CSC 212 Spring 2019 Midterm Exam 2 Study Guide

On Monday we will be holding a review session for the midterm where we will go over answers to these questions.

1. Pointers

   (a) Write a small function which takes a reference to an int and a regular int as an argument, sets the referenced int equal to the other int then returns void.

   ```
   void set(int *a, int val) {
       *a = val;
       return;
   }
   ```

   (b) Declare an array of 12 ints on the *heap.* Fill it with 1-12 in reverse order using your above function then deallocate it.

   ```
   int main() {
       int *a = new int[12];
       for(int i = 1; i <= 12; i++) {
           set(&a[12-i], i);
       }
       delete[] a;
       return 0;
   }
   ```

   (c) Describe the difference between "dot notation" vs "arrow notation"
   ie. myList.print() vs myList→ $print()$

   ```
   myList.print() calls the 'print' method from the LL
   class on a list object myList.

   myList->print() is short for (*myList).print. You use this on
   pointers to objects/structs

   eg.
   LL *myList = new LL();
   myList->print();
   delete myList;
   ```

Recursion

1. What is recursion? What types of problems are good for recursion?

   ```
   - Recursion is useful when it would be extremely complex to write
   iterative code.
   - For functions where you must divide and conquer (split arrays, find
   max of weakly unimodal arrays, etc.).
   ```

2. Draw out a call tree for the following function given foo(5). What is the Big-O runtime of this function?

   ```
   void foo(int n) {
       if(n < 2) {
           return n;
       } else {
           return foo(n-1) + foo(n-2);
       }
   }

   O(2^n)
   ```

3. Write a recursive function which takes an array if ints and its length as arguments and prints out the odd numbers. What is the runtime of this function? Justify your answer.

   ```
   void onlyOdd(int arr[], int n) {
       if(n == 0) {
           std::cout << std::endl;
       }
       else if(n % 2 != 0) {
           std::cout << arr[0] << " ";
       }
       return onlyOdd(arr + 1, n - 1);
   }
   ```

Binary Search

1. What is the runtime of binary search? Is it possible to improve the runtime why or why not?

   ```
   Binary search runs in O(log n). It is not possible to improve this
   runtime.
   ```

Unimodal Array

1. Describe the difference between a strongly and weakly unimodal array.

```
Strongly unimodal can be split into an increasing and a decreasing part
Weakly unimodal can be split into a non-in and non-dec part
```

2. How can you most efficiently find the maximum value of each type of array?

```
Strong: Traverse the array until element a[i] > a[i+1], you have found
the max
Weak: You must split the array and solve the problem recursively.
```

Recurrences

Solve the following relations using the unrolling method

1. T(0)=1, T(n)=T(n-4)-4

```
T(n) = T(n-8)-8
     = T(n-12)-12
     = T(n-4k)-4k       n-4k = 0 -> n = 4k -> k = n/4
     = T(0) - n => 1-n
```

2. T(1)=1, T(n)=T(n/3)+1

```
T(n) = T(n/9) + 2
    = T(n/27) + 3
    = T(n/3^k) + k      n/3^k = 1 -> n = 3^k -> k = log3 n
    = T(1) + log3 n = log3 n + 1
```

3. T(1)=1, T(n)=2T(n/2)+n

```
T(n) = 4T(n/4) + 2n
    = 8T(n/8) + 3n
    = 2^kT(n/2^k) + kn      n/2^k = 1 -> n = 2^k -> k = log2 n
    = nT(1) + n log2 n => n + n log2 n
```

4. T(0)=1, T(n)=T(n-1)+2

```
T(n) = T(n-1) + 2
    = T(n-2) + 4
    = T(n-3) + 6
    = T(n-k) + 2k       n-k = 0 -> n = k
    = T(0) + 2n => 2n + 1
```

