array does not have enough space, it returns 0**. If the insertion is successful, it returns 1. Note that *heapsize* is globally declared, and you can simply use *heapsize* variable when needed.
// arrCap is the size of the array heapArr.

```
int heapInsert(int heapArr[], int arrCap, int item) {
    if (heapsize == arrCap-1) return 0;             // Grading: 1 pt
    heapArr[heapsize+1] = item;                 // Grading: 1 pt
    percolateUp(heapArr, heapsize+1);           // Grading: 1 pt
    heapsize++;                                 // Grading: 1 pt
    return 1;                                   // Grading: 1 pt
}
```

3. a) What is the run-time to build a binary heap with n items using heapify: O(n) **Grading: 1 pt**

b) What is the run-time to delete the minimum item from a binary heap with n values? Please give your answer in Big-Oh notation in terms of n: O(lg n) **Grading: 1 pt**

4. Consider the following function, where a binary heap is stored in the arr array. This function implements one of the following heap functionalities. Choose the most appropriate answer from the following list of options.

void whatIsThis(int arr[], int index){
if (index > 1) {
if (arr[index/2] < arr[index]) {
swap(&arr[index], &arr[index/2]);
whatIsThis(arr, index/2);
}
}
}
	Choose the best option:
	o percolate down for a minheap
	o percolate down for a maxheap
	o percolate up for a minheap
	o [✓ **percolate up for a maxheap**]
	o heapify
	o heapsort

Grading: 2 pts all or nothing

G. Tries (2 + 1 + 5 = 8 pts)

1. How many nodes would be in a trie (including the root node) after inserting all the following words: potler, pot, pond, to, top, torn, ton

15 **(Grading: 2 pts right answer, 1 pt for 14 or 16, 0 otherwise)**

2. How many nodes would be there in total if we never inserted potler into the trie but inserted the rest of the words listed in question G1?

12 **(Grading: 1 pt right answer, 0 otherwise)**

3. Consider the following trie node struct

```
typedef struct trie_node {
    int isWord;  //1 if it is a word, 0 otherwise
    struct trie_node* next[26];
} trie_node;
```

Write a ***iterative*** function hasPrefixPlusOne that returns 1 if there exists a word in the trie that has the form: prefix + one extra letter, and returns 0 otherwise. For example, if the trie contains cat, car, cart and prefix = "ca", then hasPrefixPlusOne(root, "ca") should return 1 because cat and car match. However, if you call the function with cat, it should return 0 as there is no word of length 4 with "cat" as a prefix. **You may assume that if a pointer is not NULL in a trie that there is at least one word down that subtrie.**

```
int hasPrefixPlusOne(trie_node *root, char prefix[]) {

    int len = strlen(prefix);

    for (int i=0; i<len; i++) {                // Grading: 3 pts total

        trie_node* tmp = root->next[prefix[i]-'a'];

        if (tmp == NULL) return 0;

        root = tmp;
    }

    for (int i=0; i<26; i++)                   // Grading: 1 pt
        if (root->next[i] != NULL && root->next[i]->isWord)
            return 1;

    return 0;                                  // Grading: 1 pt
}
```

H. Bitwise Operators and Base Conversion (2 + 5 = 7 pts)

1. Perform the following base conversions:

a) $9C_{16} = ?_8$ Answer: 234_8 **Grading: 1 pt for correct answer only**

$9C_{16} = 010\ 011\ 100_2 = 234_8$

b) $117_{10} = ?_2$ Answer: 1110101_2 **Grading: 1 pt for correct answer only**

```
2 | 117
2 |  58  R 1
2 |  29  R 0
2 |  14  R 1
2 |   7  R 0
2 |   3  R 1
2 |   1  R 1
```

