Homework 3: Algorithm and Data Structure

Blen Daniel Assefa

February 22

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 ∈ Θ(g), f ∈ O(g), f ∈ o(g), f ∈ Ω(g), f ∈ ω(g), g ∈ Θ(f), g ∈ O(f), g ∈ o(f), g ∈ Ω(f), or g ∈ ω(f).

Table for reference:

$f \in o(g) \quad : \quad \lim_{n\to\infty} f(n)/g(n) = 0$

$f \in O(g) \quad : \quad \lim_{n\to\infty} f(n)/g(n) < \infty$

$f \in \Omega(g) \quad : \quad \lim_{n\to\infty} f(n)/g(n) > 0$

$f \in \Theta(g) \quad : \quad 0 < \lim_{n\to\infty} f(n)/g(n) < \infty$

$f \in \omega(g) \quad : \quad \lim_{n\to\infty} f(n)/g(n) = \infty$

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

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

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

This implies that:
$f \in O(g)$, $f \in o(g)$, $g \in \Omega(f)$ *and* $g \in \omega(f)$

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

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

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

This implies that:
$g \in O(f)$, $g \in o(f)$, $f \in \Omega(g)$ *and* $f \in \omega(g)$

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

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

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

This implies that:

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

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

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

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

This implies that:

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

Problem 3.2 *Asymptotic Analysis*

(a)

Python code:

```
def selectionSort(arr):
    for i in range(0, len(arr)):
        current_min = i
        for j in range(i + 1, len(arr)):
            if(arr[current_min] > arr[j]):
                current_min = j
        if(i != current_min):
            arr[i], arr[current_min] = arr[current_min], arr[i]
    return arr
```

(b)

Initialization:

The loop invariant holds before the first loop iteration if the array given has a length more than one. If the length of the array is greater than one, then i = 0, A[0.. i-1] is empty and an empty loop is sorted already.

Maintenance:

When the loop starts, the loop goes through the list from A[i] to A[n - 1], where n is the length of the array and the element A[i] is the minimum element of the subarray A[i..n]. New element A[i+1] is determined by searching a minimal element in sub-array A[i+1..n] and assigning the index to current_min. The continues until i = n-1.

Termination:

When the loop ends, j=n and all the elements from A[0..j-1] are sorted. So the algorithm works correctly.

(c)

Generation of random numbers for

```
from random import seed
from random import randint

# seed random number generator
seed(1)
# generate some integers

def getArray(size):
    arr = []
    for i in range(0, size):
        arr.append(randint(0, size))
    return arr

# generates Non-repeated number array
def getCheckedArray(size):
    arr = []
    for i in range(0, size):
        value = randint(0, size)
        while(value in arr):
            value = randint(0, size)
        arr.append(value)

    return arr
```

***Case A:** the case which involves the most swaps (Hint: it is not a decreasingly ordered array).*

```
def worstCase(size):

    arr = getCheckedArray(size)
    # sort the numbers
    arr.sort(reverse = True)
    arr[1 : size] = sorted(arr[1:size])

    return arr
```

This is what I believe will make the most swaps since the logic behind the swaps is, if the current positioned number is greater than the next positioned number, but the next positioned number is already sorted from the rest of the array(i.e. Next positioned number is the minimum number from it's index till the end), then the swap will happen between the two. This logic will continue till the end.

***Case B:** the case with the least swaps.*

```
def bestCase(size):
    arr = getCheckedArray(size)
    # sort the numbers
    arr.sort()
    return arr
```

Best case will be when the list is already sorted and there are no swaps.

Average case:

```
def getCheckedArray(size):
    arr = []
    for i in range(0, size):
        value = randint(0, size)
        while(value in arr):
            value = randint(0, size)
        arr.append(value)

    return arr
```

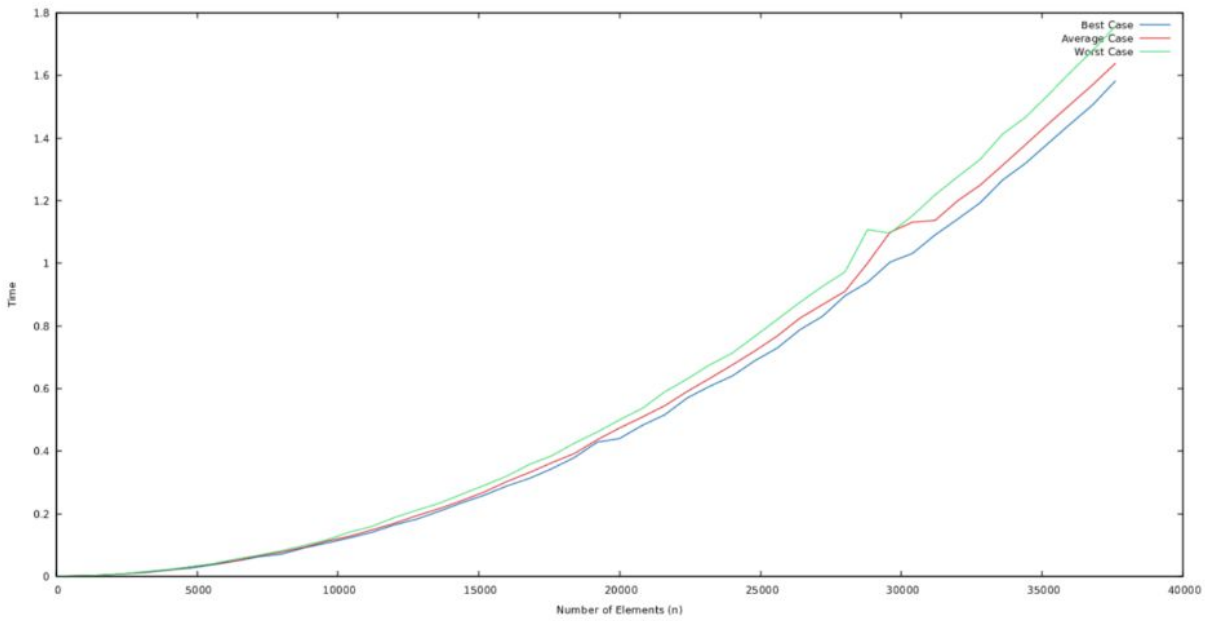
(d)

The data from my python code is named file.txt. It is inside the zip folder.

The code for the GNU - Plot is

```
set terminal pdf
set output "plot_fct.pdf"

# Line width of the axes
set border linewidth 1.5
```

```
# Line styles
set style line 1 linecolor rgb '#0060ad' linetype 1 linewidth 2
set style line 2 linecolor rgb '#dd181f' linetype 1 linewidth 2
set style line 3 linecolor rgb '#00ff00' linetype 1 linewidth 2

set xlabel 'Number of Elements(n)'
set ylabel 'Time'

# plot all functions
plot "file.txt" using 1:2 title 'Best Case' with lines linestyle 1, \
      "file.txt" using 1:3 title 'Average Case' with lines linestyle
2, \
     "file.txt" using 1:4 title 'Worst Case' with lines linestyle 3

unset output
```



(e)

Interpretations

This is because it compares the value at index j with all the remaining values A[j + 1..n] ⇒ n(n+1)/ 2 ⇒ θ(n²) regardless of the arrangements of the elements.