Algorithms and Data Structures: Homework #3

Due on February 27th, 2023, 23:00

Problem 3.1: Asymptotic Analysis

Considering the following pairs of functions f and g, show for each pair whether or not it belongs to each of the relations $f \in \Theta(g)$, $f \in O(g)$, $f \in o(g)$, $f \in \Omega(g)$, $f \in \omega(g)$, $g \in \Theta(f)$, $g \in O(f)$, $g \in o(f)$, $g \in \Omega(f)$, or $g \in \omega(f)$.

(a) $f(n) = 9n$ and $g(n) = 5n^3$

$\Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{9n}{5n^3} = 0$

$\Rightarrow \lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{5n^3}{9n} = \infty$

$\Rightarrow f \notin \Theta(g),\ f \in O(g), f \in o(g), f \notin \Omega(g), f \notin \omega(g), g \notin \Theta(f), g \notin O(f),\ g \notin o(f),\ g \in \Omega(f),$ and $g \in \omega(f)$

(b) $f(n) = 9n^{0.8} + 2n^{0.3} + 14\log(n)$ and $g(n) = \sqrt{n}$

$\Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{9n^{0.8} + 2n^{0.3} + 14\log(n)}{\sqrt{n}} = \infty$

$\Rightarrow \lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{\sqrt{n}}{9n^{0.8} + 2n^{0.3} + 14\log(n)} = 0$

$\Rightarrow f \notin \Theta(g),\ f \notin O(g),\ f \notin o(g), f \in \Omega(g), f \in \omega(g), g \notin \Theta(f), g \in O(f), g \in o(f), g \notin \Omega(f),$ and $g \notin \omega(f)$

(c) $f(n) = \frac{n^2}{\log(n)}$ and $g(n) = n\log(n)$

$\Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\frac{n^2}{\log(n)}}{n\log(n)} = \infty$

$\Rightarrow \lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{n\log(n)}{\frac{n^2}{\log(n)}} = 0$

$\Rightarrow f \notin \Theta(g),\ f \notin O(g), f \notin o(g),\ f \in \Omega(g), f \in \omega(g), g \notin \Theta(f), g \in O(f), g \in o(f), g \notin \Omega(f),$ and $g \notin \omega(f)$

(d) $f(n) = \log(3n)^3$ and $g(n) = 9\log(n)$

$\Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\log(3n)^3}{9\log(n)} = \infty$

$\Rightarrow \lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{9\log(n)}{\log(3n)^3} = 0$

$\Rightarrow f \notin \Theta(g),\ f \notin O(g),\ f \notin o(g), f \in \Omega(g), f \in \omega(g), g \notin \Theta(f), g \in O(f), g \in o(f), g \notin \Omega(f),$ and $g \notin \omega(f)$

Problem 3.2: Selection Sort

Insertion Sort was discussed during the lecture. Selection Sort is similar to Insertion Sort and works as follows. Given an array of elements, you always take the current element and exchange it with the smallest element that can be found on the right hand side of the current element. In doing so, you will gradually build up a sorted sequence on the left side (like in Insertion Sort), and in each iteration "attach" the smallest element from the remaining unsorted right side to it.

(a) Implement Selection Sort:

Algorithm: Selection Sort[A, n]

```
for i = 1 to n − 1 do
        min = i
        for j = i + 1 to n do
                if A[j] < A[min] then
                        min = j
                end if
        end for
        if min != i then
                aux = A[min]
                A[min] = A[i]
                A[i] = aux
        end if
end for
```

```
void SelectionSort(std::vector<int>& A, int n){
for (int i = 0; i < n - 1; i++){
        int aux, min = i;
             for (int j = i + 1; j < n; j++)
             if ( A[j] < A[min] )
                      min = j;
             if (min != i) {
                  aux = A [min];
                  A[min] = A[i];
                  A[i] = aux;
             }
         }
     }
```

Implementation of Selection Sort in C++

(b) Show that Selection Sort is correct (Hint: consider the loop invariant).

To prove the correctness of our algorithm, the loop invariant chosen must hold during initialization, iteration and termination of the loop. There are 2 for loops in the implementation of selection sort:

Outer Loop Invariant, which for each iteration of the for loop, the subarray $A[1 \ldots i-1]$ contains the $i-1$ smallest elements of A in increasing order; and

