Homework #1

Chapter 2 Recursion & Chapter 10 Algorithm Efficiency

Due Date: Sunday, Sep. 26, 2021 (No Late Submissions will be Accepted.)

Grades depend on neatness and clarity. Write your answers with enough detail about your approach and concepts used, so that the grader will be able to understand it easily.

Submissions should be one file containing all answers.

Chapter 2 Recursion

1. (10 points each for each exercise question) Textbook p 89 - 92, Exercises #1, 2, 6, 10, 11, 12, 13, 14, 15, and 16.

Chapter 10 Algorithm Efficiency

2. Textbook p 311, Question 1
3. Textbook p 311, Question 2
4. Textbook p 317, Question 3
5. Textbook p 320, Question 4
6. Textbook p 321 - 322, Exercises #4, 5, and 6

Chapter 2 Recursion

SUMMARY

1. Recursion is a technique that solves a problem by solving a smaller problem of the same type.
2. When constructing a recursive solution, keep the following four questions in mind:
   a. How can you define the problem in terms of a smaller problem of the same type?
   b. How does each recursive call diminish the size of the problem?
   c. What instance of the problem can serve as the base case?
   d. As the problem size diminishes, will you reach this base case?
3. When constructing a recursive solution, you should assume that a recursive call's result is correct if its precondition has been met.
4. You can use the box trace to trace the actions of a recursive function. These boxes resemble activation records, which many compilers use to implement recursion. Although the box trace is useful, it cannot replace an intuitive understanding of recursion.
5. Recursion allows you to solve problems—such as the Towers of Hanoi—whose iterative solutions are difficult to conceptualize. Even the most complex problems often have straightforward recursive solutions. Such solutions can be easier to understand, describe, and implement than iterative solutions.
6. Some recursive solutions are much less efficient than a corresponding iterative solution due to their inherently inefficient algorithms and the overhead of function calls. In such cases, the iterative solution can be preferable. You can use the recursive solution, however, to derive the iterative solution.
7. If you can easily, clearly, and efficiently solve a problem by using iteration, you should do so.

EXERCISES

1. The following recursive function `getNumberEqual` searches the array `x` of `n` integers for occurrences of the integer `desiredValue`. It returns the number of integers in `x` that are equal to `desiredValue`. For example, if `x` contains the ten integers 1, 2, 4, 4, 5, 6, 7, 8, 9, and 12, then `getNumberEqual(x, 10, 4)` returns the value 2, because 4 occurs twice in `x`.

```
int getNumberEqual(const int x[], int n, int desiredValue)
{
   int count = 0;

   if (n <= 0)
      return 0;
   else
   {
      if (x[n - 1] == desiredValue)
         count = 1;

      return getNumberEqual(x, n - 1, desiredValue) + count;
   } // end else
} // end getNumberEqual
```

   Demonstrate that this function is recursive by listing the criteria of a recursive solution and stating how the function meets each criterion.

2. Perform a box trace of the following calls to recursive functions that appear in this chapter. Clearly indicate each subsequent recursive call.
   a. `rabbit(5)`
   b. `countDown(5)` (You wrote `countDown` in Checkpoint Question 3.)

CPSC 250

3. Write a recursive function that will compute the sum of the first n integers in an array of at least n integers. *Hint:* Begin with the nth integer.

4. Given two integers, *start* and *end*, where *end* is greater than *start*, write a recursive C++ function that returns the sum of the integers from *start* through *end*, inclusive.

5. a. Revise the function `writeBackward`, discussed in Section 2.3.1, so that its base case is a string of length 1.
   b. Write a C++ function that implements the pseudocode function `writeBackward2`, as given in Section 2.3.1.

6. Describe the problem with the following recursive function:

```
void printNum(int n)
{
   std::cout << n << std::endl;
   printNum(n - 1);
} // end printNum
```

7. Given an integer $n > 0$, write a recursive C++ function that writes the integers $1, 2, \ldots, n$.

8. Given an integer $n > 0$, write a recursive C++ function that returns the sum of the squares of 1 through n.

9. Write a recursive C++ function that writes the digits of a positive decimal integer in reverse order.

10. a. Write a recursive C++ function `writeLine` that writes a character repeatedly to form a line of n characters. For example, `writeLine('*', 5)` produces the line *****.
    b. Now write a recursive function `writeBlock` that uses `writeLine` to write m lines of n characters each. For example, `writeBlock('*', 5, 3)` produces the output

```
*****
*****
*****
```

11. What output does the following program produce?

```
int getValue(int a, int b, int n);
int main()
{
   std::cout << getValue(1, 7, 7) << std::endl;
   return 0;
} // end main

int getValue(int a, int b, int n)
{
   int returnValue = 0;
   std::cout << "Enter: a = " << a << " b = " << b << std::endl;
   int c = (a + b)/2;
   if (c * c <= n)
      returnValue = c;
   else
      returnValue = getValue(a, c-1, n);

   std::cout << "Leave: a = " << a << " b = " << b << std::endl;
   return returnValue;
} // end getValue
```

12. What output does the following program produce?

```
int search(int first, int last, int n);
int mystery(int n);
int main()
{
```

CPSC 250


```
            return f(n-2) * f(n-4);
        } // end switch
    } // end f
```

Show the exact output of the program. What argument values, if any, could you pass to the function f to cause the program to run forever?

15. Consider the following function:

```
void recurse(int x, int y)
{
   if (y > 0)
   {
      x++;
      y--;
      std::cout << x  << " " << y << std::endl;
      recurse(x, y);
      std::cout << x << " " << y << std::endl;
   } // end if
} // end recurse
```

Execute the function with $x = 5$ and $y = 3$. How is the output affected if x is a reference argument instead of a value argument?

