Algorithms and Data Structures

Arnav Singh

February 24, 2020

3 Sheet 3

3.1 Asymptotic Analysis

3.1.1 $f(x) = 9x$ and $g(x) = 5x^3$

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \frac{9x}{5x^3} = \frac{9}{5x^2} = \frac{9}{\infty} = 0 \tag{1}$$

Since the limit is 0, it implies that $f \in o(g)$ and therefore also to $f \in O(g)$

$$\lim_{x\to\infty} \frac{g(x)}{f(x)} = \frac{5x^3}{9x} = \frac{5x^2}{9} = \frac{\infty}{9} = \infty \tag{2}$$

Since the limit is ∞, it implies that $g \in \omega(f)$ and therefore also to $g \in \Omega(f)$.

3.1.2 $f(x) = 9x^{0.8} + 2x^{0.3} + 14log(x)$ and $g(x) = x^{0.5}$

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \frac{9x^{0.8} + 2x^{0.3} + 14log(x)}{x^{0.5}} = \frac{\infty}{1} = \infty \tag{3}$$

The numerator is much bigger than $n^{0.5}$ and thus reaches infinity. Hence $f \in \omega(g)$ and therefore also $f \in \Omega(g)$

$$\lim_{x\to\infty} \frac{g(x)}{f(x)} = \frac{x^{0.5}}{9x^{0.8} + 2x^{0.3} + 14log(x)} = \frac{1}{\infty} = 0 \tag{4}$$

The denominator is much bigger than $n^{0.5}$ and thus reaches infinity. Hence $g \in o(f)$ and therefore also $g \in O(f)$

3.1.3 $f(x) = \frac{x^2}{logx}$ and $g(x) = xlogx$

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \frac{x^2}{x(logx)^2} = \frac{\infty}{1} = \infty \tag{5}$$

The numerator x is much bigger than log x and thus reaches infinity faster than log x. Hence $f \in \omega(g)$ and therefore also $f \in \Omega(g)$

$$\lim_{x\to\infty} \frac{g(x)}{f(x)} = \frac{x(logx)^2}{x^2} = \frac{1}{\infty} = 0 \tag{6}$$

Similarly, the denominator x is much bigger than log x and thus reaches infinity faster. Hence $g \in o(f)$ and therefore also $g \in O(f)$

3.1.4 $f(x) = (log(3x))^3$ and $g(x) = 9logx$

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \frac{(log(3x))^3}{9logx} = \frac{\infty}{1} = \infty \tag{7}$$

The numerator is $(log(3x))^3$ is to the whole cube i.e much bigger than $9logx$ and thus reaches infinity. Hence $f \in \omega(g)$ and therefore also $f \in \Omega(g)$

$$\lim_{x\to\infty} \frac{g(x)}{f(x)} = \frac{9logx}{(log(3x))^3} = \frac{1}{\infty} = 0 \tag{8}$$

The denominator is $(log(3x))^3$ is to the whole cube i.e much bigger than $9logx$ and thus reaches infinity. Hence $g \in o(f)$ and therefore also $g \in O(f)$

3.2 Selection Sort

3.2.1 Implemented Selection sort check cpp files

3.2.2 Show that Selection Sort is correct

Before we prove the correctness of the algorithm let's consider the loop invariant; For the outermost loop, after every iteration arr[0...i-1] consists of i-1 elements in sorted order, all other elements from arr[i..n-1] are larger than or equal to elements in arr[0..i-1].

Initialization:At the first iteration,when i = 0, the sublist to the left of i is empty. Therefore we can say that an empty sublist is ordered and hence satisfies the loop invariant.
Maintenance: The idea is at the start of any iteration, no element from 0 to i-1 can ever be changed again because the inner for loop will find the smallest element from arr[i + 1 . . . n-1] and will swap it for the arr[i] element. This element is swapped into the ith position. So now the new arr[i] element cannot be smaller than any element in arr[0 . . . i-1] because the loop invariant is true prior to the start of the iteration, so it will be the smallest element from i to the right and bigger than any element to its left. Hence, the loop invariant is preserved.
Termination: When the last iteration of the algorithm terminates, the loop counter will be on n-2. If the i th element is smaller, it will be exchanged with the n-1 st element. Since we know that the sublist to the left of i is already sorted and neither arr[n-2] nor arr[n-1] can be smaller than any item in arr[0 . . . n-2] because of the loop invariant, the combined list will be in sorted order. Otherwise, if the ith element is larger, the list is already sorted.