2. An int sample contains a virus if the 8 bits from the right side of the sample have at least n (n≤8) number of bits set to 1. Write a function that receives an int sample and an int n and returns 1 if the sample contains the virus. Otherwise, the function returns 0. For example, if sample = 14 and n = 2, then the 16-bit binary representation of 14 is 0000000000001110. As the right 8 bits has 3 ones (which is ≥ 2), the function should return 1 indicating that the virus is present in the sample. On the other hand, if n = 4, for the same sample = 14, the function should return 0 as the right 8 bits of the sample has less than 4 ones.

```
int hasVirus(int sample, int n) {

    int cnt = 0;                       // Grading: 1 pt
    for (int i=0; i<8; i++)            // Grading: 1 pt
        if (sample & (1<<i))           // Grading: 1 pt
            cnt++;                     // Grading: 1 pt
    return cnt >= n;                   // Grading: 1 pt
}
```

I. Hash Tables (5 + 1 = 6 pts)

1. Using the quadratic probing technique and the hash function f(x) = (3x+2)%13, show where each of the following values would be placed in a hash table of size 13, if inserted in this order: 4, 1, 6, 3, 13, 17, and 7:

index	0	1	2	3	4	5	6	7	8	9	10	11	12
value		4	13			1	7	6			17	3	

Grading: 1 pt for 4, 1 and 6 total, 1 pt for each of the rest.

2. What is the common problem of linear probing (one line)? **Answer:**

Clustering (accept if they just say this and nothing else), in more detail, once there is a consecutive run of array spaces that are occupied, this replacement strategy ensures that any item that initially hashes to any location in that consecutive run is guaranteed to be placed at the end of the consecutive run, increasing its size, thereby increasing the probability a future item will land somewhere in the cluster and continue to grow it.
Grading: 1 pt accept if they have a rough idea or even use the single word clustering or something similar.

COP 3502

J. AVL Trees (3 + 5 = 8 pts)

1. All AVL Trees are Binary Search Trees but not all Binary Search Trees are AVL Trees. Draw a valid Binary Search Tree storing the values 1, 2, 3, 4 and 5 that is NOT a valid AVL Tree. Describe the additional requirement for a Binary Search Tree to also be an AVL Tree.

a) Drawing:

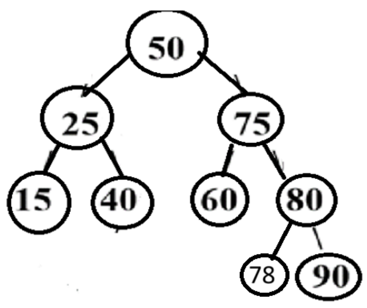
```
1,
 \
  2
   \
    3
     \
      4
       \
        5
```

Note: There are 42 possible valid binary search trees storing the numbers 1, 2, 3, 4 and 5. (This is the fifth Catalan number...) Of these, there are 6 that are valid AVL trees. so there are 42 – 6 = 36 correct answers to this question. 28 of these are trees with roots 1 or 5 and the other 8 have roots 2 or 4.

Grading: 1 pt if valid BST that isn't AVL, 0 otherwise.

b) Additional AVL Tree Requirement: For each node in the tree, the height of its left subtree and right subtree can not differ by more than 1. **Grading: 2 pts, give partial if necessary.**

2. Consider the following AVL tree. Insert 85 to this tree. If the tree requires re-balancing, then redraw the tree after rebalancing it.



When we first add 85, the tree looks like this:

```
          50
        /     \
     25          75
    /  \        /  \
  15    40   60    80
                  /  \
                78    90
                     /
                   85
```

Grading: 1 pt for showing 85 here.

