Spring 2017 COP 3502 Final Exam: Part A Solutions

Date: 4/27/2017

1) (15 pts) Use the iteration technique to determine a close form solution for the recurrence relation $T(n)$ defined below. Note: due to the nature of this recurrence, it's possible to get an exact solution for $T(n)$, so please try to do that instead of just getting a Big-Oh bound.

$$T(n) = 2T(n-1) + 2^n$$
$$T(1) = 2$$

Solution

Iterate as follows:

$$\begin{aligned} T(n) &= 2T(n-1) + 2^n \\ &= 2(2T(n-2) + 2^{n-1}) + 2^n \\ &= 4T(n-2) + 2^n + 2^n \\ &= 4T(n-2) + 2(2^n) \\ &= 4(2T(n-3) + 2^{n-2}) + 2(2^n) \\ &= 8T(n-3) + 2^n + 2(2^n) \\ &= 8T(n-3) + 3(2^n) \end{aligned}$$

In general, after k iterations, we'll have:

$$= 2^k T(n-k) + k(2^n)$$

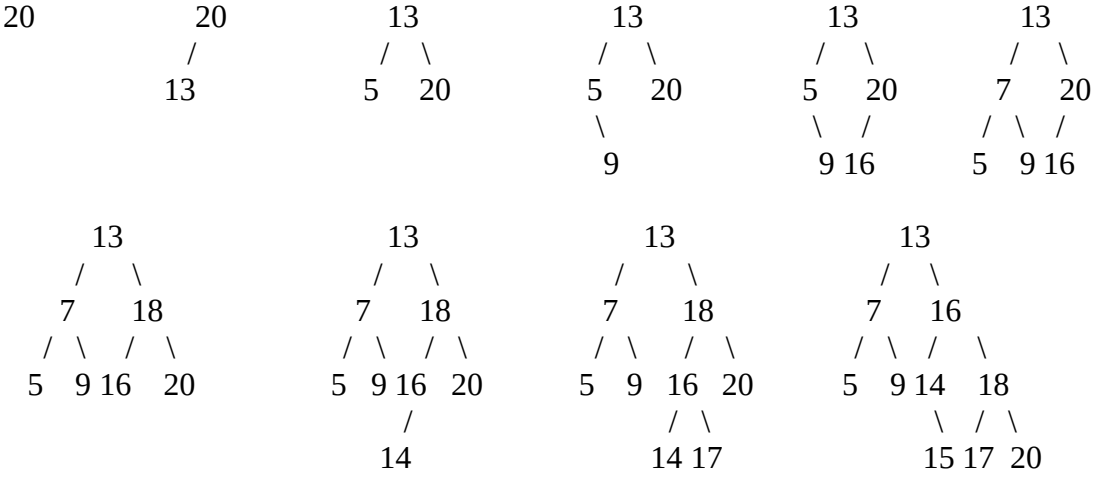
Plug in k = n-1 to obtain

$$\begin{aligned} T(n) &= 2^{n-1}T(n-(n-1)) + (n-1)(2^n) \\ &= 2^{n-1}T(1) + (n-1)(2^n) \\ &= 2^{n-1}(2) + (n-1)(2^n) \\ &= 2^n + (n-1)(2^n) \\ &= n2^n \end{aligned}$$

Grading: 2 pts each iteration, 3 pts general guess, 2 pts to plug in k = n-1, 4 pts for all of the algebra at the end.

2) (10 pts) Consider inserting the following items into an AVL tree in the following order: 20, 13, 5, 9, 16, 7, 18, 14, 17, and 15 . Show the state of the tree after the completion of each insertion. Draw a box around each of your answers. There should be 10 boxes.

Solution



Grading: 1 pt per tree, give carry over points if the insertion into the given tree is (a) an insertion into a VALID AVL tree, and (b) the insertion itself is correct for the VALID AVL tree it was into. If this isn't the case, then don't award points. (So a single error can result in losing more than 1 point because the resulting insertion isn't well-defined.)

