1. The Recursive operations for Factorial and Fibonacci sequence was discussed in class.

A) For factorial 6!a) Show recursive stack operations, provide details step-by-step, b**)** Walk through your stack operations and provide the result. c**)** Write Java code with input factorial 6! d**)** Compile and run your program, what is the running time of your algorithm?

Ans: A) Steps for factorial of 6:

* Main() will be pushed to stack
* Factorial 6 is pushed on the top of main()
* Factorial 5 is pushed
* Factorial 4 is pushed
* Factorial 3 is pushed
* Factorial 2 is pushed
* Factorial 1 is pushed
* So, top is now Factorial 1
* Now since we know value of Factorial 1 is 1, it’ll be popped out with a value 1
* Now the top is Factorial 2, so it’ll return 2 and value will be popped
* Top is Factorial 3, so 6 will be popped out
* Top is now Factorial 4 so, 24 will be popped out
* Top is now Factorial 5 so, 120 will be popped out
* Top is now at Factorial 6 so, 720 will be popped out
* At last main will execute the value and so, **720** will be returned as output.

Time Complexity of the recursive Factorial function is O(n).

B) For Fibonacci sequence with n=5, a) Show recursive stack operations, provide details step-by-step, b**)** Walk through your stack operations, provide the result. c) Provide Iterative algorithm for Fibonacci function, d**)** Write Java code for both recursive and iterative algorithms. e**)** Compile and Run your program.

Ans: B) Steps for Fibonacci with n = 5:

Algorithm for Fibonacci(Iterative):

* Begin first = 0 and next = 1 and sum = 0
* if(n == 0)

return 0

* Else if(n == 1 || n == 2)

return 1

* Else
* for(The loop will run till last number )

sum = first + next

first = next

next = sum

* End For
* Returns sum
* End else
* Main() will be called and fib(5) is pushed to stack
* fib(5) calls fib(4) so, fib(4) is pushed to stack
* fib(4) calls fib(3) so, fib(3) is pushed to stack
* fib(3) calls fib(2) so, fib(2) is pushed to stack
* fib(2) calls fib(1) so, fib(1) is pushed to stack
* fib(1) calls fib(0) so, fib(0) is pushed to stack
* Now, fib(0) is 0 so, it’s popped out from the stack
* fib(1) returns 1 and gets popped out from the stack
* fib(2) returns 1 and gets popped out of the stack
* fib(3) returns 2 and gets popped out of the stack
* fib(4) returns 3 and gets popped out of the stack
* fb(5) returns 5 and gets popped out of the stack
* It returns 5 after end of the recursion in main.

C) For Towers of Hanoi problem with n=5 discs, how does the algorithm work? What data structures would you use? provide step by step operations. Write Java code, compile and run program.

Ans: For Tower of Hanoi problem, **Stack** data structure is best suited.

**Algorithm**:

* If disc == 1, then

Move disc from source to destination.

* Else

tower(n-1, from, to, middle)

Move disc from source to destination

tower(n-1, middle, from, to)

* End If

Steps for the Tower of Hanoi for n = 5:

Move disc 1 from A to C

Move disc 2 from A to B

Move disc 1 from C to B

Move disc 3 from A to C

Move disc 1 from B to A

Move disc 2 from B to C

Move disc 1 from A to C

Move disc 4 from A to B

Move disc 1 from C to B

Move disc 2 from C to A

Move disc 1 from B to A

Move disc 3 from C to B

Move disc 1 from A to C

Move disc 2 from A to B

Move disc 1 from C to B

Move disc 5 from A to C

Move disc 1 from B to A

Move disc 2 from B to C

Move disc 1 from A to C

Move disc 3 from B to A

Move disc 1 from C to B

Move disc 2 from C to A

Move disc 1 from B to A

Move disc 4 from B to C

Move disc 1 from A to C

Move disc 2 from A to B

Move disc 1 from C to B

Move disc 3 from A to C

Move disc 1 from B to A

Move disc 2 from B to C

Move disc 1 from A to C

5. Write a recursive method to sumDigits that has one integer parameter and returns the sum of the digits in the integer specified. The method should throw IllegalArgumentException if the integer specified is negative. For example, if the integer is 26497, then this method should return 28. Remember, your method should not use iterative loops.

Ans: public class SumofDigits {

public static int sumDigits(int n){

if(n < 0){

throw new IllegalArgumentException();

}else if(n == 0){

return 0;

}

else{

return (n % 10 + sumDigits(n/10));

}

}

public static void main(String[] args){

int n = 26497;

System.out.println("The sum of digits of " + n + " is: " + sumDigits(n));

}

}

6. Write a recursive method countStringBinary that has one integer parameter n and returns the number of binary strings of length n that do not have two consecutive 0's. For example, for n = 4, the number of binary strings of length 4 that do not contain two consecutive 1's is 2: 1111, 1110, 1101, 1011, 1010, 0111, 0110, 0101

Ans: public class CountStringBinary {

public static int countStringBinary(int n){

if(n == 1){

return 2;

}if(n == 2){

return 3;

}

return countStringBinary(n-1) + countStringBinary(n-2);

}

public static void main(String[] args){

int n = 4;

if(n > 0){

System.out.println(countStringBinary(n));

}

}

}

7. An *n*-bit Gray code is a list of the 2*n* different *n*-bit binary numbers such that each entry in the list differs in precisely one bit from its predecessor. The *n* bit binary reflected Gray code is defined recursively. How does algorithm works for n=4, describe step-by-step. Write Java code, compile and run program.

Ans: For n = 4, the Gray code algorithm follows these steps:

* Generate (n-1) bit binary codes and prepend each binary code with 0.
* Use (n-1) bit binary codes in reverse order and prepend with 1.