COP 3502 Section 2: Recurrence Relations, Data Structures Tracing

Assigned: Friday, July 17, 2020
Due: Friday, July 24, 2020 (via Webcourses Section 2)

Directions: Please either type your answers in a .doc or .docx file or write on paper and scan your answers into a SINGLE .pdf file and submit that file over Webcourses. (Thus, the accepted file types will be .doc, .docx and .pdf.) Check Webcourses for the specific time the assignment is due.

Recurrence Relation Problems

1) Find the Big-Oh solution to the following recurrence relation using the iteration technique. Please show all of your work, including 3 iterations, followed by guessing the general form of an iteration and completing the solution. Full credit will only be given if all of the work is accurate (and not just for arriving at the correct answer.)

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

2) Find the Big-Oh solution to the following recurrence relation using the iteration technique. Please show all of your work, including 3 iterations, followed by guessing the general form of an iteration and completing the solution. Full credit will only be given if all of the work is accurate (and not just for arriving at the correct answer.)

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

3) Using the iteration technique, find a tight Big-Oh bound for the recurrence relation defined below:

$$T(n) = 3T\left(\frac{n}{2}\right) + n^2, \text{ for } n > 1$$
$$T(1) = 1$$

Hint: You may use the fact that $\sum_{i=0}^{\infty}(\frac{3}{4})^i = 4$ and that $3^{log_2 n} = n^{log_2 3}$, and that $log_2 3 < 2$.

4) Use the iteration technique to solve the following recurrence relation in terms of n:

$$T(n) = 3T(n-1) + 1, for\ all\ integers\ n > 1$$
$$T(1) = 1$$

Please give an exact closed-form answer in terms of n, instead of a Big-Oh answer.
(Note: A useful summation formula to solve this question is $\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$.)

AVL Tree Trace Problems

5) (a) Create an AVL tree by inserting the following values in the order given: 38, 72, 58, 16, 3, 24, 8, and 15. Show the state of the tree after each insertion.

(b) Draw the state of the tree after the deletion of the node containing the value 16.

6) Show the result of inserting the following values into an initially empty AVL tree:

10, 7, 3, 22, 16, 13, 5, 18, 20 and 19.

Draw a box around your result ***after each insertion.***

7) (a) Create an AVL tree by inserting the following values into an initially empty AVL Tree in the order given: 7, 8, 54, 13, 35, 66, 50, and 12. Show the state of the tree after each insertion.

(b) Draw the state of the tree after the deletion of the node containing the value 7.

Binary Heap Problems

8) Show the state of a binary heap (min heap) after the insertion of the following items, in this order, into an initially empty binary heap:

13, 2, 19, 16, 14, 3, 9, 12, 6, 2, 18, 11 and 8

9) Show the array representation of the final state of the heap from the previous question.

Index	1	2	3	4	5	6	7	8	9	10	11	12	13
Value													

10) Delete the minimum value from the heap in question 8 and show a picture of the resulting heap.

Hash Table Problems

11) Use the following hash function to insert the given elements into the hash table below. Use **quadratic probing** to resolve any collisions. You may assume that the correct table size (in this case, 10) is always passed to the function with the key that is being hashed.

```
int hash(int key, int table_size)
{
     int a = (key % 100) / 10;
     int b = key % 10;
     return (a + b) % table_size;
}
```

Keys to insert (one by one, in the following order): 2555, 1523, 5893, 800, 956

0	1	2	3	4	5	6	7	8	9

12) There are two hash functions that take in strings as input shown below. Each returns an integer in between 0 and 1,000,002. (Note: 1,000,003 is a prime number.) Which of these two is a better hash function? Explain the weakness in the other function.

```
int f1(char* str) {

    int i = 0, res = 0;
    while (str[i] != '\0') {
        res = (256*res + (int)(str[i]))%1000003;
        i++;
    }
    return res;
}

int f2(char* str) {

    int i = 0, res = 0;
    while (str[i] != '\0') {
        res = (res + (int)(str[i]))%1000003;
        i++;
    }
    return res;
}
```

13) Consider using a hash table of size 16 to store integers, using the linear probing technique to resolve collisions and the following hash function (intended to take non-negative integer input):

```
int f(int n) {
    int res = 0;
    while (n > 0) {
        res = res ^ (n&15);
        n = (n >> 4);
    }
}
```

Show the state of the hash table after executing the following insertions, in this order:

182, 92, 41, 130, 111, 105, 94, and 43.