Questions 3 - 7 will involve the solution to the following problem:

You are a collector of rare Star Wars figurines. Your goal is to collect as many unique figurines in a particular set of 30. You are attending a convention where there are up to 20 vendors. Each vendor only sells a single package with some subset of the figurines. Each vendor has a fixed price for their package. Unfortunately, you have a limited budget. Your goal is to figure out which packages to buy (essentially a subset of the vendors) that fit within your budget so that you maximize the number of unique figurines you obtain. (Assume that you don't have any before the convention.)

To solve the problem we will do the following:

1) We will store two integers for each vendor, v_i: set_i and $cost_i$. The first will be an integer to be represented as a bit-mask, storing the subset of figurines (numbered 0 through 29) while the second will be the cost of buying that subset of figurines from that vendor. For example, if set_i = 29 and $cost_i$ = 2045, then you could obtain figurines 0, 2, 3, 4 (note that $2^0 + 2^2 + 2^3 + 2^4 = 29$) at a cost of $2045.

2) We will iterate through each subset of vendors. In doing so we will want to calculate two things for each subset of vendors: (1) the total cost of buying all of the packages sold by the subset of vendors and (2) the number of unique figurines we would get if we bought each of those vendors' packages.

You will write functions that perform the following tasks related to this problem:

3) A function that reads in the set of figurines sold by each vendor and how much that set costs and fills in two arrays, set and cost, to store the bitmask representation of the set of figurines that each vendor is selling and the sets corresponding cost.

4) A function that takes in an array of costs of packages of figurines, the length of that array, and a single integer representing a subset of packages and returns the total cost of buying all of the packages of figurines designated by that subset.

5) A function that takes in an array of integers representing the sets each vendor sells, the length of that array, and a single integer representing a subset of packages and returns a single integer bitmask representing all unique figurines found within that subset of packages.

6) A function that takes in a bitmask and returns the number of items (ie the number of bits set to 1) represented in that bitmask.

7) A function that calls the other four functions to solve the problem.

3) (7 pts) Assume that in main, the number of vendors was read in. This value will be passed into the function for this question as an integer n. In this function, you will read in n lines of input from standard input, each containing information about one vendor. The goal of your function will be to read in this information and fill the two arrays passed as parameters (set and cost) with the appropriate bitmask and cost in each array index corresponding to each vendor. Each line of input starts with a positive integer k ($k \leq 30$), representing the number of figurines that vendor has to sell. This is followed by k unique integers in the range [0, 29], representing the figurines in that vendor's package. This is followed by one last integer, c, indicating the cost of the package sold by that vendor. Fill in the incomplete portions of the function below:

```
void readVendors(int set[],int cost[], int n) {

    int i;
    for (i=0; i<n; i++) {

        int numF, bit, j;

        set[i] = 0 ;                    // 1 pt

        scanf("%d", &numF);
        for (j=0; j<numF; j++) {
            scanf("%d", &bit);

            set[i] |= (1<<bit) ;        // + also works
        }                               // 2 pts set[i]
        scanf("%d", &cost[i]);          // 1 pt +=, |=
    }                                   // 3 pts 1<<bit
}
```

4) (7 pts) Complete this function so that it takes in the cost array (read in question 3), its length, and a bitmask representing a subset and returns the sum of the items in the cost array indicated by the bitmask. For example, if the bitmask = 13, the function should return cost[0]+cost[2]+cost[3], since $13 = 2^0 + 2^2 + 2^3$.

```
int sumCost(int cost[], int n, int bitmask) {
    int res = 0, i;
    for (i=0; i<n; i++)

        if ( (bitmask & (1<<i)) != 0 )  // 5 pts - 1 pt bitmask
                                        // 1 pt &, 2 pts 1<<i, 1 pt !=0
            res = res + cost[i] ;  // 2 pts
    return res;
}
```

