Honors Computer Science I (COP 3502) Final Exam Solutions
Date: 4/25/2024

1) (10 pts) Perform the following base conversions. Show work below each answer slot.

(a) (2 pts) 11010111_2 = 215 $_{10}$

$11010111_2 = 2^7 + 2^6 + 2^4 + 2^2 + 2^1 + 2^0 = 128 + 64 + 16 + 4 + 2 + 1 = 215$

Grading: 1 pt breakdown, 1 pt answer

(b) (4 pts) 935_{10} = 12220 $_5$

5 | 935
5 | 187 R 0
5 | 37 R 2
5 | 7 R 2 Answer is 12220_5.
5 | 1 R 2
5 | 0 R 1

Grading: 3 pts process, 1 pt reading remainders in reverse order.

(c) (4 pts) $AD834_{16}$ = 2554064 $_8$

$AD834_{16}$ = 1010 1101 1000 0011 0100_2 = 010 101 101 100 000 110 100_2
= 2 5 5 4 0 6 4

Grading: 1 pt base 2, 2 pts group by 3s from right, 1 pt final answer

2) (5 pts) Show the result of each of the following bitwise operations:

(a) (1 pt) 47 & 26 = 10

(b) (1 pt) 13 | 49 = 61

(c) (1 pt) 86 ^ 35 = 117

(d) (1 pt) 113 >> 3 = 14

(e) (1 pt) 5 << 5 = 160

101111 &	001101 |	1010110 ^
011010	110001	0100011
---	---	---
001010 (10)	111101 (61)	1110101 (117)

113 >> 3 = 1110001 >> 3 = 1110 (14)

5 << 5 = 101 << 5 = 10100000 (160)

Grading: 1 pt all or nothing

3) (15 pts) Assume that a `struct player` stores information for one player. To store a team, we could store an array of pointer to `struct player`, and then have each of the pointers point to a single `struct player`. For several teams, we would have an array of array of pointer to `struct player`. Write a function that takes in an array of integers, ***teamsizes***, and the size of that array, ***numteams,*** and dynamically allocates array of array of pointer to `struct player`, allocating an array of size ***numteams***, and then for each of the elements of this array, allocating an array of pointers of the appropriate size based on the array teamsizes. Finally, allocate memory to store each of the individual struct players. For example, if teamsizes = [3, 8, 15, 11], then there would be 4 pointers of type `struct player**`, 37 pointers of type `struct player*` and memory for 37 `struct player` variables allocated. Then return a pointer to the allocated memory.

```
struct player*** makeTeamsSpace(int* teamsizes, int numteams) {

  // Grading: 4 pts
  struct player*** res = malloc(numteams*sizeof(struct player**));

  // Grading: 1 pt
  for (int i=0; i<numteams; i++) {

    // Grading: 4 pts
    res[i] = malloc(teamsizes[i]*sizeof(struct player*));

    // Grading: 2 pts
    for (int j=0; j<teamsizes[i]; j++)

      // Grading: 3 pts
      res[i][j] = malloc(sizeof(struct player));
  }
  return res; // Grading 1 pt
}
```

4) (10 pts) Find a Big-Oh solution via the iteration technique to the following recurrence relation:

$$T(n) = 3T\left(\frac{n}{3}\right) + n, for\ n > 1$$
$$T(1) = 1$$

$T(n) = 3T\left(\frac{n}{3}\right) + n$ → first iteration
$T(n) = 3(3T\left(\frac{n}{9}\right) + \frac{n}{3}) + n$
$T(n) = 9T\left(\frac{n}{9}\right) + n + n$
$T(n) = 9T\left(\frac{n}{9}\right) + 2n$ → second iteration
$T(n) = 9(3T\left(\frac{n}{27}\right) + \frac{n}{9}) + 2n$
$T(n) = 27T\left(\frac{n}{27}\right) + n + 2n$
$T(n) = 27T\left(\frac{n}{27}\right) + 3n$ → third iteration

We guess after k iterations we have the following:

$T(n) = 3^kT\left(\frac{n}{3^k}\right) + kn$

Since T(1) = 1, let $\frac{n}{3^k} = 1$, which means that $n = 3^k$ and $k = log_3 n$. Plug in this value of k to yield:

$T(n) = nT\left(\frac{n}{n}\right) + n(log_3 n) = nT(1) + n(log_3 n) = n + n(log_3 n) = \boldsymbol{O(nlgn)}$

Grading: 1 pt first iteration, 1 pt second iteration, 2 pts third iteration, 2 pts guess, 2 pts plug in, 2 pts arrive at big oh result

5) (5 pts) Write a recursive function that takes in a positive integer, ***n***, and prints out the binary representation of ***n*** in reverse. So, if the initial call to the function had ***n*** = 22, then the function should print out "01101", without quotes.

