CS-203 – MIDTERM EXAM SOLUTION – 2022 SUMMER

GIUSEPPE TURINI – KETTERING UNIVERSITY

Instructions

- This exam is take-home, open-book, open-notes, and individual (no collaboration).
- Each part indicates the points awarded if correctly answered (partial credit available).
- Submit your solution as a single PDF file, via email using your Kettering account.
- The submission deadline is: Sunday 31 July 2022, before the end of the day.

Student Information

- Student full name (readable) and signature:

Exercise 1 (50 points)

Consider the following *iterative* (*i.e.*, *non-recursive*) algorithm:

```
void mysteryAlgorithm1(int[] A) {
   int n = A.length;
   boolean swapped = false;
   do {
      swapped = false;
      for(int i = 0; i < n-1; i++) {
         if(A[i] > A[i+1]) {
            int swap = A[i];
            A[i] = A[i+1];
            A[i+1] = swap;
            swapped = true;
         }
      }
      for(int i = n-2; i >= 0; i--) {
         if(A[i] > A[i+1]) {
            int swap = A[i];
            A[i] = A[i+1];
            A[i+1] = swap;
            swapped = true;
         }
      }
   } while(swapped);
}
```

Analyze this algorithm (*i.e.*, mysteryAlgorithm1), and:

a Determine what this algorithm computes, briefly explaining its strategy (10 points).

b Determine input size and basic operation, briefly explaining your choices (10 points).

c Express the basic operation count as a summation (10 points).

d Convert the basic operation count summation into a closed-form expression (10 points).

e Find the efficiency class of the basic operation count, using proper notation (10 points).

Note If necessary, perform this analysis for both the best-case and worst-case.

Exercise 2 (50 points)

Consider the following *recursive* algorithm:

```
int mysteryAlgorithm2(int n) {
   if(n == 0) { return 1; }
   else {
      int tmpRes = mysteryAlgorithm2(n-1);
      int res = n;
      for(int i = 1; i < tmpRes; i++) {
         res += n;
      }
      return res;
   }
}
```

Analyze this algorithm (*i.e.*, mysteryAlgorithm2), and:

a Determine what this algorithm computes, briefly explaining its strategy (10 points).

b Determine input size and basic operation, briefly explaining your choices (10 points).

c Write the recursive definition of the algorithm computation (5 points).

d Write the recursive definition of basic operation count (5 points).

e Convert the basic operation count into a closed-form expression (10 points).

f Find the efficiency class of the basic operation count, using proper notation (10 points).

Note If necessary, perform this analysis for both the best-case and worst-case.

Solution Exercise 1 (50 points)

Note The following is a comprehensive solution, including details, alternative answers, and extra comments that were not needed to score the maximum points.

a Determine what this algorithm computes, briefly explaining its strategy (10 points).

Given an input array A of integers, the algorithm (*"cocktail sort"*) sorts the input array values (increasingly) in-place (i.e., without using a significant amount of extra memory).

The sorting strategy is based on a sequence of *"double swapping scans"*: a forward begin-to-end swapping-scan, followed by a backward end-to-begin swapping scan, performed multiple times. Each individual *"swapping scan"* iteratively swaps 2 adjacent array items if they are not arranged in increasing order. So, at the end of each swapping scan at least 1 array element is placed in the correct sorted position.

After a *"double swapping scan"* is performed, if no swap was done, the algorithm stops, and the sorting of the input array is complete; otherwise, if a swap was done, the algorithm performs another *"double swapping scan"* etc.

b Determine input size and basic operation, briefly explaining your choices (10 points).

The input size is the item-size of the input array A (referred as n from now on).

This factor (n) clearly controls the amount of work done by the algorithm; however, the content of the input array affects the running time too. In fact, given a fixed array size: an array already sorted will require the minimum amount of work (only 1 *"double swapping scan"*), whereas an unsorted array will require more work.

For this reason, analyzing best-case and worst-case scenarios will be necessary.

To approximate the amount of work done by this algorithm, we have to evaluate the 4 main stages performed:

1. Initialization of local variables (lines 2-3).
2. *"do-while"* iteration (outer loop, lines 4-22).
3. *"for"* loop 1 (inner loop, forward begin-to-end swapping-scan, lines 6-13).
4. *"for"* loop 2 (inner loop, backward end-to-begin swapping scan, lines 14-21).

The basic operation for stage 1 is the assignment. The basic operation for stage 2 is the check of the local variable *"swapped"*. The basic operation for both stage 3 and 4 is the comparison of 2 array items.