Grading note: Many ways to do the if that are different, so check each one…

5) (7 pts) Complete this function so that it takes in the set array (initialized in question 3), its length and a bitmask representing a subset of items from the set array and returns a new bitmask representing the set of figurines that would be obtained if each of the sets of figurines indicated by the bitmask were purchased. For example, if bitmask was 13 (representing vendors 0, 2 and 3, and vendor 0 sold the subset 35 (figurines 0, 1 and 5), vendor 2 sold the subset 129 (figurines 0 and 7) and vendor 3 sold the subset 30 (figurines 1, 2, 3 and 4), then your function should return 191, representing that with all of these vendors we can get figurines 0, 1, 2, 3, 4, 5 and 7.

```
int setBought(int set[], int n, int bitmask) {
    int res = 0, i;
    for (i=0; i<n; i++)

        if ((bitmask &(1<<i)) != 0  ) // 5 pts same as #4

            res = res | set[i]; // 2 pts, must be |
    return res;
}
```

6) (7 pts) Complete the function below so that takes in an integer n and returns the number of bits set to 1 in the binary representation of n. ***In order to get full credit, please use bitwise operators.*** (Note: 4 pts out of 7 will be given to correct solutions that don't use bitwise operators.) You may assume that the integer n has 32 bits.

```
int bitCount(int n) {

    int i, res = 0;
    while (n > 0) {
        res += (n&1);
        n >>= 1;
    }
    return res;

}
```

Grading: lots of ways to do this…so criteria is "flexible" to different methods:

Reasonable loop mechanism: 2 pts
Correctly isolating a single bit: 2 pts
Adding 1 per bit on: 2 pts
Returning the result: 1 pt

7) (10 pts) In this function you'll put everything together. This function takes in the array set, the array cost, the length of both arrays and the budget for buying figurines. It will return the most number of unique figurines you can buy that cost less than or equal to the budget. Please fill in the blanks indicated:

```
int solve(int set[], int cost[], int n, int budget) {

    int best = 0, i;

    for (i=1; i< (1<<n) ; i++) {          // 2 pts

        int mycost = sumCost( cost , n , i ); // 1 pt each

        if ( mycost <= budget ) {              // 2 pts

            int numFig = bitCount(setBought(set,n,i)); // 3 pts

            if (numFig > best) best = numFig;
        }
    }
    return best;
}
```

8) (15 pts) Consider a binary search tree of integers where each node stores the height of that node (from the bottom of its subtree). Note: the height stored in this struct is consistent with the height described for each node in class for AVL trees. The struct for such a tree is as follows:

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

Write a ***recursive*** function that takes in a pointer to a `btnode root`, and an integer value to insert into the tree, and creates a new node to store value, inserts that node into the binary search tree and returns a pointer to the resulting root of the tree. You may assume that all values in the tree are distinct and that the newly inserted value isn't equal to any value currently in the tree. ***Your function must also adjust the heights of each node in the tree for which the heights have changed.***

You may assume you can call a function int max(int a, int b) which returns the maximum of its two parameters.

```
int max(int a, int b);
```

```
btnode* insert(btnode* root, int value) {

    if (root == NULL) {
        btnode* res = malloc(sizeof(btnode));
        res->data = value;
        res->height = 0;
        res->left = NULL;
        res->right = NULL;
        return res;
    }

    if (value < root->data)
        root->left = insert(root->left, value);
    else
        root->right = insert(root->right, value);

    if (root->left != NULL)
        root->height = root->left->height + 1;
    if (root->right != NULL)
        root->height = max(root->height, root->right->height+1);

    return root;
}
```

Grading: 5 pts NULL case, 4 pts for recursive call left or right, 6 pts updating the height (2 pts for correct general logic for both sides and 2 pts for avoiding NULL ptr errors).

9) (5 pts) An O(nlgn) algorithm takes half a second to run on an input of size 2^{20}. Roughly, how long will it take (in seconds) to run on an input of size 2^{25}, in seconds? (3 pts for setting up the arithmetic for the solution, 2 pts for reducing your answer without a calculator to a time, in seconds.)

