**Exsercises - Java Recursion #1** Name\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Period\_\_\_\_\_\_\_\_\_

1)Show the output of each of the 3 methods when you call show(123):

public static void show(int n) public static void show(int n)

{ {

if(n != 0) System.out.println(n%10);

show(n / 10); if(n != 0)

System.out.println(n%10); show(n/10);

} }

public static void show(int n)

{

System.out.println(n%10);

if(n != 0)

show(n/10);

System.out.println(n%10);

}

2)Using the following method answer the questions:

public static int puzzle(int base, int limit)

{

if(base > limit)

return -1;

else

if (base == limit)

return 1;

else

return base \* puzzle(base+1, limit);

}

1. Identify the base/terminating case(s)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
2. Identify the recursive case(s) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
3. What is the output for System.out.println(puzzle(14,10)); \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
4. What is the output for System.out.println(puzzle(4,7)); \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
5. What is the output for System.out.println(puzzle(0,0)); \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3) Complete the code to recursively evaluate the sum=1/1 + ½ + 1/3 + …+1/n, n>=1.

public static double sum(int n) //pre:n>=1

{

if(\_\_\_\_\_\_\_\_\_\_) //terminating/base case - where the recursion stops

return \_\_\_\_\_\_\_\_;

return \_\_\_\_\_\_\_ + sum(\_\_\_\_\_\_\_\_);

}

Evaluate each method f for the given input parameters:

4) public static int f (int x) System.out.println( f(3));\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

{

if (x == 0) //very similar in structure to recursive factorial

return (x + 1);

return 1 + f(x – 1);

}

5) public static int f (int x,int y) System.out.println( f(6,5));\_\_\_\_\_\_\_\_\_\_\_\_\_\_

{

if (x <= 0)

return 0;

if (y >= x)

return 1 + f(y,x); //note that we are sending y’s value before x here

return 2 + f (x-3,y-1);

}

6) public static int f (int x) System.out.println(f (f (f (f (f (f (3)))))));\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

{ //do inner most parenthesis 1st and work your way out

if (x < 5)

return x \* x + 1;

if (x == 5)

return x \* x – 3;

return f(x-2);

}

7) public static void print (int n) print(3);\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

{

print (n-1); //there is something horribly wrong here

System.out.print(n);

print(n-1);

}

8) Assume we have an array list[] = {4,-1,5,1,8,3,-2,1,6,7}

public static int calc(int list[], int first, int last)

{

if (first > last) //first is used as the index of an element in the array

return 0;

return list[first] + calc(list, first+1, last);

}

Show the output for System.out.println(“The result is “ + calc(list,2,7));

**Exercises - Java Recursion #2** Name\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Period\_\_\_\_\_\_\_\_\_

1) public static int mystery (int num)

{

if (Math.abs(num) <= 9)

return 1;

return (mystery(num/10) + 1);

}

What would mystery(-349) return? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2) public static int answer (int n)

{

if (n==1)

return 2;

return (2 \* answer(n-1));

}

What would answer(5) return? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3) Ackerman’s function is defined as such:

n+1 when m==0

ack(m,n) = { ack(m-1, 1) when m!=0 and n==0

ack(m-1, ack(m, n-1)) when m!=0 and n!=0

Evaluate the following:

ack(0,5) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ack(1,0) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ack(1,1) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Write a java method that returns the result of Ackerman’s function given two integers m and n:

4) Write a recursive method called countUp, where the method displays a sequence of numbers counting up from 1 to the argument sent:

countUp(4) should output 1234

countUp(9) should output 123456789

5) Write a recursive method to raise a given power to an exponent. You may assume that the base is non-zero and the exponent is a non-negative integer. Extra credit if it also works for negative exponents with an extra recursive call.

6) Write a recursive method that will display a base 10 number as its binary equivalent.

baseTwo(5) should output 101

baseTwo(7) should output 111

baseTwo(8) should output 1000

baseTwo(10) should output 1010

**Exercises – Merge Sort Name\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Period**\_\_\_\_\_\_\_\_\_\_

The concept of the merge sort is to recursively break down an array into logical sub-arrays, sorting each individual sub-array and merge them back together into a single, sorted array. The bulk of the work is done with the merge process.

1. In order to better understand this method, write a similar method detailed below.

//pre: list1 and list2 are both non-empty and sorted from least to greatest

//post: return a new array that has all elements from list1 and list2 and is sorted

**public static int[]** merge (**int[]** list1, **int[]** list2)

{

**int** *size* = list1.length + list2.length;

**int[]** *ans* = **new int[***size***]**;

**return** *ans*;

}

Hint: have two index variables to represent what elements you are comparing in each of

the two lists. Use a while loop that goes while each index is < its corresponding array size. Compare the array values at their indexes, and copy the smaller of the two to the

ans array. Then, advance the index of the array that had its element copied. Once the

loop ends, you know one of the arrays ran out of elements. Copy any remaining elements from the other array to the ans array.

