Adam Hayen

Midterm Exam - Adam Hayen

1 (a) - Algorithm strategy

The purpose of this algorithm is to sort the integer input array A from smallest to largest. The result of sorting is stored back into array A. Elements are swapped in the array until it is sorted. This sorting algorithm is a variation of bubble sort called "Cocktail sort".

1 (b) - input size and basic operation

The algorithm accepts an array of int as its input, so it has an input size of n, with n being the size of the input array. Arrays of the same size but with different content may require a different amount of work to be sorted, so we need to analyze the best and worst case.

The most significant operations to estimate the work done by the algorithm are: The do-while loop and the increments of the two nested for loops. These are the most significant because they increase with the size of the input array.

1 (c) - Basic operation count

Worst case: $c(n) = (\sum_{i=0}^{n-1} (\sum_{j=0}^{n-2} (1) + \sum_{j=0}^{n-2} (1)))$

Best case: $c(n) = (\sum_{i=0}^{n-2} 2)$

1 (d) - Closed form expression

Worst case:

$$(\sum_{i=0}^{n-1} (n-1) + (n-1)) = (\sum_{i=0}^{n-1} (2n-2)) = n(2n-2) = 2n^2 - 2n$$

Best case:

$$c(n) = (\sum_{i=0}^{n-2} 2) = 2n - 2$$

1 (e) - Efficiency class of the algorithm

Worst case: $c(n) = 2n^2 \in O(n^2)$

Best case: $c(n) = 2n - 2 \in \Omega(n)$

2 (a) - Algorithm strategy

The purpose of this algorithm is to compute the factorial of the input, n. It recursively calculates the result using addition. The result is returned as an integer.

2 (b) - Input size and basic operation

The algorithm accepts an integer as an input, so it has an input size of n, with n being the magnitude of the input. The amount of work being done depends on n.

The basic operations are the assignment and increments to the variable “res”. This operation happens several times inside the for loop, and again in each recursive iteration.

2 (C) - Algorithm computation

$$mystery2(n) = \begin{cases} \sum_{i=1}^{mystery2(n-1)} n, & n > 0 \\ 1, & n = 0 \end{cases}$$

2 (d) - basic operation count

$$c(n) = \begin{cases} 0, & n = 0 \\ c(n-1) + (n-1)!, & n > 1 \end{cases}$$

2 (e) - closed form expression

Apply recurrence :

$c(n) = c(n - 1) + (n - 1)!$

$= [c(n - 2) + (n - 2)!] + (n - 1)!$

$= [[c(n - 3) + (n - 3)!] + (n - 2)!] + (n - 1)!$

Create pattern:

$$c(n) = c(n - i) + (\sum_{k=1}^{i} (n - k)!),\ with\ i \in [1, n]$$

Solve pattern:

$$c(n) = c(n - n) + (\sum_{k=1}^{n} (n - k)!) = (\sum_{k=1}^{n} (n - k)!)$$

2 (f) - Efficiency class

Average case: $c(n) = (\sum_{k=1}^{n} (n - k)!) \in \Theta((n - 1)!)$

It is not necessary to analyze the best and worst case, because the same input magnitude will do the same amount of work.