ADS Homework 3

Justin Morris

3.1a)

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

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \to \lim_{n\to\infty} \frac{9n}{5n^3} \to \frac{9}{5n^2} \to \frac{9}{\infty} \to 0$$

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} \to \lim_{n\to\infty} \frac{5n^3}{9n} \to \lim_{n\to\infty} \frac{500^2}{9} \to \frac{5\infty}{9} \to \frac{\infty}{9}$$

Belongs to

$f \in O g(n)$

$f \in O(g)$

$g \in \Omega(f)$

$g \in \omega(f)$

Does not belong -

$f \notin \Omega(g)$

$f \notin \Theta(g)$

$f \notin \omega(g)$

$g \notin o(f)$

$g \notin \Theta(f)$

$g \notin O(f)$

3.1b)

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

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

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

Belongs to	Does not belong
$f \in \Omega(g)$	$f \notin \Theta(g)$
$f \in \omega(g)$	$f \notin O(g)$
$g \in O(f)$	$f \notin o(g)$
$g \in o(f)$	$g \notin \Theta(f)$
	$g \notin \Omega(f)$
	$g \notin \omega(f)$

3.1 C)

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

$$\lim_{n\to\infty} \frac{\frac{n^2}{\log(n)}}{n\log n} \to \lim_{n\to\infty} \frac{n^2}{\log(n)} \cdot \frac{1}{n\log n} \to \frac{n}{\log(n)^2} \to \infty$$

$$\lim_{n\to\infty} \frac{n\log n}{\frac{n^2}{\log(n)}} \to \frac{n\log n}{1} \cdot \frac{\log n}{n^2} \to \frac{\log n^2}{n} \to 0$$

Belongs to	Does not belong
$f \in \Omega(g)$	$f \notin o(g)$
$f \in \omega(g)$	$f \notin O(g)$

$g \in o(f)$

$g \in O(g)$

$f \notin \Theta(g)$

$g \notin \Omega(f)$

$g \notin \Theta(f)$

$g \notin \omega(f)$

3.1d)

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

$$\lim_{n\to\infty} \frac{(\log(3n))^3}{9\log(n)} \to \lim_{n\to\infty} \frac{(\log(3))^3 + (\log(n))^3}{9\log(n)} \to \lim_{n\to\infty} \frac{(\log(n))^3}{9\log(n)} \to$$

$$\lim_{n\to\infty} \frac{(\log(n))^2}{9} \to \infty$$

$$\lim_{n\to\infty} \frac{9\log(n)}{(\log(3n))^3} \to \frac{9\log(n)}{(\log(3))^3 + (\log(n))^3} \to \frac{9}{(\log(n))^3} \to$$

$$\frac{9}{(\log(n))^3} \to 0$$

Belongs to

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

Does not belong

$f \notin o(g)$
$f \notin O(g)$
$f \notin \Theta(g)$
$g \notin \Omega(f)$
$g \notin \Theta(f)$
$g \notin \omega(f)$

3.2 A)

selection sort

```
ofstream out;

// help from https://www.geeksforgeeks.org/selection-sort/ for selection sort
void selectionSort(int arr[], int size){
    // run for each number in larger caseBay
    for(int i = 0; i < size-1; i++){

        // set min value index to iterator
        int minIndex = i;

        // find the minimum element
        for (int j = i+1; j < (size); j++){
            if(arr[minIndex] > arr[j]){
                minIndex = j;
            }
        }

        // before swapping, make sure not self
        if(minIndex != i){
            int tempSwap = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = tempSwap;
        }
    }
}
```

Find minimum element

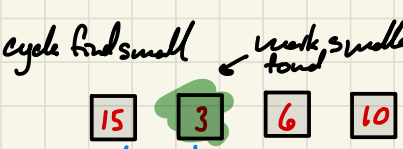
Swap elements

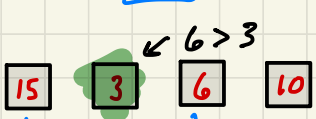
How selection sort works:

