You have to do all the 5 problems in this section of the exam. Partial credit cannot be given unless all work is shown and is readable. Be complete, yet concise, and above all _be neat_. 
1. [10 pts] Circle the correct choices in each of the following parts:

(i) The worst case complexity of searching for a value in an unsorted array of n integers is
   a) O(1)       b) O(log n)    c) O(n)    d) O(n log n)

(ii) The worst case complexity of dequeuing one item from a queue using an array implementation is
     a) O(1)       b) O(log n)   c) O(n)    d) O(n log n)

(iii) The time complexity of attaching a linked list containing k elements at the end of another linked list containing j elements would be
      a) O(j)       b) O(k)      c) O(j+k)  d) O(jk)

(iv) The status of function calls during the execution of a computer program is best modeled using which of the following
     a) stack       b) queue     c) binary search tree

(v) An infix expression is being converted to its postfix form using a stack. The character read from the expression is ‘+’ and the stack contains the following elements.

If the character read from the expression is ‘+’, the stack should look like

Answer: b (Grading: 2 pts each)
2. [8 pts] Trace the following function when it is called from the main program through, \( simple(113) \), and give the final value returned to main().

```c
int simple ( int n)
{
    if (n < 2)    return n;
    else
        return n%2 + simple(n/2);
}
```

\( simple(113) = \)

\[ 1 + simple(56) = \]

\[ 1 + 0 + simple(28) = \]

\[ 1 + 0 + 0 + simple(14) = \]

\[ 1 + 0 + 0 + 0 + simple(7) = \]

\[ 1 + 0 + 0 + 0 + 1 + simple(3) = \]

\[ 1 + 0 + 0 + 0 + 1 + 1 + simple(1) = \]

\[ 1 + 0 + 0 + 0 + 1 + 1 + 1 = \]

\[ 4 \]

Grading: 1 pt for each step
3. [12 pts] Write the recurrence relation for this function and work out the worst case time complexity for it, using the iteration technique.

```c
1 int modpower(int a, int n, int mod) {
2     if (n == 0) return 1;
3     if (n == 1) return a % mod;
4     answer = power(a, n/2, mod);
5     if (n%2 == 0)
6         return (answer*answer) % mod;
7     else
8         return (answer*answer*a) % mod;
9 }
```

Let $T(n)$ represent the running time of this function, where $n$ represents the exponent in the problem. Then we have the following recurrence relation:

$$T(n) = T(n/2) + O(1)$$

because whenever the function is called with the parameter $n$, a single call is made to the function with a parameter $n/2$, plus a constant amount of work. We solve this recurrence relation using iteration:

$$T(n) = T(n/2) + 1$$

$$= T(n/4) + 1 + 1 \quad (1 \text{ pt})$$

$$= T(n/8) + 1 + 1 + 1 \quad (2 \text{ pt})$$

From here, we deduce the general pattern after $k$ iterations:

$$= T(n/2^k) + k \quad (3 \text{ pts})$$

We want to iterate until we get to $T(1)$. This occurs when $n/2^k = 1$. Thus, we find that $n = 2^k$ and $k = \log_2 n$. (2 pts)

Thus, we find the solution to be

$$T(n) = T(1) + \log_2 n = 1 + \log_2 n = O(\log n). \quad (2 \text{ pts})$$
4. [8 pts] Develop a RECURSIVE function that accepts an integer \( \text{num} \), and prints out in order the disk numbers that are moved for the optimal Towers of Hanoi solution with \( \text{num} \) disks total. For example, if \( \text{num} \) is 3, then the function should print the following sequence: 1213121

If \( \text{num} \) is 4, the function should print: 121312141213121

```c
void hanoi(int num)
{
    if (num > 0) {  // 2 pts
        hanoi(num-1);  // 2 pts
        printf("%d",num);  // 2 pts
        hanoi(num-1);  // 2 pts
    }
}
```
5. [12 pts] A circular linked list has a struct defined as follows:

```c
struct circLL {
    int data;
    struct circLL *next;
};
```

Write a function that deletes the first node in a circular linked list. In particular, your function should return a pointer to the front of the adjusted list. If the original list has no elements, then NULL should be returned. Make sure to free the memory for the deleted node.

```c
struct circLL* deleteFront(struct circLL* front) {
    if (front == NULL) return NULL; // 2 pts

    if (front->next == front) { // 1 pt
        free(front); // 1 pt
        return NULL; // 1 pt
    }

    struct circLL* last = front; // 1 pt
    while (last->next != front) // 2 pts
        last = last->next; // 1 pt

    last->next = front->next; // 1 pt
    free(front); // 1 pt
    return last->next; // 1 pt
}
```