16. Perform a box trace of the recursive function `binarySearch`, which appears in Section 2.4.2, with the array 1, 5, 9, 12, 15, 21, 29, 31 for each of the following search values:

- **a.** 5
- **b.** 13
- **c.** 16

17. The algorithm `kSmall`, as discussed in Section 2.4.4, assumes that the array contains distinct values. Modify the algorithm so that the array can contain duplicate values. You will need to decide whether the second smallest value in the array 7 7 8 9, for example, is 7 or 8. Although, your choice is arbitrary, it will affect your algorithm.

18. Imagine that you have 101 Dalmatians; no two Dalmatians have the same number of spots. Suppose that you create an array of 101 integers: The first integer is the number of spots on the first Dalmatian, the second integer is the number of spots on the second Dalmatian, and so on. Your friend wants to know whether you have a Dalmatian with 99 spots. Thus, you need to determine whether the array contains the integer 99.

- **a.** If you plan to use a binary search to look for the 99, what, if anything, would you do to the array before searching it?
- **b.** What is the index of the integer in the array that a binary search would examine first?
- **c.** If none of your Dalmatians have 99 spots, exactly how many comparisons will a binary search require to determine that 99 is not in the array?

19. This problem considers several ways to compute x^n for some $n \geq 0$.

- **a.** Write an iterative function `power1` to compute x^n for $n \geq 0$.
- **b.** Write a recursive function `power2` to compute x^n by using the following recursive formulation:

$x^0 = 1$
$x^n = x \times x^{n-1}$ if $n > 0$

CPSC 250


```
   std::cout << "mystery(30) produces the following output: \n";
   result = mystery(30);
   std::cout << "mystery(30) = " << result << "; should be 5\n";
   return 0;
} // end main

int search(int first, int last, int n)
{
   int returnValue = 0;
   std::cout << "Enter: first = " << first << " last = "
             << last << std::endl;

   int mid = (first + last)/2;
   if ( (mid * mid <= n) && (n < (mid+1) * (mid+1)) )
      returnValue = mid;
   else if (mid * mid > n)
      returnValue = search(first, mid-1, n);
   else
      returnValue = search(mid+1, last, n);
   std::cout << "Leave: first = "
             << first << " last = " << last << std::endl;
   return returnValue;
} // end search

int mystery(int n)
{
   return search(1, n, n);
} // end mystery
```

13. Consider the following function that converts a positive decimal number to base 8 and displays the result

```
void displayOctal(int n)
{
   if (n > 0)
   {
      if (n / 8 > 0)
         displayOctal(n / 8);
      std::cout << n % 8;
   } // end if
} // end displayOctal
```

Describe how the algorithm works. Trace the function with $n = 100$.

14. Consider the following program:

```
int f(int n);

int main()
{
   std::cout << "The value of f(8) is " << f(8) << std::endl;
   return 0;
} // end main

/** @pre  n >= 0. */
int f(int n)
{
   std::cout << "Function entered with n = " << n << std::endl;
   switch (n)
   {
      case 0: case 1: case 2:
         return n + 1;
      default:
```

Chapter 10 Algorithm Efficiency

Question 1 How many comparisons of array items do the following loops contain?

```
for (j = 1; j <= n - 1; j++)
{
   i = j + 1;
   do
   {
      if (theArray[i] < theArray[j])
         swap(theArray[i], theArray[j]);
      i++;
   } while (i <= n);
}
```

Question 2 Repeat Question 1, replacing the statement `i = j + 1` with `i = j`.

Question 3 What order is an algorithm that has as a growth-rate function

a. $8 \times n^3 - 9 \times n$ **b.** $7 \times \log_2 n + 20$ **c.** $7 \times \log_2 n + n$

Question 4 Consider a sequential search of n data items.

a. If the data items are sorted into ascending order, how can you determine that your desired item is not in the data collection without always making n comparisons?

b. What is the order of the sequential search algorithm when the desired item is not in the data collection? Do this for both sorted and unsorted data, and consider the best, average, and worst cases.

c. Show that if the sequential search algorithm finds the desired item in the data collection, the algorithm's order does not depend upon whether or not the data items are sorted.

4. Suppose that your implementation of a particular algorithm appears in C++ as

```
for (int pass = 1; pass <= n; pass++)
{
   for (int index = 0; index < n; index++)
   {
      for (int count = 1; count < 10; count++)
      {
         . . .
      } // end for
   } // end for
} // end for
```

The previous code shows only the repetition in the algorithm, not the computations that occur within the loops. These computations, however, are independent of n. What is the Big O of the algorithm? Justify your answer.

5. Consider the following C++ function `f`, which calls the function `swap`. Assume that `swap` exists and simply swaps the contents of its two arguments. Do not be concerned with `f`'s purpose. How many comparisons does `f` perform?

```
void f(int theArray[], int n)
{
   for (int j = 0; j < n; j++)
   {
      int i = 0;
      while (i <= j)
      {
         if (theArray[i] < theArray[j])
            swap(theArray[i], theArray[j]);
         i++;
      } // end while
   } // end for
} // end f
```


6. For large arrays, is a sequential search faster than a binary search in the worst case? Explain.
7. Show that any polynomial $f(x) = c_n x^n + c_{n-1} x^{n-1} + \cdots + c_1 x + c_0$ is $O(x^n)$.
8. Show that for all constants $a, b > 1$, $f(n)$ is $O(\log_a n)$ if and only if $f(n)$ is $O(\log_b n)$. Thus, you can omit the base when you write $O(\log n)$. *Hint:* Use the identity $\log_a n = \log_b n / \log_b a$ for all constants $a, b > 1$.