Tracing up the ancestral path, 85 is balanced, 90 is balanced, 80 balanced, but 75 is not. Identify 75 as A, 80 as B and 90 as C and restructure as follows:

```
          50
        /     \
     25          80
    /  \        /    \
  15    40   75        90
            /  \      /
          60   78   85
```

Grading: 1 pt for 80, 1 pt for 75, 1 pt for 90, 1 pt for rest

K. Dynamic Memory Allocation (5 pts)

The following code has 5 memory management issues. Identify and write the issues clearly beside the lines as a comment. You can assume that the code will continue till the end of the function while answering this question. **(Grade: Total 5 pts. [For each identifying 0.5 pt, proper reasoning 0.5 pts]. A wrong answer will deduct a point from a correct answer.)**

```
int n = 10,
int *p1, *p2, *p3, **p4;
char str1[100] = "test string ";
char *str2;
strcpy(str2, str1);                    // Error #1: no room in str2
p1 = (int *)malloc(n * sizeof(int));
p2 = (int *)malloc(n * sizeof(int));
for(int i=0; i<n; i++)
      p1[i] = rand()%100;
p2 = p1;                               // Error #2: memory leak old mem
                                       // pointed to by p2 is inaccessible.

*p3 = 50;                              // Error #3: p3 not pointing to
p4 = (int **) malloc(n * sizeof(int*)); // allocated memory.
for(int i=0; i<n; i++)
      p4[i] = rand()%100;              // Error #4: Type mismatch, LHS is
                                       // type int*, RHS is type int.

free(p1);
free(p2);                              // Error #5: p1 and p2 were pointing
free(p4);                              // to the same thing, so there's no
                                       // memory to free.
```

Grading: 1 pt for identifying the lines, since only 1 pt per line, I would just give it to them as long as the reason is somewhat close.

L. Stack/Queues (5 pts)

A word is called a palindrome if it reads the same when reversed. For example, the word “planet” is *not* a palindrome because its reverse, “tenalp,” is different from the original. In contrast, the word “rotor” *is* a palindrome, since reversing it still gives “rotor.” Other examples of mirror words include “noon,” “civic,” “radar,” and “pip.”

Write a function that takes a string as its parameter and returns 1 if the string is a mirror word and 0 otherwise. You **must use stack operations** to perform this check.(Your grade depends on correctly using the stack, not on simply solving the problem.) Assume the following stack functions are ***already provided***. The stack has enough space. The top of the stack is controlled through standard push/pop operations.

```
#include <stdio.h>
#include <string.h>
void initialize(stack* s);         // initializes an empty stack
int push(stack* s, char value);    // pushes a character onto the stack
int isEmpty(stack* s);             // returns 1 if the stack is empty, 0 otherwise
char pop(stack* s);                // pops and returns the top character
char peek(stack* s);              // returns the top character without removing it
```

// (pop and peek return 'I' if the stack is empty)

//Complete the following function:

```
int isPalindrome(char *str) {
    stack s;  //you really don't need to know the properties of stack
    initialize(&s);
    int len = strlen(str);

    for (int i=0; i<len; i++)                         // Grading: 1 pt
        push(&s, str[i]);                             // Grading: 1 pt

    for (int i=0; i<len; i++)                         // Grading: 0 pts
        if (pop(&s) != str[i])                        // Grading: 1 pt
            return 0;                                 // Grading: 1 pt

    return 1;                                         // Grading: 1 pt
}
```

M. Linked Lists (5 pts)

Write a **recursive** function that takes in a pointer to the head of a linked list, **head,** and a positive integer, **div,** and prints out each value stored in the list divisible by **div.**

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

void printDivisible(node* head, int div) {
    if (head == NULL) return;                         // Grading: 1 pt
    if (head->data%div == 0)                          // Grading: 1 pt
        printf("%d ", head->data);                    // Grading: 1 pt
    printDivisible(head->next, div);                  // Grading: 2 pts
}
```

N. Algorithm Analysis (5 pts)

Write the run-time of the following operations/algorithms:

a) Worst case run-time of inserting a word with length p in a trie: **O(p)**

b) Worst case run-time of inserting an item in a binary heap of n elements: **O(lg n)**

c) Best case run-time of insertion sort of n elements: **O(n)**

d) Best case run-time of selection sort of n elements: **O(n^2)**

e) Best case run-time of merge sort of n elements: **O(nlgn)**

Grading: 1 pt for each, has to be correct.