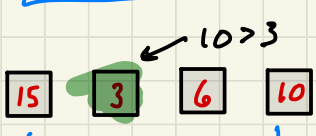
Explanation by example

1) init

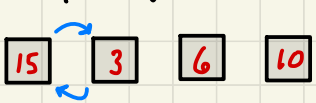
2) cycle findsmall







3) Swap Values



4) repeat using next index

Returned:

```
Original array
9 8 7 6 5 4 3 2 1 70

Sorted array
1 2 3 4 5 6 7 8 9 70
```

when is run

```
int main(){

    int arr[10] = {9,8,7,6,5,4,3,2,1,70};

    cout << "\nOriginal array" << endl;
    for (int i = 0; i < 10; i++){
        cout << arr[i] << " ";
    }
    cout << endl;


    selectionSort(arr, 10);

    cout << "\nSorted array" << endl;
    for (int i = 0; i < 10; i++){
        cout << arr[i] << " ";
    }
    cout << "\n" << endl;
}
```

3.2B) From - https://facultyweb.cs.wwu.edu/~wehrwes/courses/csci241_18f/lectures/L03/L03.pdf

To show that A is correct, we would need to check,

1) Initialization - true at the start.
2) Termination - loop ends when postcondition is true
3) Progress - makes progress to post condition
4) Maintenance - true after each iteration

3.2C)

```
// cycle through the array and insert random elements
for (int j = 0; j < sizeArr; j++){
    arr[j] = (rand() % 100 + 1);
}
```

```
void caseA(int arr[], int sizeArr){

    // open caseA csv in append mode
    out.open("caseA.csv", ios_base::app);

    int max_arr = 0;

    // find the largest element to make the worst case array
    for (int k = 0; k < sizeArr; k++){
        // check array and set new max if found
        if (arr[k] > max_arr){
            max_arr = arr[k];
        }
    }

    // set first element to make worst case, insert in index 0 max * 2
    arr[0] = (max_arr*2);

    // time execution
    auto startCaseA = high_resolution_clock::now();
    selectionSort(arr, sizeArr);
    // stop time by getting current time and subtract to find total time
    auto stopCaseA = high_resolution_clock::now();
    auto caseA = duration_cast<microseconds>(stopCaseA - startCaseA);

    // write and close file
    out << sizeArr << "," << (double)caseA.count() << endl;
    out.close();
}
```

Case A - most swaps

The worst case would be when we have the largest number at the front of the list. SO, we do that... after having generated random values, and finding the max of them to know how long first element needs to be. Then set element at index 0.

Output to csv file

Best case

The best case, 0 swaps would then be an array already in order. Thus, run caseA first and caseB after so it is already sorted.

```
void caseB(int arr[], int sizeArr){

    // open caseB csv in append mode
    out.open("caseB.csv", ios_base::app);

    // get and save current time
    auto startCaseB = high_resolution_clock::now();
    // sort array using selection sort
    selectionSort(arr, sizeArr);
    // get current time and subtract to find execution time
    auto stopCaseB = high_resolution_clock::now();
    auto caseB = duration_cast<microseconds>(stopCaseB - startCaseB);

    // write and close file
    out << sizeArr << "," << (double)caseB.count() << endl;
    out.close();
}
```

3.2D)

Main

```
int main(){

    // cycle through 2000, iterating at 15
    for(int i = 0; i < 2000; i+=15){

        // rename the size of the array
        int sizeArr = i;
        // create the array
        int arr[sizeArr];

        // cycle through the array and insert random elements
        for (int j = 0; j < sizeArr; j++){
            arr[j] = (rand() % 100 + 1);
        }

        // case A
        caseA(arr, sizeArr);

        // Case B
        // case B will always be sorted as it is after case A which is sorted
        caseB(arr, sizeArr);

        // case C
        caseC(sizeArr);
    }

    return 0;
}
```

Run enough times to get data, but not every n value as that would take a long time

New array length, this new array isn't

Random elements

Run cases (functions)

