Answer each question on separate pages; attach programs and output with the appropriate questions. Make sure that each question is
   • fully stated at the top of the document, and is
   • answered with full statements and easy to understand arguments.
HAND WRITTEN documents WILL NOT BE ACCEPTED. Place answers in numeric order, return this exam document stapled on top of the exam answers. Make sure that you write your name on this document and on all program headers. Place your name in the header or footer of all other pages of the document.
1. A contract job offer has a pay structure as follows:
   - First day of the contract pays $0.01
   - Second day pays $0.02
   - Third day pays $0.04
   - Fourth day pays $0.08
   - i.e., the amount doubles every day through day 31 then the algorithm starts at $0.01 again

   a) Write a recursive method that given the day (number) computes the amount of money paid for that day.

   ```java
   public static double totalpay(int n)
   {
       if (n > 31)
           return (n/31)*pay(31) + pay(n%31);
       else return pay(n);
   }
   
   public static double pay(int n)
   {
       if (n == 1) return 0.01;
       else return n * pay(n-1);
   }
   ```

   b) Assuming that the contract runs for 45 days, what is the total amount of money paid.

   ```java
   public class MT182_1
   {
       public static void main(String[] args)
       {
           System.out.println(totalpay(45));
       }

       public static double totalpay(int n)
       {
           double totalpay;
           if (n > 31)
               totalpay = ((n/31)*pay(31) + pay(n%31));
           else totalpay = pay(n);
           return totalpay;
       }

       public static double pay(int n)
       {
           if (n == 1) return 0.01;
           else return n * pay(n - 1);
       }
   }
   ```

   ANSWER
   8.2283865417792E+31
2. Greatest Common Divisor (GCD)
   The GCD of two numbers is the largest positive integer that divides both numbers without a remainder.

Write a specification for a method that computes the GCD of two integers. Include a statement of purpose, the preconditions, the postconditions, and a description of the parameters.

Prichard & Carrano     pages 84-93; 203-209

Statement of Purpose
   For any two given integers, find the largest integer that divides both while yielding a zero remainder.

Parameters
   Integers x and y

Precondition
   At least one integer cannot be zero.
   Since \((x == y)\) implies that \(\text{GCD}(x, y)\) is also equal to \(x \&/\text{or} \ y\), it is a common requirement that one of the integers, e.g., \(x\), be greater than the other integer, e.g., \(y\)
   i.e., \(x > y\)

   The restriction \(x > y > 0\) can be argued to have no significant effect on the universality of the GCD, i.e., except for the sign, the negative value of an integer has the same factors as the positive value of the same integer. If one tries to argue about the GCDs of negative values, e.g., -5, and a positive value, e.g., +5, the positive value will always win out in a contest regarding the GREATEST Common Divisor.

Postcondition
   The largest integer that divides both \(x \& y\) while leaving a zero remainder, has been located.

Description of the Procedure
   Create a sorted list of all the factors of \(x\). It will contain at least the number "1" and "x"

   Create a sorted list of all the factors of \(y\). It will contain at least the number "1" and "y"

   Determine the largest number which is common to both lists. That number is the GCD of \(x \& y\)

   http://en.wikipedia.org/wiki/Greatest_common_divisor
3. Design and Implement an ADT that represents a credit card. The ADT data should include the customer name, the account number, the next due date, the reward points, and the account balance. The initialization operation should set the data to client-supplied values. Include operations for a credit card charge, a cash advance, a payment, the addition of interest to the balance, and the display of the account statistics.

ANSWER TO BE SUPPLIED IN THE NEAR FUTURE

SEE TEXT FOR EXAMPLES
4. Ackermann’s Function

\[
Acker(m, n) = \begin{cases} 
  n + 1 & \text{if } m == 0 \\
  Acker(m - 1, 1) & \text{if } n == 0 \\
  Acker(m - 1, Acker(m, n - 1)) & \text{otherwise}
\end{cases}
\]

grows rapidly with respect to the size of \( m \) & \( n \)

a) Compute \( Acker(1, 2) \)

```java
public class MT182 {
    public static void main(String[] args) {
        System.out.println(Acker(1, 2));
    }

    public static int Acker(int m, int n) {
        if (m == 0) return n + 1;
        else if (n == 0) return Acker(m - 1, 1);
        else return Acker(m - 1, Acker(m, n - 1));
    }
}
```

b) Manually trace \( Acker(1, 2) \)

Hint: Expect many recursive calls even for \( Acker(1, 2) \)
5. Greatest Common Divisor

THEOREM
If \( a > b > 0 \) such that \( b \) is not a divisor of \( a \) then
\[
gcd(a, b) = gcd(b, a \mod b)
\]
if \( b \) divides \( a \) then
\[
b = gcd(a, b)
\]
hence
an appropriate choice for the base case is
\[
a \mod b = 0
\]

Recursive Definition
\[
gcd(a, b) = \begin{cases} 
  b & \text{if (} a \mod b = 0 \text{)} \\
  gcd(b, \mod b) & \text{otherwise}
\end{cases}
\]

