COP 3502 Study Group Sheet: Recursion Solutions

1) The code below returns the nth Harmonic number. (Note: n must be positive.) Rewrite the function recursively.

```
double harmonic(int n) {
    double res = 0;
    for (int i=1; i<=n; i++)
        res += 1.0/i;
    return resl
}
```

Rewrite this method ***recursively***:

```
double harmonic(int n);
```

Solution

```
double harmonic(int n) {
    if (n == 1) return 1.0;
    return 1.0/n + harmonic(n-1);
}
```

2) Write a *recursive* function returns the sum of the digits of its input parameter n. You may assume that n is non-negative. For example, productDigits(274) should return 56, since 2*7*4 = 56.

```
int productDigits(int n);
```

Solution

```
int productDigits(int n) {
    if (n < 10) return n;
    return (n%10)*productDigits(n/10);
}
```

3) Without running the function below, determine the output of the function call doit(4):

```
void doit(int n) {
    if (n>0) {
        doit(n-1);
        printf("%d ", n);
        doit(n-1);
    }
}
```

What is this function similar to, in structure?

Solution

1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

This is similar in structure to the Towers of Hanoi code. The numbers printed here represent the disk number being moved each time.

4) The function below is an attempt at a recursive binary search of a sorted array. Why is this function no faster than a basic linear search through the array?

```
int search(int numbers[], int low, int high, int value) {

    if (low > high) return 0; // Not found.

    int mid = (low+high)/2;

    if (numbers[mid] == value) return 1; // Found.

    return search(numbers, low, mid-1) ||
           search(numbers, mid+1, high);
}
```

Solution

If the number is NOT in the array, then the return statement will run both branches of the recursion. (Note that if the number is in the left of the array, when the first search call returns true, the second call never occurs due to short circuiting.) In running both branches of the recursion, it's possible that every index of the array is examined, when searching for the number.

5) Imagine being a particle starting at the coordinates (x1, y1) in the Cartesian plane, moving to (x2, y2), where x1 ≤ x2 and y1≤ y2, and at each step you could either add 1 to your x coordinate or add 1 to your y coordinate. Write a recursive function to calculate the number of different ways to make the journey. (No need to code a base case for when it's not possible, ie when x1 > x2 or y1 > y2.)

```
int numWays(int x1, int y1, int x2, int y2);
```

Solution

```
int numWays(int x1, int y1, int x2, int y2) {
    if (x1 == x2 && y1 == y2) return 1;

    int res = 0;
    if (x1 < x2) res += numWays(x1+1, y1, x2, y2);
    if (y1 < y2) res += numWays(x1, y1+1, x2, y2);
    return res;
}
```