CSE 15
Introduction to Data Structures
Midterm 1 Review Problems

1. Recall the recursive function `C(n, k)` in the program `BinomialCofficients.c` discussed in lecture and posted on the webpage. Write a box trace of the function call `C(5, 3)`. Use this trace to find the value of `C(5, 3)`. Notice that in the full recursion tree for `C(5, 3)`, the value `C(3, 2)` is evaluated 2 times, and `C(2, 1)` is evaluated 3 times. Suggest a modification to the function that would allow it to avoid computing the same values multiple times. Carry out your suggestion by writing the C code.

2. Write a recursive function in C called `sum(n)` that computes the sum of the integers from 1 to n. Hint: emulate the factorial function discussed in lecture (replace multiplication by addition). Modify your answer to recursively compute the sum of the integers from n to m, where $n \leq m$ (return 0 if $n > m$).

3. Write recursive functions in C that determine the sum of the elements in an int array. Do this in 3 ways.
   a. Write a recursive function `sumArray1(int A[], int n, int k)` that returns the sum of the leftmost k elements of array `A[]`, of length n. Get the k^{th} element from the left, then compute the sum of the leftmost $(k-1)$ elements recursively. Return the sum of the two. (This function doesn't actually need the length n.)
   b. Write a recursive function `sumArray2(int A[], int n, int k)` that returns the sum of the rightmost k elements of array `A[]`, of length n. Get the k^{th} element from the right, then compute the sum of the rightmost $(k-1)$ elements recursively. Return the sum of the two. (This function does need the length n.)
   c. Write a recursive function `sumArray3(int A[], int p, int r)` that returns the sum of the subarray `A[p..r]`. Hint: use `MergeSort()` as a model.

4. Write a modification of the recursive C functon `BinarySearch()` that prints out the sequence of array elements that are compared to the target.

5. What output does the following C program produce?

```
#include<stdlib.h>
#include<stdio.h>

int getValue(int a, int b, int n){
   int x, c;
   printf("arrive: a = %d, b = %d\n", a, b);
   c = (a+b)/2;
   if( c*c <= n ){
      x = c;
   }else{
      x = getValue(a, c-1, n);
   }
   printf("depart: a = %d, b = %d\n", a, b);
   return x;
}

int main(){
   printf("%d\n", getValue(3, 13, 5));
   return EXIT_SUCCESS;
}
```

6. Perform a box trace of the following recursive function for the input $n = 100$, and determine the output. What does the function do?

```
#include<stdio.h>

void doSomething(int n){
   if(n>=8){
      doSomething(n/8);
   }
   printf("%d", n%8);
}
```

7. Use what you learned in problem 6 above to create a recursive function called `printInteger()` that prints a string representation of the integer n expressed in base b. For instance, the function call `printInteger(100,8)` would return the String "144", which was printed in problem 6. Your function need not allow bases greater than 10 (although that would be a good exercise), but it should deal correctly with the input $n = 0$ (i.e. return the digit "0" in that case).

```
   void printInteger(int n, int b){
      // your code starts here











      // your code ends here
   }
```

8. Consider the following C program.

```
#include<stdio.h>
#include<stdlib.h>

int main(void){
   int i, j;
   double x = 4.2, y;
   double * A = calloc(4, sizeof(double));
   double B[] = {1.2, 5.3, 2.1, 3.4};
   double *p, *q;

   p = malloc(sizeof(double));
   y = x+2;
   q = &y;
   *p = *q + 2.5;

   for(i=0; i<4; i++){
      j = 3-i;
      *(A+i) = B[j] + i;
   }
   printf("%f, %f, %f, %f\n", *A, *B, *p, *q);
   A = B;
   printf("%f, %f, %f, %f\n", *A, *(A+1), *(A+2), *(A+3) );
   return(EXIT_SUCCESS);
}
```

a. Write the output of this program exactly as it would appear on the screen:

b. List the pointer variables in this program, and for each one, state whether it points to stack memory or heap memory. If at some point in the program the pointer changes from stack to heap or heap to stack, note the point in the program where that happens.

c. Does this program contain any memory leaks? If so, what alteration(s) would be needed to eliminate those leaks?

9. Write a C function called `search()` with the prototype below that takes as input a null (`'\0'`) terminated `char` array S (i.e. a string) and a single `char c`, and returns the leftmost index in S at which the target `c` appears, or returns -1 if no such index exists.

```
int search(char* S, char c){
  // your code goes here




}
```

10. Write a C function called `diff()` with the prototype below that takes as input two null (`'\0'`) terminated char arrays (i.e. strings) A and B and returns the *difference* string consisting of all chars in A that are not in B. The returned chars should be stored in a null terminated char array in the same order they appeared in A, possibly with repetitions. The returned array should be allocated from heap memory to be the same length as A (although the null terminator `'\0'` might appear before the end of this array.) You may use the `strlen()` function found in the library `string.h` to determine this length. You may also assume the existence of a C function called `search()` as described in problem 6.

```
char* diff(char* A, char* B){
  // your code goes here




}
```