Please consider that, selecting the "swap" as the basic operation to evaluate the work done by the "swapping scans" is incorrect, because it is not executed all the time (and it could lead to an erroneous estimate of the algorithm performance).

c Express the basic operation count as a summation (10 points).

d Convert the basic operation count summation into a closed-form expression (10 points).

e Find the efficiency class of the basic operation count, using proper notation (10 points).

The best-case scenario is when the algorithm performs the minimum amount of work possible: when the input array is already sorted, no swap performed, and only 1 *"double swapping scan"* is executed.

$$C_{best}(n) = 2 + \left(\sum_{i=0}^{n-2} 1\right) + \left(\sum_{i=n-2}^{0} 1\right) + 1 = 2 + (n-1) + (n-1) + 1 = 2n + 1 \in \Theta(n)$$

The worst-case scenario is when this algorithm performs the maximum amount of work possible: when each *"swapping scan"* only places 1 item in its correct sorted position, so a total of ⌊n/2⌋ iterations of the "do-while" are necessary, plus 1 additional run to check that the sorting is completed (no swaps), for a total of ⌊n/2⌋+1 times.

$$C_{worst}(n) = 2 + \sum_{j=1}^{\lfloor n/2 \rfloor + 1} \left[\left(\sum_{i=0}^{n-2} 1\right) + \left(\sum_{i=n-2}^{0} 1\right) + 1\right] = 2 + \sum_{j=1}^{\lfloor n/2 \rfloor + 1} (2n-1) =$$

$$= 2 + (\lfloor n/2 \rfloor + 1)(2n-1) \in \Theta(n^2)$$

So, in the general case scenario, the count of the basic operation can be classified in the following classes in terms of its order of growth:

$$C(n) \in \Omega(n)$$

$$C(n) \in O(n^2)$$

Solution Exercise 2 (50 points)

Note The following is a comprehensive solution, including details, alternative answers, and extra comments that were not needed to score the maximum points.

a Determine what this algorithm computes, briefly explaining its strategy (10 points).

Given an input integer value n, the algorithm computes n! (n factorial), recursively (using a decrease-by-1 design pattern) by using only additions (plus operator).

b Determine input size and basic operation, briefly explaining your choices (10 points).

The input size is the magnitude of the input integer value (referred as n from now on).
This factor (n) controls the amount of work done by the algorithm, and it is the only factor (no best-case and worst-case analyses are necessary).

To approximate the amount of work done by this algorithm, we have to evaluate the 3 main stages performed:

1. Initial checks (lines 2-3).
2. Recursive call (line 4).
3. Iterative computation (lines 5-9).

The basic operation for stage 1 is the equality check. The basic operation for stage 2 is the recursive function call. The basic operation for stage 3 is the addition.
Please consider that, in stage 1 and stage 3 we can neglect constant-time operations as the initialization of local variables and return statements to output results.

c Write the recursive definition of the algorithm computation (5 points).

$$F(n) = \begin{cases} 1 & if\ n = 1. \\ n + \sum_{i=1}^{F(n-1)-1} n & if\ n > 1. \end{cases} \quad \rightarrow \quad F(n) = \begin{cases} 1 & if\ n = 0. \\ F(n-1) * n & if\ n > 0. \end{cases}$$

d Write the recursive definition of basic operation count (5 points).

$$C(n) = \begin{cases} 1 & if\ n = 0. \\ 1 + C(n-1) + \left(\sum_{i=1}^{(n-1)!-1} 1 \right) & if\ n > 0. \end{cases} \quad \rightarrow \quad C(n) = \begin{cases} 0 & if\ n = 0. \\ C(n-1) + (n-1)! & if\ n > 0. \end{cases}$$

e Convert the basic operation count into a closed-form expression (10 points).

$$C(n) = C(n-1) + (n-1)! = C(n-2) + (n-2)! + (n-1)! = \cdots$$

$$pattern: \quad C(n-i) + \sum_{k=1}^{i} (n-k)! \quad 0 \leq i \leq n$$

$$solving\ pattern\ for\ last\ value\ of\ i: \quad i = n \quad C(n-n) + \sum_{k=1}^{n} (n-k)!$$

$$C(n) = C(0) + \sum_{k=1}^{n} (n-k)! = \cdots$$

$$check\ math\ solution\ of\ the\ sum\ of\ factorials$$

f Find the efficiency class of the basic operation count, using proper notation (10 points).

$$C(n) \in \Omega(n!)$$

$$C(n) \in O(\ ?\ )$$