Name: _____

Entry number: _____

There are 4 questions for a total of 15 points.

---

1. Answer the following:

   (a) (½ point) State true or false: Let $f(n) = 5n2^n + 3^n$ and $g(n) = n3^n$. Then $f(n) = O(g(n))$.

   (a) **True**

   (b) (½ point) State true or false: Let $f(n) = 5n2^n + 3^n$ and $g(n) = n3^n$. Then $g(n) = O(f(n))$.

   (b) **False**

   (c) (½ point) Express the running time of the algorithm below in big-O notation.

```
Alg2(A, n)
  - for i = 1 to n
      - for j = 2i to n
          - A[i] ← A[j] + 1
```

   (c) $O(n^2)$

   (d) (1 ½ points) Solve the following recurrence relation and write the exact value of $T(n)$. Show calculations.

$$T(n) = 2 \cdot T(n/2) + n^2; \quad T(1) = 1 \qquad \text{(assume } n \text{ is a power of 2)}$$

   (d) $T(n) = 2n^2 - n$

Solution:

$$\begin{aligned}
T(n) &= 2 \cdot T(n/2) + n^2 \\
&= 2 \cdot (2 \cdot T(n/2^2) + (n/2)^2) + n^2 \\
&= 2^2 \cdot T(n/2^2) + n^2 \cdot (1 + 1/2) \\
&\vdots \\
&= 2^i \cdot T(n/2^i) + n^2 \cdot (1 + 1/2 + ... + 1/2^{i-1}) \\
&\vdots \\
&= 2^{\log n} \cdot T(n/2^{\log n}) + n^2 \cdot (1 + 1/2 + ... + 1/2^{\log n - 1}) \\
&= n \cdot T(1) + n^2 \cdot \frac{1 - 1/n}{1 - 1/2} \\
&= n + 2n(n-1) \\
&= 2n^2 - n
\end{aligned}$$

2. (4 points) Solve the following recurrence relation and write the exact value of $T(n)$.

$$T(n) = \begin{cases} T(n-1) & \text{if } n > 1 \text{ and } n \text{ is odd} \\ 2 \cdot T(n/2) & \text{if } n > 1 \text{ and } n \text{ is even} \\ 1 & \text{if } n = 1 \end{cases}$$

2. $T(n) = 2^{\lfloor \log n \rfloor}$

(*Hint: Unrolling may be tricky. You may want to guess the bound and then use induction to confirm your guess.*)

Solution: The answer follows from the following claim.

Claim: For all $k \geq 0$, the following holds: For all $2^k \leq n < 2^{k+1}$, $T(n) = 2^k$.

Proof. We show this by induction on k. Let $P(k)$ denote the given proposition in the claim. We need to show that $\forall k, P(k)$ is true.

Base step: Base case is trivially true since $T(1) = 1$.

Inductive step: Suppose $P(1), P(2), ..., P(i)$ are true. We will show that $P(i+1)$ is true. Consider any $2^{i+1} \leq n < 2^{i+2}$. We need to consider the case when n is even and n is odd.

If n is odd, then $T(n) = T(n-1) = 2 \cdot T(\frac{n-1}{2})$. Note that $2^{i+1} \leq n - 1 < 2^{i+2}$. So, we have $2^i \leq (n-1)/2 < 2^{i+1}$. Applying induction hypothesis, we get $T(n) = 2^{i+1}$.

If n is even, then $T(n) = 2 \cdot T(n/2)$. Since $2^{i+1} \leq n < 2^{i+2}$, we have $2^i \leq n/2 < 2^{i+1}$. Applying induction hypothesis, we get that $T(n) = 2^{i+1}$. □

3. (4 points) In the lecture, we saw an implementation of Dynamic Arrays using regular arrays involving doubling the size of the array on overflows. What is the amortized running time for insert operations for the implementation where we increase the size of the array by 42 on overflows. That is, if the size of the array before overflow was m, then the size of the array after overflow is $m + 42$. Also, you may assume that we start with an empty array of size 1.

Solution: Starting from an empty array of size 1, consider n insertions. We can break down the running time into time for copying when an overflow occurs and time for inserting into the array. Note that the total time for insertion is going to be n since n elements are being inserted. Let us now focus on the time for copies.

Every time the current array overflows, we need to copy. The time for this copying is equal to the size of the array when the overflow occurs. The first time the array overflows, size of the array is 1. The second time the array overflows, size of the array is $(1 + 42)$. The third time the array overflows, size of the array is $(1 + 42 + 42)$. So, the i^{th} time the array overflows, the size of the array is $(1 + 42 \cdot (i - 1))$. Suppose k is the number of times the array overflows on n insertions. Then we have

$$1 + 42(k - 1) \leq n < 1 + 42k$$

This gives $k = \lfloor \frac{n-1}{42} \rfloor$. So, the total cost of copying is

$$\sum_{i=1}^{k} 1 + 42(i - 1) = k + 42 \cdot \frac{k(k-1)}{2} \leq \frac{n-1}{42} + \frac{(n-1) \cdot (n-43)}{2 \cdot 42} = O(n^2).$$

So, the asymptotic running time for insert operation is $O(n)$.

4. (4 points) Consider the following basic java implementation of linked list that we have used in the lecture. Complete the body of the method *SortList* below that is supposed to sort the integers in the linked list in non-decreasing order. It will be sufficient to give an $O(n^2)$-time algorithm for this problem. You are NOT allowed to use arrays within this method.

```
class Node{
   public int value;
   public Node next;
}
class LinkedList{
   public Node head;
   public int size;
   public LinkedList(){size=0; head = null;}
   //other methods
   ⋮
   //Implementation of Bubble Sort
   public void SortList(){
       Node N;
       for(int i=0;i<size-1;++i){
           N = head;
           for(int j=0;j<size - i - 1;++j){
               if(N.value > N.next.value){
                   int x = N.value;
                   N.value = N.next.value;
                   N.next.value = x;
               }
               N = N.next;
           }
       }
   }
}
```