COP 3502 Final Exam Review Questions

1) The following question deals with dynamic memory allocation involving the following struct:

```
typedef struct {
    char title[50];
    char author[50];
    int pages;
} BookT;
```

a) Allocate an array of n items of type BookT*. Call the array myBooks. Assume n is already declared and stores a positive integer.

b) Consider the situation where myBooks[i] is a pointer to an array of one genre of book. Assume that the integer array numBooks of size n is such that numBooks[i] stores how many books of genre i there are. For example, if n = 4 and numBooks stored 3, 12, 4 and 7, then there are 3 books of genre 0, 12 books of genre 1, 4 books of genre 2 and 7 books of genre 3. Write code to allocate space for each of n arrays to store each genre of books. Declare any variables you need but assume that numBooks exists and stores the desired values already.

c) Free all of the memory that is directly or indirectly pointed to by myBooks after step b.

2) Write a function that takes in a file pointer to a file that contains a positive integer, *n*, on the first line (signifying how many numbers in the file follow), followed by *n* doubles, 1 per line. Your function should read in *n*, dynamically allocate an array of *n* doubles, read those doubles into the array in the natural order, and then return a pointer to this array. The function prototype is given below:

```
int* readData(FILE *ifp) {




}
```

3) Perform the following base conversions:

a) 1347_{10} converted to base 7.

b) 4562_8 converted to base 10.

c) 10011101011101_2 converted to base 16.

d) 4567_9 converted to base 7.

e) Convert 137_8 to base 5

4) Use the iteration technique to determine a Big-Oh solution for the following recurrence relation:

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

5) Determine $\sum_{i=n}^{2n-1}(in + n + i + 3)$ in terms of n. Put a box around your final answer.

6) 100,000 thousand searches on a database with n = 2^{16} records takes 60 milliseconds. A single search on this database with n records executes in O(lg n) time. Consider a situation where the database size increased to n = 2^{20}. How long would 500,000 searches on this database take?

7) Write a function that takes in pointers to two sorted linked lists, combines them by rearranging links into one sorted linked list, effectively merging the two sorted lists into one, and returns a pointer to the new front of the list. Since no new nodes are being created and no old nodes are being deleted, your code should **NOT** have any mallocs or frees. Also, note that this destroys the old lists. If you try to print either listA or listB after calling merge, the lists are likely to print differently. (***Hint: This is probably much easier to do recursively.***) Fill in the function prototype provided below and use the struct provided below:

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

node* merge(node* listA, node* listB) {









}
```

8) Complete the function below so that it returns a pointer to the node in the linked list that points to the smallest item in the list. For example, if the list stores 3, 12, 18, 2, 9, and 6, with front point to the 3, a pointer to the node storing the two should be returned. Please use the struct and function prototype shown below. Return NULL if the input list is NULL.

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

struct node* getPtrToMin(struct node* front) {









}
```

9) Write a **recursive function** that deletes every other node in a linked list pointed to by the pointer front, which is passed in as a parameter. In particular, make sure you delete the first, third, fifth, etc. nodes and return a pointer to the new list. If the list has zero or one item in it, NULL should be returned.

```
struct ll* {
    int data;
    struct ll* next;
};

struct ll* delEveryOther(struct ll* front) {









}
```

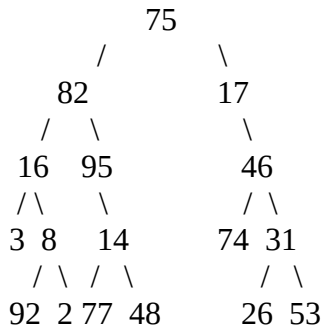
10) One way to calculate the cube root of a positive number is to run a binary search between a number known to be less than the desired cube root and a number known to be greater than the desired cube root. Write a ***recursive*** function that returns an approximation within 10^{-9} to the cube root of a double x, where the cube root is known to be in between a double low and a double high. A wrapper function has been written for you that calls your recursive function. Remember that the function $f(x) = x^3$ is a monotonically increasing function. (Hint: Your base case is when low and high are within 10^{-9} of each other!!!)

```
#define EPSILON 1e-9

double mycuberoot(double x) {
    double max = 1;
    if (x > max) max = x;
    return reccuberoot(x, 0, max);
}

double reccuberoot(double x, double low, double high) {









}
```

11) Consider a binary search in the array below for the value 28, which values are checked in the array before it is found? (Please list these in the order in which they are checked.)

index	0	1	2	3	4	5	6	7	8	9	10	11	12
value	2	3	7	10	11	15	18	20	22	26	27	28	30

12) Provide the preorder, inorder and postorder traversals of the binary tree shown below.

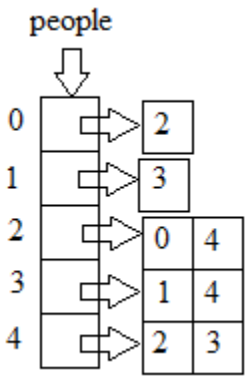


Preorder:
Inorder:
Postorder:

13) An efficient way to store data about the relationships amongst a group of n people is as follows: give each person a unique label from 0 to n-1. For each person, create a dynamically sized array, the size of the number of acquaintances that person has. For example, if person 0 has 3 acquaintances, then their array would be size three and each item in this array would be the number of one of their acquaintances. Since the people are labeled 0 to n-1, we can store each person's array, in an array itself! Consider an example with 5 people, where we have the following acquaintance pairs:

(0, 2), (1, 3), (2, 4), (3, 4)