Solution

Let T(n) be the run-time for the algorithm on an input of size n. The given information can be represented as follows:

$$T(n) = cnlgn$$
$$T(2^{20}) = c2^{20}\lg(2^{20}) = .5\ sec$$
$$c(20)2^{20} = .5\ sec$$
$$c = \frac{.5}{20 \times 2^{20}}\ sec$$

Now, solve for $T(2^{25})$:

$$T(2^{25}) = c(2^{25})(lg2^{25}) = \frac{.5}{20 \times 2^{20}} \times 2^{25} \times 25\ sec = \frac{.5 \times 5 \times 2^5}{4}\ sec = \mathbf{20\ sec}$$

Grading: 2 pts solving for c, 1 pts plugging in $n=2^{25}$, 2 pts simplifying to final answer (can give partial if there is an arithmetic error along the way)

10) (5 pts) Convert 211021_3 to base 7. Put your final answer on the line shown at the bottom of the page. Show all of your work. You will only receive full credit if you used the method shown in class to do both conversions.

Solution

$211021_3 = 2 \times 3^5 + 1 \times 3^4 + 1 \times 3^3 + 2 \times 3^1 + 1 = 2 \times 243 + 81 + 27 + 6 + 1 = \mathbf{601}$

7 | 601
7 | 85 R 6
7 | 12 R 1
7 | 1 R 5
7 | 0 R 1

The final result is $\mathbf{1516_7}$.

Note: We can double check by seeing that $1 \times 7^3 + 5 \times 7^2 + 7 + 6 = 343 + 245 + 13 = 601$.

Grading: 3 pts to get 601, 2 pts to get 1516.

11) (10 pts) Write a segment of code that allocates space for a two dimensional array of ***pointers to blob*** with dimensions r and c, and then sets each of those pointers to point to newly allocated uninitialized space for a single blob. You may assume that blob is a typedef for a struct and that the size of a single blob is just sizeof(blob). In addition to grid, blob, i and j, you may use r and c (already set). The first line is given to you.

```
int i,j;
blob*** grid = malloc(r*sizeof(blob**));

for (i=0; i<r; i++) {
    grid[i] = malloc(c*sizeof(blob*));
    for (j=0; j<c; j++)
        grid[i][j] = malloc(sizeof(blob));
}
```

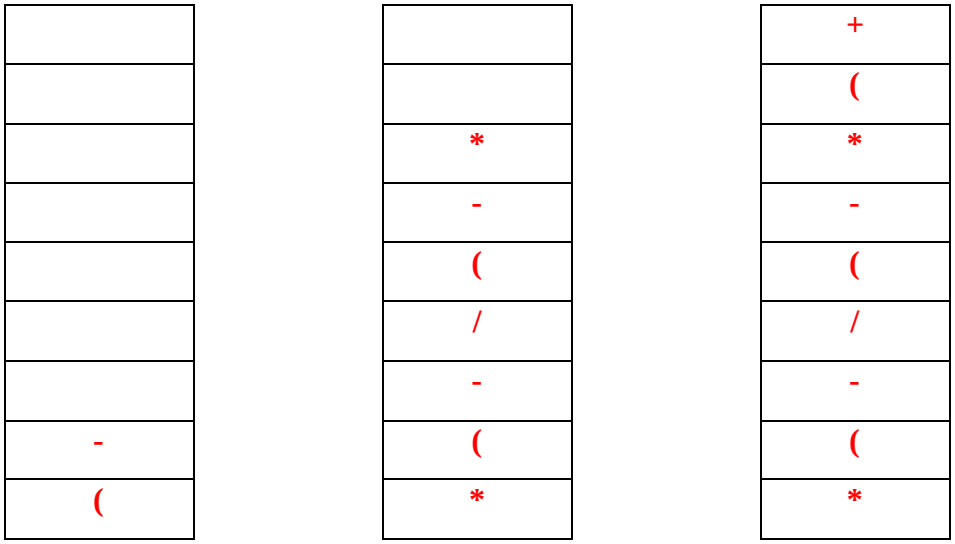
Grading: 1 pt r for loop (must run to r)
4 pts - 1 pt LHS, 1 pt malloc, 1 pt c, 1 pt sizeof(blob*)
1 pt c for loop (must run to c)
4 pts grid[i][j] malloc - 1 pt LHS, 1 pt malloc, 2 pts sizeof(blob)

12) (2 pts) What programming language is named in honor of mathematician Ada Lovelace?