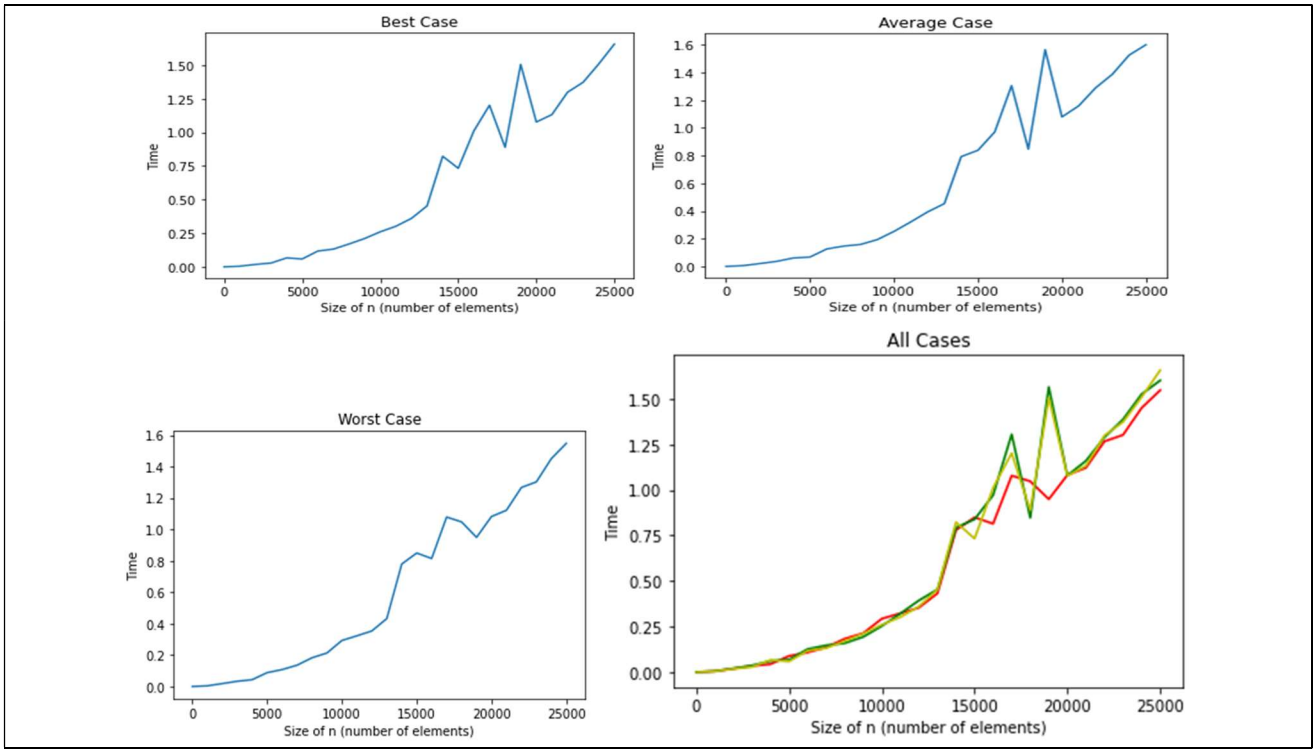
Inner Loop Invariant, which for each iteration of the for loop, min holds the position of the smallest value of the elements of the subarray $A[i \ldots j-1]$.

(c) Generate random input sequences of length n as well as sequences of length n that represent Case A and Case B for the Selection Sort algorithm. Case A: the case which involves the most swaps (Hint: it is not a decreasingly ordered array). Case B: the case with the least swaps. Briefly describe how you generated the sequences (e.g., with a random sequence generator using your chosen language).

Generating random sequences as input to the algorithm has been done using rand() function, part of cstdlib library, which produces a pseudo-random integral number.

Case A: Case with the most amount of swaps happens on certain types of occasions. The highest number of swaps is $n-1$, one swap for each iteration of the outer for loop. This would happenonly in cases when all the numbers in the sequence are unique. In case that the numbers are produced randomly, meaning there is a chance for duplicates, the number of swaps would be $n-k$, where k is the number of duplicates of the biggest number in the sequence (if sorting increasingly).
Case B: Case with the least amount of swaps theoretically happens when the input sequence has already been sorted. This case has been generated by choosing a random sequence of numbers using rand() function and sorting them increasingly using sort() function, part of the algorithm library.

(d) Run the algorithm on the sequences from (c) with length n for increasing values of n and measure the computation times. Plot curves that show the computation time of the algorithm in Case A, Case B, and average case for an increasing input length n. Note that in order to compute reliable measurements for the average case, you have to run the algorithm multiple times for each entry inyour plot. You can use a plotting tool/software of your choice:



(e) Interpret the plots from (d) with respect to asymptotic behavior and constants.

The asymptotic behavior is determined by the number of comparisons made inside the inner loop. Algorithm has to go through the loop $n-1$ times, which is linear, and for each of them, it has to gothrough $n-i$ comparisons in order to find the smallest number in the remaining subarray, $A[i+1 \ldots n]$.

The growth rate of the algorithm is quadratic and this is reflected in the graph, visible from its parabolicshape. All 3 lines have small differences between them, because assignment of *min* and number of swap processes take constant time and linear time respectively. Therefore as $n \to \infty$, time taken to sort will grow quadratically and still the differences will be small, as n^2 will dominate n of the swaps.