Java Method
\[
public static int gcd( int a, int b)
{
  if ( a % b == 0 )
    return b; // base case
  else
    return gcd(b, a % b);
}
\]

ANSWERS

a) What happens if \( b > a \)?
\[
gcd(a, b) \Rightarrow gcd(b, a) \quad \text{from \"return gcd(b, a\%b\"")}
\]

b) How is the problem getting smaller? I.e.,
Does the method always approach the base case?
from \"return gcd(b, a\%b)\", i.e., \( a\%b < a \)

b) Why is the base case appropriate?
a\%b == 0 \Rightarrow b \text{ is a divisor of } a
continuation of the algorithm yields the gcd
6. Given the sorted linked list

```
| head | 21 | 27 | 33 | 38 | 38 |
```

Write a Java Method to insert the node `30` into the appropriate location in the list. Be sure to include methods to place the reference variables `curr` and `prev` in the appropriate locations to implement the insertion of the specified node.

**ANSWER**
```
Node create = new Node(30, null);
prev = head;
curr = prev.next;
while (curr.item < create.item)
{
    prev = prev.next;
curr = curr.next;
}
prev.next = create;
create.next = curr;
```

b. Given the sorted linked list

```
| head | 21 | 27 | 33 | 38 | 38 |
```

Write a Java Method to delete the node `33` from the list.

Be sure to include methods to place the reference variables `curr` and `prev` in the appropriate locations to implement the deletion of the specified node.

**ANSWER**
```
prev = head;
curr = prev.next;
while (curr.item < 33)
{
    prev = prev.next;
curr = curr.next;
}
curr = curr.next; morte prev.next;
prev.next = curr; morte.next = null;
morte = null;
```
7.

a. Given the circular link list

write a pseudo code method to delete the 1st node.

last = list;
while(last.next != list) last = last.next;
list = list.next;
last.next = list;

a. Given the circular link list

write a pseudo code method to delete the 1st node.

first = list.next;
list.next = first.next;
first = null;
8. For the ADT Sorted List Operations, write the preconditions and postconditions for each of the operations

a. `+createSortedList()`
   
   **Preconditions**
   none
   
   **Postconditions**
   An empty Sorted Link List is created

b. `+sortedIsEmpty():Boolean {query}`
   
   **Preconditions**
   none
   
   **Postconditions**
   If Sorted Link List is empty returns TRUE
   Else returns FALSE

c. `+sortedSize():integer {query}`
   
   **Preconditions**
   none
   
   **Postconditions**
   returns number of items in Sorted Link List

d. `+sortedAdd(in item:ListItemType)`
   
   **Preconditions**
   Item to be inserted has been identified
   
   **Postconditions**
   Item is inserted into the appropriate position
   Otherwise an OutOfBoundsException is thrown

e. `+sortedRemove(in item:ListItemType)`
   
   **Preconditions**
   Item to be deleted has been identified
   
   **Postconditions**
   If the identified item was located in the list
   then it was deleted
   else the method throws an exception
f. +sortedGet(in index:integer)

   Preconditions
   Index holds the value to be returned
   Postconditions
   If index is within the bounds of the array the item is returned
   otherwise an OutOfBoundsException is thrown

g. +locateIndex(in item:ListItemType):integer {query}

   Preconditions
   Item to be located is specified
   Postconditions
   If item is found the index is returned
   otherwise the index following where
   the item should be located is returned
9. The recursive method `getNumberEqual` searches an array `x` of `n` integers for occurrences of the integer `val`. It returns the number of integers in `x` that are equal to `val`. For example, if `x` contains the nine integers 2, 4, 7, 5, 6, 9, 7, 8, and 7 then `getNumberEqual(x, 9, 7)` returns the value 3 since 7 occurs three times in `x`.

```java
public static int getNumberEqual(int x[], int n, int val)
{
    int count;
    if (n <= 0) return 0;
    else if (x[n-1] == val) count = 1;
    else count = 0;
    return getNumberEqual(x, n-1, val) + count;
}
```

Using the Criteria for Constructing Recursive Solutions, listed below, demonstrate exactly how the method meets each of the criterions.

Criteria for Constructing Recursive Solutions
I. The problem must be capable of being restated as smaller problem of the same type.
   - each call to `getNumberEqual` reduces the index `n` by 1
     hence it ultimately must reach zero

II. A limited sequence of recursive calls, if not each recursive call, must diminish the size of the problem.
   - each call to `getNumberEqual` reduces the index `n` by 1
     hence it ultimately must reach zero

III. It must be possible to isolate an instance of the problem that can serve as a base case.
    - Since all indexes e.g., `n >= 0`
      when the index `n <= 0` is a reasonable base case

IV. It must always be possible to reach the base case within a finite number of recursive calls.
    - each call to `getNumberEqual` reduces the index `n` by 1
      hence it ultimately must reach zero