We would store this in an array of size 5, where each item is an array as follows:



Write a function that takes in a file pointer to a file that stores data about a set of acquaintances, dynamically allocates the array of arrays described above and returns a pointer to that array of arrays. The file format is as follows:

The first line contains a single positive integer n, the number of people described in the file. The following n lines contain information about the acquaintances of each of person 0 through person n-1. Each of these lines starts with a positive integer n_i, the number of acquaintances of person i. The following n_i values are the numbers of the people that person i is acquainted with, in increasing order. It's guaranteed that the data is consistent. If person i is acquainted with person j, then person j will be acquainted with person i. The input file storing the data shown in the previous picture is as follows:

```
5
1 2
1 3
2 0 4
2 1 4
2 2 3
```

Please fill in the function prototype given on the next page. (Note: To actually implement this in a meaningful way, we'd have to store the value of n and the sizes of each of the arrays so that we could properly iterate through the structure. I haven't asked you to do these tasks in order to make the solution shorter.)

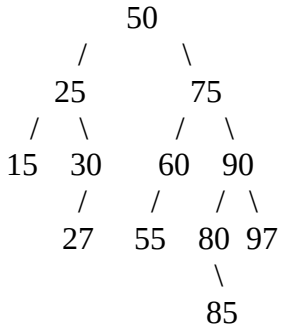
```
int** readgraph(FILE* ifp) {




}
```

14) During a Merge Sort of 8 items, the merge function gets called 7 times. Show the result of the array after each of the merges, as they occur in sequential order, on the array shown below. (The last row has been filled in for you since after the last merge, the array is sorted.)

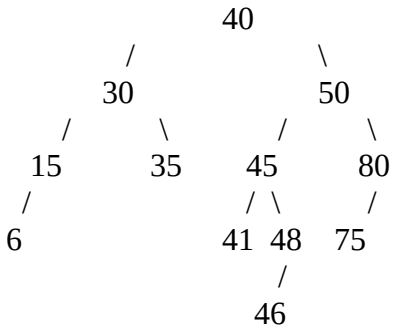
Solution

Initial	18	6	5	13	9	17	14	1
merge 1								
merge 2								
merge 3								
merge 4								
merge 5								
merge 6								
Last	1	5	6	9	13	14	17	18

15) Show the final result of deleting the node storing 15 from the AVL tree shown below. Draw a box around your answer.

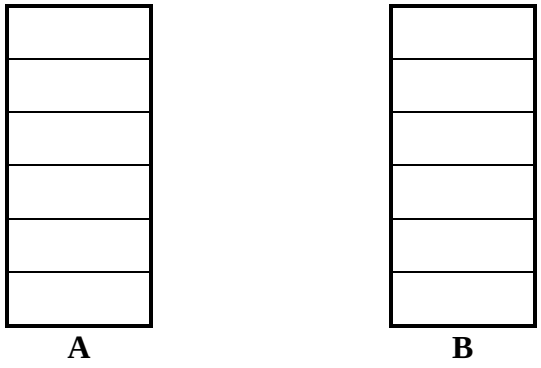


16) Show the result of deleting 35 from the AVL Tree below:



17) Evaluate the following postfix expression showing the contents of the stack in each of the positions (A and B) indicated.

					A						B			
6	8	2	+	*	7	3	–	/	10	+	3	–		



A **B**

Value of Postfix expression:

18) Consider inserting the following items into an empty minimum heap, in this sequence: 18, 12, 9, 22, 13, 17, 6, 47, 15, and 8. Show the final contents of the heap, as they are stored in a 1-based array. You may draw tree pictures of the varying stages of the heap for partial credit, but for full credit you must fill in the array provided below.

Index	1	2	3	4	5	6	7	8	9	10
Value										

19) Consider running heap sort by repeatedly extracting the minimum item from the heap shown below, in its array representation, until it contains no more elements:

index	1	2	3	4	5	6	7	8	9	10	11
value	3	10	4	20	11	6	79	83	27	53	12

Show the contents of the heap after each item has been deleted. For convenience, the item deleted is noted in the first column.

20) Draw the heap corresponding to the following array:

Index	1	2	3	4	5	6	7	8	9
Value	3	8	6	15	9	7	12	22	16

21) Show the result of inserting the value 4 into the heap from the previous question.

22) Using separate chaining hashing, and the hash function $f(x) = (x^2 + 3x + 7) \% 19$, draw a picture of what the structure would look like after adding the following values, in this order: 3, 4, 7, 8, 2, 9, 14, 1, 15, and 12. Assume that we insert new items to the ***front*** of the appropriate linked list.

23) Consider the problem of printing out each five digit number (that can start with 0), where each digit is distinct and the difference between consecutive digits must exceed 2. An example of such a number is 07259. We can use backtracking to print out all such numbers. Two functions are given below: a wrapper function that calls a recursive backtracking function and solves the original problem, and the recursive function that solves arbitrary sub-problems. The wrapper function is completed for you. Complete the recursive function so that the task is properly solved. A call to the wrapper function should print out all the desired numbers.

```
#define SIZE 5
#define NUMDIGITS 10

int print() {
    int digits[SIZE];
    int used[NUMDIGITS];
    int i;
    for (i=0; i<NUMDIGITS; i++)
        used[i] = 0;
    printRec(digits, 0, used);
}

void printRec(int digits[], int k, int used[]) {

    int i;
    if (k == SIZE) {
        for (i=0; i<k; i++)
            printf("%d", digits[i]);
        printf("\n");
        return;
    }

  // Fill in recursive code here.









}
```