CPSC 1160 SECTION 001 INSTRUCTOR: GLADYS MONAGAN

ASSIGNMENT #4: SORTING

DUE DATE WITH BRIGHTSPACE: OCTOBER 12 AT 11:50 PM

THE LEARNING OUTCOMES

- to follow an algorithm from a description and to implement it in C++ (a sorting algorithm)
- to do a theoretical analysis of some code and to express it in big O notation
- to do a theoretical analysis of code written by the student and to express it in big O notation
- to write a recurrence relation with initial value if the function being analyzed is recursive
- to analyze code for best, average, worst cases

READINGS

- class notes
- §7.10
- chapter 17 on Recursion
  - not covered §17.3, §17.6, §17.7
- chapter 18 on Developing Efficient Algorithms
  - §18.1 - §18.4
  - not covered: Towers of Hanoi analysis §18.4.3
  - not covered: §18.5, (§18.6, §18.7), §18.8, §18.9, §18.10
  - yes Chapter Summary
- chapter 19 on Sorting §19.1 - §19.5
  - not covered §19.6, §19.7, §19.8

THEORETICAL ANALYSIS ON PAPER

Do a theoretical analysis of the following C++ function `mMultiply`.

Count the number of **multiplications** done in `mMultiply` (and its helper functions) as a function of n.

Show your workings.

Express the answer in big O notation.

```
int I(int i, int j, int n)
{
   return n * i + j;
}

int dotProduct(const int A[], const int B[], int i, int j, int n)
{
   int t = 0;
   for (int k = 0; k < n; k++) {
      t += A[ I(i, k, n) ] * B[ I(k, j, n) ];
   }
   return t;
}

void mMultiply( const int A[], const int B[], int C[], int n )
{
   for (int i = 0; i < n; i++ ) {
      for (int j = 0; j < n; j++ ) {
         C[I(i,j,n)] = dotProduct(A, B, i, j, n );
      }
   }
}
```

DESCRIPTION OF A SORTING ALGORITHM

Suppose we want to sort n integers of an array A in **ascending** order using the algorithm described below.

For the **following explanation** given below, assume that the number of elements is n > 5 and that there are no duplicates in the array. There is an array of integers A with n elements.

- We first compare A[0] with A[1]. If A[0] > A[1] then A[0] and A[1] are swapped.
- We next compare A[1] with A[2]. If A[1] > A[2] then A[1] and A[2] are swapped.
- We continue comparing in pairs and swapping as necessary until all n elements of the array are compared and possibly swapped. The largest element of the array is now at A[n-1].
- We repeat the whole process but this time in the "opposite direction" ignoring A[n-1]: we compare A[n-2] with A[n-3]. If A[n-3] > A[n-2] then A[n-2] and A[n-3] are swapped.
- We next compare A[n-3] with A[n-4]. If A[n-4] > A[n-3] then A[n-4] and A[n-3] are swapped.
- We continue comparing in pairs like that down the array until all n -1 elements have been compared. The smallest element in the array is then in A[0].
- Then the next pass is back in the "opposite direction" ignoring A[0] (we think of it as going up this time) comparing A[1] with A[2]. If A[1] > A[2] then A[1] and A[2] are swapped.
- We continue comparing and swapping in pairs as necessary all the way until A[n-3] is compared (and possibly swapped) with A[n-2]. At this point A[n-2] has the second largest element.
- Now on the way down we compare A[n-3] with A[n-4] and swap if necessary and so on.
- We continue until the whole array is sorted in ascending order.

For your assignment and in general, do **not** assume that n > 5.

For your assignment and in general, do **not** assume that there are no duplicates. Duplicates are allowed.

I just made those two simplifications to explain the algorithm above.

IMPLEMENT

1. Implement the algorithm described in the previous section and use the function prototype

```
void sort(int A[], int n);
```

   Test for yourself on a small array of integers. You may overload the function `sort` but you must still use the function prototype above and call the overloaded function in sort(A, n).

   Do not keep track of the swaps, i.e. you do not need a Boolean variable that would help you optimize whenever any elements were not swapped in a pass.

2. Now using the rand() library function, generate an array of 17 random integers in the range

   0 <= random number <= 999

   Print out the array before you `sort` it and after you `sort` it. Print 5 integers per line (use a named constant for 5) with the last line having at most 5 numbers.
   Format your output so that you can tell quickly whether the sorting is working.

3. For `sort`
   a. What is the worst case and the best case and the average case?
   b. Is the analysis different for the different cases? Why or why not?

4. Repeat 1. and 2. but use this time a **recursive function** that implements the sorting algorithm. Use the function prototype

```
void sortR(int A[], int n);
```

   Yes, you may overload the function.
   Like in the class notes for selection sort and insertion sort, recursive functions *can* have loops

   Given that `n` is the size of the array (i.e. `n` is the number of elements in the array), do a theoretical analysis for the running time of the algorithm: In your **theoretical** analysis, count the number of comparisons `sortR` does to sort `n` values. Count comparisons between array elements the way we have done in class.
   (Do not modify your code to do this theoretical analysis. This assignment is different from the lab.)
   Show your working and answer the questions for `sortR`.
   a. What is the worst case and the best case and the average case?
   b. Is the analysis different for the different cases? Why or why not?

SUBMIT WITH BRIGHTSPACE AS A SINGLE ZIP FILE: PART A) TO PART F) AS LISTED BELOW

A) The source code of the complete sorting programs: one program with an iterative version `sort` function and one program with a recursive version `sortR` function (you may put both versions in a single program if you want). Document clearly your functions.

B) Some "screen captures" or "print screens" that show that your iterative version `sort` works:
   1. Show the input array of 17 random integers (at most 5 integers per line)
   2. Show the sorted output array of 17 random integers (at most 5 integers per line)

C) Some "screen captures" or "print screens" that show that your recursive version `sortR` works:
   1. Show the input array of 17 random integers (at most 5 integers per line
   2. Show the sorted output array of 17 random integers (at most 5 integers per line)

Submit as a handwritten scanned document or as a MS Word or pdf file:

D) The theoretical analysis of the recursive version `sortR` and the answers to the questions 4a. 4b.
   -> Give a recurrence relation with initial value.
   -> Solve the recurrence relation with initial value and give your answer in closed-form as a function of n.
   -> Write the resulting expression in big O notation
   Show your work.
   Give the answers to the questions 4a. and 4b.

E) Give the complexity of `sort` in big O notation (no need to show your work).
   Give the answer to the questions 3a. and 3b.

F) The theoretical analysis of mMultiply

Do not plagiarize.