5. Define and solve the recurrence of the following function. Assume each print statement takes 1 time.

```
void bar(int n) {
    if(n == 0) {
        std::cout << "fin" << std::endl;
    } else {
        std::cout << n << " ";
        bar(n-1);
        bar(n-1);
        bar(n-1);
    }
}

T(0) = 1, T(n) = 3T(n-1) + 1
T(n) = 3T(n-1) + 1
    = 3[3T(n-2) + 1] + 1 => 9T(n-2) + 1 + 3
    = 9[3T(n-3) + 1] + 1 + 3 => 27T(n-3) + 1 + 3 + 9
    = 3^kT(n-k) + sum{i = 0} {k-1} 3^i      n-k = 0 -> n = k

    // explain geometric series formula

    = 3^n + (1 - 3^n)/(1-3)
```

6. What is the Big-O runtime of this function?

    `O(3^n)`

Mergesort

1. What is the worst case runtime of mergesort?

    `n log n`

2. What is the main advantage of mergesort vs quicksort?

    ```
    Quicksort has a worst case runtime of O(n^2), mergesort has
    a worst case of O(n log n). Mergesort is also stable, where quicksort
    is not.
    ```

Quicksort

1. Illustrate the state of the array below after applying the quicksort partition method from the lecture slides, once.

    `[13, 15, 19, 20, 9, 11, 8, 6, 4, 14, 1, 10]`

```
[6, 10, 1, 4, 9, 11, 8, 13, 20, 14, 19, 15]
                        p
```

2. What index value will be returned by partition?

   pivot index is 7

3. What is the worst case of this particular partition method? What input would cause this behavior?

   Worst case is maximally unbalanced partition, if the pivot ends up being the min or max in the array.

4. What is one partition choice which is less likely to encounter worst-case behavior?

   The center or a random point would minimize the chance of worst case every time.

5. In what situation would you be likely to choose quicksort over mergesort?

   Since quicksort runs in-place, it is a good choice in situations where memory is limited.

Linked Lists

1. You are implementing a CDLL with members *head* and *tail* and Nodes with members *data, next*, and *prev.* Write a method which appends a node to the end of the list. What is the runtime of this function?

```
void CDLL::append(int val) {
    if(!head) { // empty list
        node *p = new node(val);
        head = p;
        tail = p;
        p->next = p->prev = p;
    }
    else if(head == tail) { // single element list
        node *p = new node(val);
        p->next = head;
        p->prev = tail;
        tail->next = p;
        head->prev = p;
        tail = p;
    }
    else { // list with at least two elements already exists
        node *p = new node(val);
        p->next = head;
        p->prev = tail;
        tail->next = p;
        head->prev = p;
        tail = p;
    }

    // last two cases actually the same, so you can
    actually just have one if youd like
}
```

2. Write a method for a SLL class with members *head* and *tail*, which returns the average of the elements in the list. Nodes have members *next* and *data* (which stores a double). What is the runtime of this function?

```
double SLL::average() {
    if(!head) {
        return 0.0
    }

    node *curr = head;

    double sum = 0.0;
    double num = 0;

    while(curr) {
        sum += curr->data;
        num++;
        curr = curr->next;
    }

    return (sum/num);
}
```

`Function is O(n), must traverse the entire n-element list`

3. What is one advantage of a linked list vs a traditional array? One weakness?

`Linked Lists can add or remove elements easier than traditional arrays. Unfortunately, this means that they have O(n) time random access` as compared to arrays' O(1) random access

Stacks

1. Draw out the contents of each stack after the function has concluded

```
int main() {
    stack<int> p;
    stack<int> q;

    p.push(1);
    p.push(3);
    q.push(2);

    for(int i = 0; i < p.size(); i++) {
        q.push(p.top());
        q.push(i);
    }
    q.pop()

    /* evaluate stack contents at this point */

    return 0;
}
```

p: 1 3

q: 2 3 0 3

2. Explain the difference between LIFO and FIFO data structures. Which of these is stack?

LIFO means whatever goes in last comes out first, "Last In First Out", like a stack of coins. Whatever is at the top gets taken off first.

FIFO means first come, first served or "First In First Out"; FIFO. Like the line at the grocery store. You may have heard lines like this referred to as queues. "Queue up to sign in" etc.