For the following arrays, show the resulting array after each step of the mergeSort process: 1 step for each split of a sub-array, and 1 step for each merging of sub-arrays. Sub-arrays are underlined.

Example: given 8 5 7 2 1 6 9 4,

8 5 7 2 1 6 9 4

8 5 7 2 1 6 9 4

8 5 7 2 1 6 9 4

5 8 2 7 1 6 4 9

2 5 7 8 1 4 6 9

1 2 4 5 6 7 8 9

1. 6 3 9 1 5 8 2 7
2. 8 7 6 5 4 3 2 1
3. 4 3 7 6 2 9 0 5

**Exercises BIG-OH 1** Name\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Find the Big-Oh efficiency of the following: Period\_\_\_\_\_\_\_\_

1) public static void loops1(int n)

{

for (int a=0; a<n; a++)

for (int b=0; b<n; b++)

for(int c=0; c<n; c++)

System.out.println(“\*\*” + n + “\*\*”);

}

2) public static void loops2(int n)

{

for (int a=0; a<n; a++)

for (int b=0; b<n; b++)

for(int c=0; c<5; c++)

System.out.println(“\*\*” + n + “\*\*”);

}

3) public static void loops3(int n)

{

for (int a=0; a<n; a++)

for (int b=0; b<7; b++)

for(int c=0; c<5; c++)

System.out.println(“\*\*” + n + “\*\*”);

}

4) public static void loops4(int n)

{

for (int a=0; a<2; a++)

for (int b=0; b<7; b++)

for(int c=0; c<5; c++)

System.out.println(“\*\*” + n + “\*\*”);

}

5) public static void loops5(int n)

{

for (int a=0; a<n; a++)

System.out.println(“\*\*” + n + “\*\*”);

for (int b=0; b<n; b++)

System.out.println(“$$” + n + “$$”);

for(int c=0; c<n; c++)

System.out.println(“~~” + n + “~~”);

}

6) public static void loops6(int n)

{

for (int a=0; a<n; a++)

System.out.println(“\*\*” + n + “\*\*”);

for (int b=0; b<n; b++)

for(int c=0; c<n; c++)

System.out.println(“$$” + n + “$$”);

}

7) public static void loops7(int n)

{

for (int a=0; a<n; a++)

for (int b=0; b<a; b++)

for(int c=0; c<5; c++)

System.out.println(“\*\*” + n + “\*\*”);

}

8) public static void loops8(int n)

{

for(int c=0; c<5; c++)

{

int i=0;

while(i<c)

{

System.out.println(“\*\*” + n + “\*\*”);

i++;

}

}

}

9) public static void loops9(int []list)

{

for (int a=0; a<10; a++)

for (int b=0; b< list.length; b++)

System.out.println(list[b]\*a);

}

10) public static void loops10(int [][]list)

{

for (int a=0; a<list.length; a++)

for (int b=0; b<list[0].length; b++)

for(int c=0; c<20; c++)

System.out.println(list[a][b] \* c);

}

**Exercises BIG-OH 2** Name\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Find the Big-Oh efficiency of the following: Period\_\_\_\_\_\_\_\_

1. int size=list.length;

…

for(int j=0; j<size; j++)

System.out.print(list[j] + “ “);

1. char[][] board=new char[size][size];

…

for(int r=0; r<size; r++)

{

for(int c=0; c<size; c++)

System.out.print(board[r][c]);

System.out.println();

}

1. F(n) = 3n3 + 200n2 + 1024n (consider which part grows the fastest)
2. if (list.length > 0)

{

for(int j=0; j<list.length; j++)

System.out.print(list[j] + “ “);

System.out.print(“There are “ + list.length + “ people in the list. “);

}

1. F(n) = (n2 + 2n + 1) / (n + 1) //be careful
2. F(n) = 3n2 + 200n\*log2n2 + 50n\*log2n + 342n4 + 300n
3. int ran = (int)(Math.random()\*100);

System.out.println(ran + “ is your lucky number.”);

1. String[] names = new String[size]; //be careful

…

for(int r = 0; r<10; r++)

for(int c = 0; c< size; c++)

System.out.println(names[s]);

1. System.out.println(“Hello World”);
2. F(n) = 1,000,000n + n1,000,000 + 2n
3. public static int recurA(int n)

{

if (n == 1)

return 1;

return n + recurA(n – 1);

}

1. public static int recurB(int n)

{

if (n == 1)

return 1;

return n + recurB(n / 2);

}

1. public static void doStuff(int n)

{

for(int i=1; i<=n; i++)

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

} //as defined above

1. public static void repeatMore(int n)

{

for(int i=n; i>0; i--)

doStuff(n); //as defined above

}

1. public static int recurC(int n)

{

if (n == 1)

return 1;

return n + recurC(n – 1) + recurC(n - 2);

}

1. public static int sumRecurC(int n)

{ int sum=0;

for(int i=100; i <= n+100; i++)

sum += recurC(n);

return sum;

}