Case C

```
void caseC(int sizeArr){

    // open caseC csv in append mode
    out.open("caseC.csv", ios_base::app);

    // create double to store summed execution time
    double caseCSum = 0.0;

    // run to get reliable average time as noted in question
    for (int m = 0; m < 10; m++){

        // creating array and add elements
        int arr[sizeArr];
        for (int j = 0; j < sizeArr; j++){
            arr[sizeArr] = (rand() % 100 + 1);
        }

        // get and save current time
        auto startCaseC = high_resolution_clock::now();
        // sort array using selection sort
        selectionSort(arr, sizeArr);
        // get current time and subtract to find execution time
        auto stopCaseC = high_resolution_clock::now();
        auto caseCs = duration_cast<microseconds>(stopCaseC - startCaseC);
        // sum execution times
        caseCSum += caseCs.count();
    }

    // write, take average for 15 runs and close
    out << sizeArr << "," << caseCSum/(float)10 << endl;
    out.close();
}
```

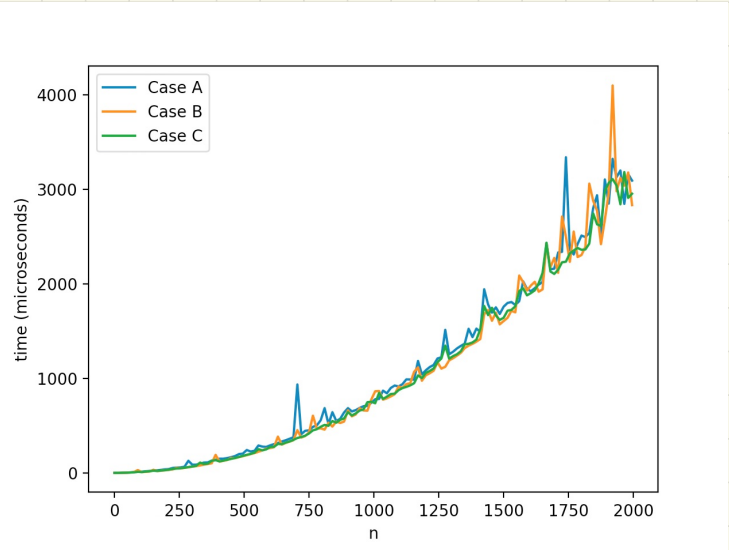
Idea is to run multiple times to get reliable average. Thus, run 10 times, new value in array, seven times, Average and save to CSV (:

Post getting data, plot using pyplot. Installed env and used pip to install.

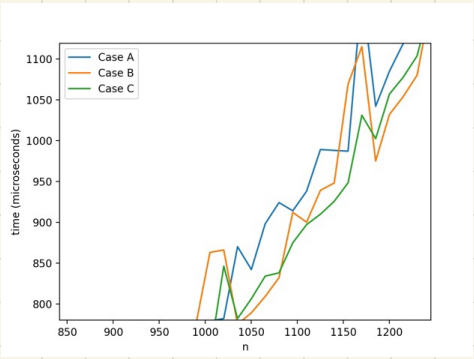
```python
import matplotlib.pyplot as plt
import numpy as np

# help with plotting from (https://www.kite.com/python/answers/how-to-plot-data-from-a-csv-file-in-python)
caseA = np.genfromtxt('First year/Spring 2020/ADS/Homework 3/caseA.csv', delimiter=",", names=["x", "y"])
caseB = np.genfromtxt('First year/Spring 2020/ADS/Homework 3/caseB.csv', delimiter=",", names=["x", "y"])
caseC = np.genfromtxt('First year/Spring 2020/ADS/Homework 3/caseC.csv', delimiter=",", names=["x", "y"])

# plot values from lists created from csv files
plt.plot(caseA['x'], caseA['y'], label='Case A')
plt.plot(caseB['x'], caseB['y'],label='Case B')
plt.plot(caseC['x'], caseC['y'], label='Case C')

# set labels
plt.ylabel('time (microseconds)')
plt.xlabel('n')

plt.legend()

# plot graph
plt.show()

if __name__ == "__main__":
    pass
```

Get data from CSVs

plot data



This, forming the graph.



Zoomed in to see best case, average, and worst case. Too seeing how it can differ.

3.2e)

As seen in the graphs, the lines are all pretty close. This reinforces that selection sort has a time complexity of n^2.

Best case: $\Omega(n^2)$

Worst case: $O(n^2)$

Average case: $\Theta(n^2)$

As well, as we know, whatever constants there are, will become non significant.