Homework 1 : Algorithms and Data Structures

Digdarshan Kunwar

February 2019

Problem 1.1

Considering the following pairs of functions f and g, show for each pair whether $f \in \Theta(g),\ f \in O(g),\ f \in o(g),\ f \in \Omega(g),\ f \in \omega(g),\ g \in \theta(f),\ g \in O(f),\ g \in o(f),\ g \in \Omega(f)$, or $g \in \omega(f)$.

(a) $f(n) = 3n$ *and* $g(n) = n^3$

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

From the above it implies that :

$f \in O(g),\ g \in \Omega(f), f \in o(g),\ g \in \omega(f)$

(b) $f(n) = 7n^{0.7} + 2n^{0.2} + 13\log n$ and $g(n) = \sqrt{n}$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{7n^{0.7} + 2n^{0.2} + 13\log n}{\sqrt{n}} = \infty$$
$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{\sqrt{n}}{7n^{0.7} + 2n^{0.2} + 13\log n} = 0$$

From the above it implies that :

$g \in O(f),\ f \in \Omega(g), g \in o(f),\ f \in \omega(g)$

(c) $f(n) = n^2/\log n$ and $g(n) = n\log n$

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

From the above it implies that :

$f \in \Omega(g), g \in O(f),\ f \in \omega(g), g \in o(f)$

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

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

From the above it implies that :

$f\in\Omega(g), g\in O(f),\ f\in\omega(g), g\in o(f)$

Problem 1.2

Selection Sort

!! NOTE !!
Run file named selection-sort form the zip file.
The code to compile is selection-sort.cpp.
The graph is generated via GNU Plot with the data from data.txt in plot.pdf.

(a) Implement Selection Sort.

```
void selectionSort(int arr[], int n)
{
    int i, j, min_index,tmp;
    for (i = 0; i < n-1; i++)
    {
        // Finding  the minimum element in unsorted array
        min_index = i;
        for (j = i+1; j < n; j++)
          if (arr[j] < arr[min_index])
            min_index = j;

        // Swap the found minimum element with the element in index i of the array
        tmp=arr[min_index];
        arr[min_index]=arr[i];
        arr[i]=tmp;
    }
}
```

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

Here considering the the program.
How the program works is that it searches for the the minimum element in the unsorted part of the array and swaps it with the current element and as the loop continues to it's next iteration the element gets included in the sorted part of the array.

Loop Invariant in the Outer Loop : Here at the start of every iteration of the outer loop of the program the sub Array A[0..i-1] consists of the smallest elements of the main array A[0..n-1] which are sorted in order.

Initialization
Initially the sorted part of the array has no elements in it.When the i=1 then there is only one element.And one element is always sorted in the array size of one.
Maintenance
During maintenance the left part of the sub-array A[0..i-1] is always maintained to be sorted.So in each iteration an element from the unsorted part of the array A[i+1..n-1] or

the element with the min_index gets added to the sorted part of the array as the largest element within the sorted part of the array.
Termination
As the loop invariant maintains,when the loop terminates the unsorted part of the array has no elements.Thus the the whole array is sorted. Therefore the selection sort is able to sort the array.

(c) Generate random input sequences of length n as well as sequences of length n that represent the worst case and the best case for the Selection Sort algorithm.

Here the program generating random sequenced number is within the file RandomNumber.h. It has a class RandomNumber where there are static methods to generate best random and the worst cases for the particular n values.
RandomNumber::bestCase(n) firstly generates the random sequenced array with n elements then the sorts array to for the bestCase evaluation.
RandomNumber::averageCase(n) generates the random sequenced array with n elements.
RandomNumber::worstCase(n) generates the reversed-sorted array of the bestCase with n elements for later evaluation for complexity.

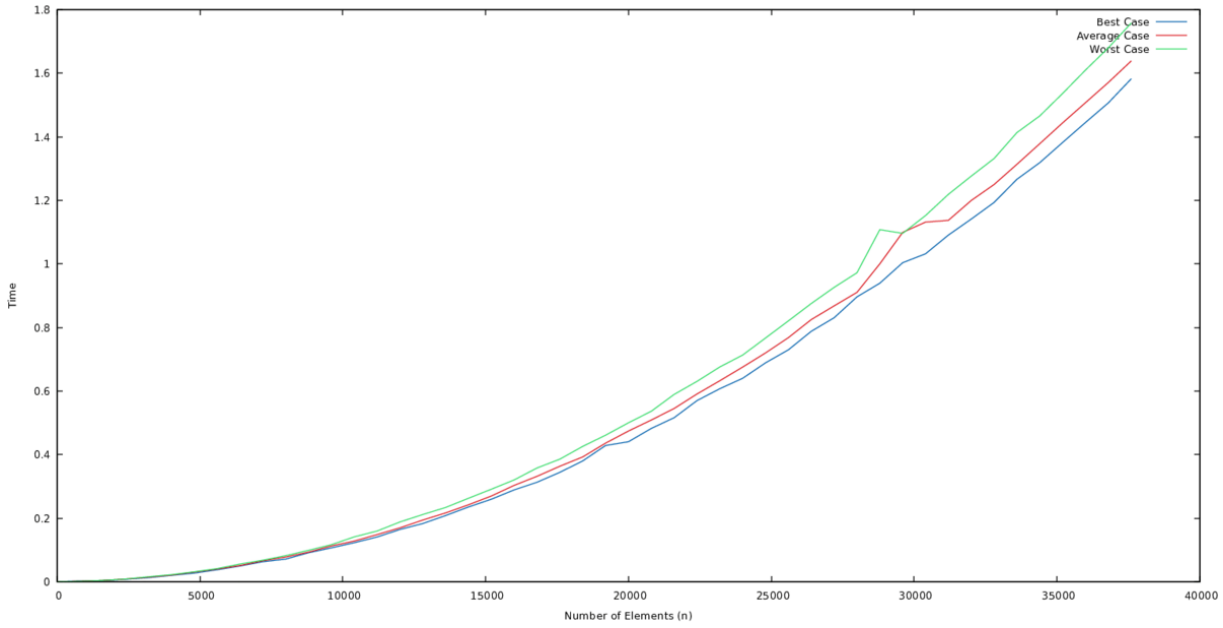
The code is attached.

(d) The curve is given below the data for the plot are in the file data.txt; In the file the first column is n the first value is the time taken by the best case, the second is for the average case and the third is for the worst case.
I plotted the data using gnuplot.

Plot.pdf is also attached in the zip file.
The code for Gnuplot in given below.

```
set xlabel 'Number of Elements(n)'
set ylabel 'Time'
# Axes ranges
set style line 1 linecolor rgb '#0060ad' linetype 1 linewidth 1
set style line 2 linecolor rgb '#dd181f' linetype 1 linewidth 1
set style line 3 linecolor rgb '#44ee66' linetype 1 linewidth 1

plot "data.txt" using 1:2 title "Best Case" with lines linestyle 1,
     "data.txt" using 1:2 title "Average Case" with lines linestyle 2,
     "data.txt" using 1:2 title "Wors Case" with lines linestyle 3
```



(e) **Interpretation**

Here from the selection sort algorithm and the the curve we can interpret that the difference in the cases is to it's minimal.In the array A[n] with the n elements the number of comparisons is n-i+1 in it's i^{th} loop.
So for the n times we have :

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

So the graph is quadratic for all the cases.But the difference in the curve is because of the assignment of min_index and the swapping of the array elements.
This holds for both the best- and worst-case running time.
So the algorithm is $\Theta(n^2)$