Ada

13) (10 pts) Convert the following infix expression to postfix, showing the contents of the operator stack at the three points A, B and C indicated during evaluating the expression and provide the converted postfix expression.

```
                        A                       B       C
( 3 + 6 ) * ( 7 -   27 / ( 19 - 2 *  ( 4 +  1 ) ) )
```



A B C

Resulting postfix expression:

3 6 + 7 27 19 2 4 1 + * - / - *

Grading: 1 pt first stack, 2 pts second stack, 2 pts third stack, 5 pts expression

14) (5 pts) Show the result of insertion sort after each iteration when sorting the array shown below:

Array	13	6	9	3	12	5	7
After 1st iteration	**6**	**13**	**9**	**3**	**12**	**5**	**7**
After 2nd iteration	**6**	**9**	**13**	**3**	**12**	**5**	**7**
After 3rd iteration	**3**	**6**	**9**	**13**	**12**	**5**	**7**
After 4th iteration	**3**	**6**	**9**	**12**	**13**	**5**	**7**
After 5th iteration	**3**	**5**	**6**	**9**	**12**	**13**	**7**
After last iteration	3	5	6	7	9	12	13

Grading: 1 pt per row, point is only earned if row is perfect, no exceptions.

15) (3 pts) Consider a Merge Sort running on an array of size n = 16, using the code shown in class (the base cases are arrays of size 0 and 1, otherwise recursive calls are made to Merge Sort). How many times will the Merge function get called during the sort?

Solution
On an array of size 2, Merge gets called 1 time. On an array of size 4, it gets called 3 times (once for the recursive call to Merge Sort on the left, once on the right, and once to put the two sides together). Similarly, on an array of size 8, Merge gets called 3 (left) + 3 (right) + 1 (together) = 7 times. Finally, on an array of size 16, Merge gets called 7 (left) + 7 (right) + 1 (together) = 15 times. In general, Merge gets called 2^n - 1 times on a Merge Sort of 2^n items.

15

Grading: 3 pts for 15, 2 pts for 7, 14 or 16, 1 pt for anything else within 8, 0 pts otherwise

16) (6 pts) Consider the following game. You are given a positive integer, *n*. Your goal is to change that integer to a number 0 or less. You are allowed two possible operations to reduce your number: (1) subtract 10 from it, (2) integer divide it by 3. Write a ***recursive*** function that calculates the minimum number of operations you have to apply in sequence to reduce your number to 0 or less. (For example, if your number was 19, you could divide it by 3 to obtain 6 and then subtract 10 to obtain -4, to obtain the goal in 2 steps. There is no way to do this in 1 step, so the correct answer is 2 for this case.) **Hint: recursively try both moves, see which one "wins" the game for you more quickly and build off that move.**

My Initial Intended Solution
```
int minMovesToWinSlow(int n) {
    if (n == 0) return 0;
    if (n <= 10) return 1;
    int div3 = minMovesToWinSlow(n/3) + 1;
    int sub10 = minMovesToWinSlow(n-10) + 1;
    return min(div3, sub10);
}
```

My Better Solution, Still Recursive
```
int minMovesToWin(int n) {
    if (n == 0) return 0;
    if (n <= 10) return 1;
    return minMovesToWin(n/3) + 1;
}
```

Grading: 4 pts base cases, 6 pts recursive call or calls.

17) (1 pt) What type of food is served at Blaze Pizza? **Pizza** **(Give to All)**