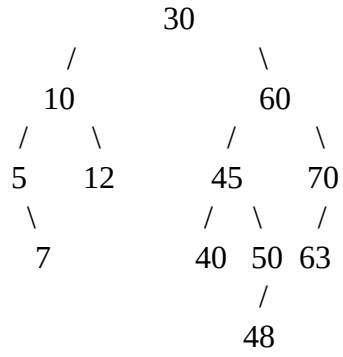
```
void printRevBin(int n) {

  if (n > 0) {            // Grading: 1 pt
    printf("%d", n%2);    // Grading: 2 pts
    printRevBin(n/2);     // Grading: 2 pts
  }
}
```

6) (10 pts) A treasure map is an n by n character array. Each character in the map is one of the following character: `' '`, `'*'`, or `'T'`. A space is a square that one can pass through. A star is a forbidden square. A capital T is a treasure. Complete the recursive function below so that it takes in the character array, ***map***, an integer, ***n***, representing the number of rows and columns in the character array, and two integers, ***r*** and ***c***, respectively, representing the starting location for a treasure hunt. This is guaranteed not to be a forbidden square. After collecting the treasure there (if there is any), your function should randomly pick a valid direction to move, either to the right (r, c+1), or down (r+1, c) and recursively start the search from there. If one of the two is blocked or out of bounds, go the other way, if both are blocked or out of bounds (base case), return the current score (0 or 1). At the conclusion of the function, return the total number of treasures gathered on the path.

```
int treasure(char** map, int n, int r, int c) {

    int score = 0;
    if (map[r][c] == 'T')
        score = 1;

    if (r == n-1) {
        if (c == n-1 || map[r][c+1] == '*') return score;
        return score + treasure(map, n, r, c+1);
    }
    if (c == n-1) {
        if (map[r+1][c] == '*') return score;
        return score + treasure(map, n, r+1, c);
    }
    if (map[r+1][c] == '*' && map[r][c+1] == '*') return score;
    if (map[r+1][c] == '*') return score + treasure(map, n, r, c+1);
    if (map[r][c+1] == '*') return score + treasure(map, n, r+1, c);

    if (rand()%2 == 0)
        return score + treasure(map, n, r+1, c);
    return score + treasure(map, n, r, c+1);
}
```

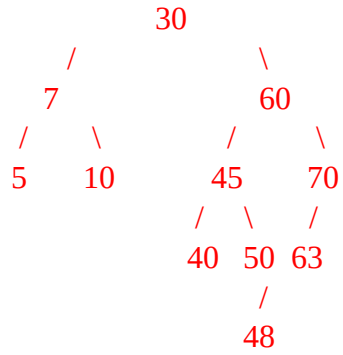
Grading: 2 pts first 0/1, 3 pts return when stuck, 3 pts go one way when there's only one choice, 2 pts randomly choose when we can.

7) (7 pts) Show the result of deleting 12 from the AVL tree shown below:



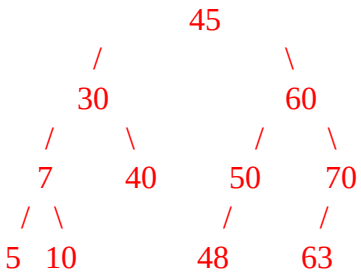
After first restructure:

Grading: 2 pts for 1st restructure



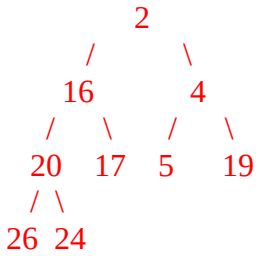
After second restructure:

Grading: 2 pts 45 root
1 pt 30 and 60, L and R
3 pts reattachments
automatic 0,1/7 if not valid AVL



8) (4 pts) Draw the binary heap stored in the array below.

Index	1	2	3	4	5	6	7	8	9
Value	2	16	4	20	17	5	19	26	24



Grading: All or nothing on the 4 pts.

9) (5 pts) Determine the summation shown below in terms of n. Put a box around your answer.

$$\sum_{i=2n+1}^{4n} 3i$$

$$\sum_{i=2n+1}^{4n} 3i = [3\sum_{i=1}^{4n} i] - [3\sum_{i=1}^{2n} i] = \frac{3(4n)(4n+1)}{2} - \frac{3(2n)(2n+1)}{2} = 3n(2(4n+1) - (2n+1)$$

$$= 3n(8n + 2 - 2n - 1) = 3n(6n + 1) = \mathbf{18n^2 + 3n}$$

Grading: 1 pt split, 1 pt plug in each formula, 2 pts to simplify answer

10) (9 pts) An algorithm to process an array of size n runs in O(nlgn) time. On an array of size 2^{18}, the algorithm takes 50 ms. How long will the algorithm take on an array of size 2^{24}? (Put a box around your answer.)

