ECE264 Fall 2013
Exam 1, September 24, 2013

In signing this statement, I hereby certify that the work on this exam is my own and that I have not copied the work of any other student while completing it. I understand that, if I fail to honor this agreement, I will receive a score of ZERO for this exam and will be subject to possible disciplinary action.

Signature:

You must sign here. Otherwise you will receive a **2-point** *penalty.*

Read the questions carefully.
Some questions have conditions and restrictions.

This is an *open-book, open-note* exam. You may use any book, notes, or program printouts. No personal electronic device is allowed. You may not borrow books from other students.

One learning objective (recursion) is tested in this exam. To pass an objective, you must receive 50% or more points in this exam.

Contents

Learning Objective 1 (Recursion) Pass Fail

Total Score:

1 Recursive Formula (5 points)

For a given positive integer, we want to partition it into the sum of some positive integers, or itself. For example, 1 to 4 can be partitioned as

```
1 = 1          2 = 1 + 1          3 = 1 + 1 + 1          4 = 1 + 1 + 1 + 1
                 = 2                = 1 + 2                = 1 + 1 + 2
                                    = 2 + 1                = 1 + 2 + 1
                                    = 3                    = 1 + 3
                                                           = 2 + 1 + 1
                                                           = 2 + 2
                                                           = 3 + 1
                                                           = 4
```

- One way to partition 1.
- Two ways to partition 2.
- Four ways to partition 3.
- Eight ways to partition 4.

In general, there are 2^{n-1} ways to partition value n. You can use this fact and do not have to prove it.

In the above examples,
- when partitioning 1, "3" is not used.
- when partitioning 2, "3" is not used.
- when partitioning 3, "3" is used once.
- when partitioning 4, "3" is used twice.

Question: When partitioning value n ($n > 4$), how many times is "3" used?
You $\boxed{\textbf{must}}$ explain your answer. An answer without explanation will receive no point.

Additional information is on the next page.
This question asks you to write a mathematical formula, not a C program.

The answer is **not** $n - 2$.
You can use the following table to validate your formula. If your formula does not match these cases, the formula is definitely wrong. However, matching these cases does not mean your formula is correct. You must have a systematic way to find the formula. Do not try to find a formula to match these values.

n	4	5	6	7	8	9
f(n)	2	5	12	28	64	144

2 Incorrect Recursive Binary Search (5 points)

Consider the following incorrect implementation of recursive binary search.

```
int binarysearch(int * arr, int key, int low, int high)
{
  if (low >= high) /* ERROR: should be >, not >= */
    {
      return -1;
    }
  int mid = (low + high) / 2;
  if (arr[mid] == key)
    {
      return mid;
    }
  if (arr[mid] > key)
    {
      return binarysearch(arr, key, low, mid - 1);
    }
  return binarysearch(arr, key, mid + 1, high);
}
```

The main function is shown below.

```
#include <stdio.h>
#include <stdlib.h>
int binarysearch(int * arr, int key, int low, int high);
#define ARRAYSIZE 10
int main(int argc, char * * argv)
{
  int arr[ARRAYSIZE] = {1, 12, 23, 44, 65, 76, 77, 98, 109, 110};
  int ind;
  for (ind = 0; ind < ARRAYSIZE; ind ++)
    {
      printf("%d\n", binarysearch(arr, arr[ind], 0, ARRAYSIZE));
    }
  return EXIT_SUCCESS;
}
```

One line in `binarysearch` is incorrect, as marked by a comment. What is the output of this program? Please put your answers in the following table.
Hint: You will **not** receive the **full** score if your answers are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

3 Trace Recursive Function (5 points)

Consider the following program.

```
#include <stdio.h>
#include <stdlib.h>
int f(int n, int * count)
{
  (* count) ++; /* how many times f is called? */
  if ((n == 0) || (n == 1))
    {
      return 1;
    }
  int val = 0;
  int a = f(n - 1, count);
  int b = f(n / 2, count);
  val += a + b;
  return val;
}

int main(int argc, char * * argv)
{
  int count = 0;
  int val = 4;
  int fv = f(val, & count);
  printf("f(%d) = %d, count = %d\n", val, fv, count);
  return EXIT_SUCCESS;
}
```

What are the values of `fv` and `count` printed by this program? $\boxed{\textbf{Explain}}$ how you get the answers. An answer without explanation will receive no point.

You need to write **two** answers.

fv	
count	

You can write an arithmetic expression without calculating the result. For example, if the answer is 17, you can write 1 + 5 + 11.
Hint: **Drawing** the calling relationships will help you determine how many times the function is called and how the values change.

4 Reverse Array (5 points)

The following is a recursive function that reverses the elements in an array. Please fill in the necessary code.

```
#include <stdio.h>
#include <stdlib.h>
/*
 * The function takes two pointers of integers.
 * Each pointer stores an address; an integer is stored at the address.
 * The function swaps the values stored at the two addresses.
 */
void swap(                       ) /* fill the arguments (0.5 point) */
{
  /* fill the code (1 point) */





}
/* ------------------------------------ */
void reversehelper(int * arr, int low, int high)
{

  if (              ) /* fill the terminating condition (1 point) */

    {
      /* do not call recursive function again */
      return;
    }

  /* swap the first and the last elements */

  swap(                 ); /* fill the arguments (1 point) */


  /* call the function itself to reverse the remaining elements of the
     array (1 point) */




}
/* ------------------------------------- */
void reverse(int * arr, int len)
{
  /* call the helper function */
  /* fill the arguments (0.5 point) */

  reversehelper(                    );

}

/* ------------------------------------- */
/* Do not change anything below this line */
void printArray(int * arr, int len)
{
  int ind;
  for (ind = 0; ind < len; ind ++)
    {
      printf("%d ", arr[ind]);
    }
  printf("\n");
}

#define ARRAYSIZE  16

int main(int argc, char * * argv)
{
  int arr[ARRAYSIZE];
  int iter;
  for (iter = 0; iter < ARRAYSIZE; iter ++)
    {
      /* initialize the array */
      int ind;
      for (ind = 0; ind < ARRAYSIZE; ind ++)
        {
          arr[ind] = ind;
        }
      reverse(arr, iter);
      printArray(arr, iter);
    }
  return EXIT_SUCCESS;
}
```