3.2.3 Briefly describe how you generated the sequences

I generated random sequences using the srand function.

Case A is the case with the most swaps this is when the highest number is in the begging of the array.For eg, 5,1,2,3,4 etc. I implemented this by making arr[0]=size which means the last element becomes the first.

For Case B, the case with the least swaps is when the array is already sorted, i.e 0 swaps are performed. I created a random array, then I sorted it using the sort algorithm from the algorithm library, so now my array is sorted and then I perform the selection sort and measure the time using the chrono library.

I outputed the values from the compiler into the output.txt file and then used gnuplot to read data from the output.txt file and plot the graph.

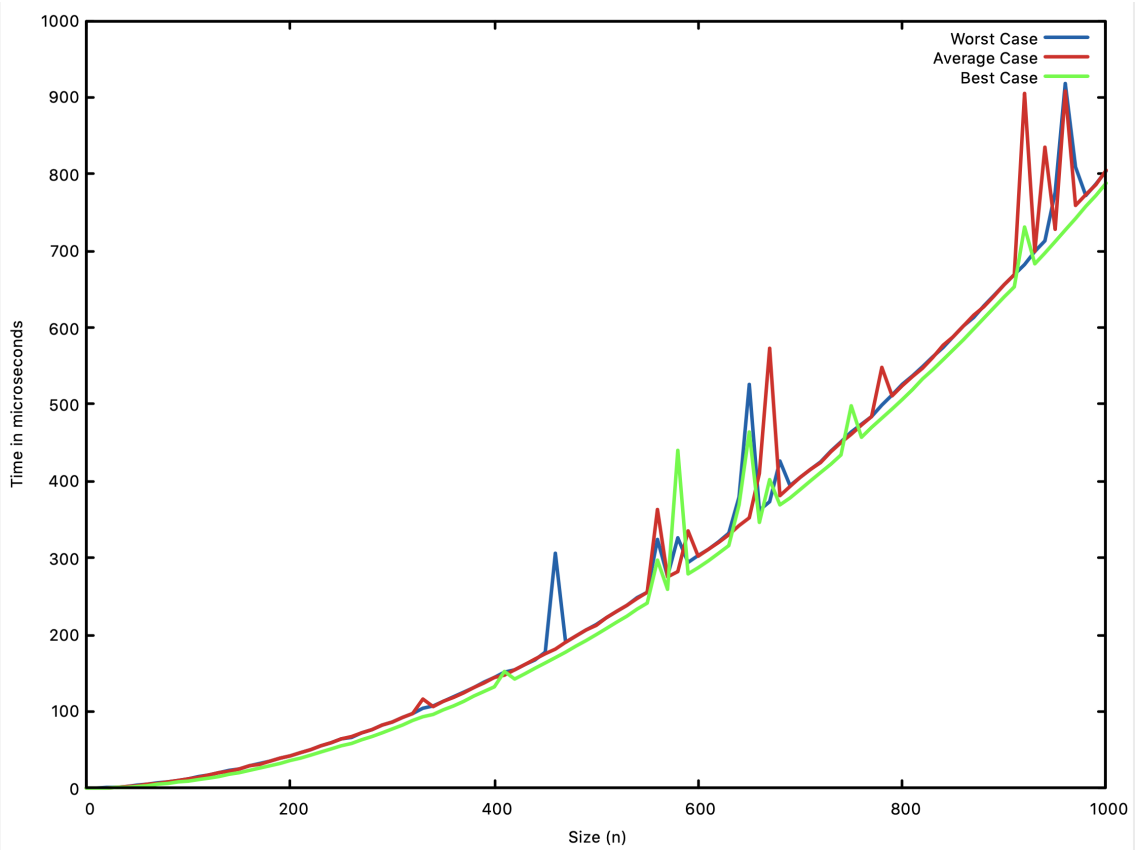

Figure 1: Plot of the computation time(microseconds) of the three cases with respect to increasing input size (n).

3.2.4 Graph of the computation time for the worst, best and average case for size up-to 1000 elements.

3.2.5 Interpretation of the Graph

From the graph, it is clearly evident that all the three cases have the same time complexity. The number of comparisons will always be:

$$\sum_{i=0}^{\infty}(n-1) = \frac{n(n+1)}{2} \tag{9}$$

If we take the highets order term this implies that we have a time complexity of $\Theta(n^2)$

The small fluctuations of the graph are due to the swaps that need to take place and is perhaps also based on the optimization of the compiler.However, we can take these to be outlier's from the general trend of the curve.