Let the time the algorithm takes on an input of size n be T(n) = cnlgn. Using the given information we have

$$T(2^{18}) = c(2^{18})(lg2^{18}) = 50ms \rightarrow c = \frac{50mc}{(2^{18})(18lg2)}$$

Now, solve for $T(2^{24})$:

$$T(2^{24}) = \frac{50mc}{(2^{18})(18lg2)} \times 2^{24} \times (lg2^{24}) = \frac{(50ms) \times (2^6) \times 24(lg2)}{18lg2} = \frac{4}{3} \times 50 \times 64ms$$

$$= \frac{12800}{3} ms = \frac{12.8}{3} sec = \mathbf{4.}\overline{\mathbf{26}}\ \boldsymbol{sec}$$

Grading: 3 pts find c, 3 pts plug in 2^{24}, 3 pts simplify to a reasonable answer

11) (10 pts) A "dominating" node in a binary tree is one that stores a value greater than both of its children. If a node has no children, it's trivially a dominating node. If a node has only one child, it's a dominating node if the value in the node is greater than the one child. Write a ***recursive*** function that takes in a pointer to a binary tree node and returns the number of dominating nodes in the binary tree rooted at that node.

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

int numdominating(bintreenode* root) {

    if (root == NULL) return 0;     // Grading: 1 pt

    // Grading: 2 pts
    if (root->left == NULL && root->right == NULL) return 1;

    // Grading: 4 pts
    int score = 1;
    if (root->left != NULL && root->data <= root->left->data)
        score = 0;
    if (root->right != NULL && root->data <= root->right->data)
        score = 0;

    // Grading: 3 pts
    return score + numdominating(root->left)
                 + numdominating(root->right);
}
```

12) (5 pts) Consider using the strategy of quadratic probing in a hash table of size 137 (valid indexes 0 through 136, inclusive). If the initial hash value of an element to insert is 134, what are the ***next*** five indexes after 134 that would be candidates for insertion of the element:

135 , 1 , 6 , 13 , 22

Grading: 1 pt per slot

13) (15 pts) Huffman Coding provides a prefix free code. Imagine a language that adheres to the same principle: no valid word in the language is a prefix of another valid word in the language. In that case, a string of words with spaces omitted could be properly interpreted. For example, if the valid words were, “apple, “ate”,”adam”, “the”, “after” and “slept”, then we could take the string “adamatetheappleafteradamslept” and confidently split the string into separate words as follows: “adam ate the apple after adam slept.” Write an ***iterative*** function that takes in a pointer to the root of a trie and a single string that is the valid concatenation of words in the trie, and returns an array of integers storing the lengths of each word in the string, in sequence. To be usable, this function needs to also store in a pointer ptrNumWords, the number of words the sentence has. (Since you don’t know how long the array should be, initially allocate it to be the size of the input string, then resize it to the right size at the end of the function before returning it.)

```
typedef struct trie {
    int isWord;
    struct trie* next[26];
} trie;

int* wordlengths(trie* root, char* str, int* ptrNumWords) {

    int n = strlen(str);                        // Grading: 1 pt
    int* res = calloc(n, sizeof(int));          // Grading: 1 pt
    int i = 0, j = 0;                           // Grading: 1 pt

    while (i<n) {                               // Grading: 1 pt

        trie* tmp = root;                       // Grading: 1 pt
        int oldi = i;
        while (!tmp->isWord) {                  // Grading: 1 pt
            tmp = tmp->next[str[i]-'a'];        // Grading: 1 pt
            i++;                                // Grading: 1 pt
        }
        res[j] = i - oldi;                      // Grading: 2 pts
        j++;                                    // Grading: 1 pt
    }

    res = realloc(res, j*sizeof(int));          // Grading: 2 pts
    *ptrNumWords = j;                           // Grading: 1 pt
    return res;                                 // Grading: 1 pt
}
```

14) (5 pts) Show the result of each iteration of insertion sort running on the array below.

Array	12	3	9	7	1	8	4
1st iter	3	12	9	7	1	8	4
2nd iter	3	9	12	7	1	8	4
3rd iter	3	7	9	12	1	8	4
4th iter	1	3	7	9	12	8	4
5th iter	1	3	7	8	9	12	4
6th iter	1	3	4	7	8	9	12

Grading: 1 pt per row, whole row has to be right to get the point.

15) (5 pts) Show the result of running the partition algorithm shown in class on the array below, using the leftmost element as the partition element.

Original	10	3	16	18	14	8	9	15	2	6
Partition	8	3	6	2	9	10	14	15	18	16

Grading: ½ per slot, round down

16) (4 pts) Consider the array below going through a Merge Sort. What does the array look like right before the last call to the Merge function?

Original	13	6	9	1	2	18	15	8
Before Last Merge	1	6	9	13	2	8	15	18

Grading: ½ per slot, round down

17) (1 pt) In what city is the majority of the hit Netflix series “Emily in Paris” set? **Paris (Give to All)**