Algorithms and Data Structures

Spring 2019

Assignment 1

Date: February 12, 2019

Taiyr Begeyev

Problem 1.1 Asymptotic Analysis

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

(a) $f(n) = 3n$ **and** $g(n) = n^3$
1.1 First, let's show $f \in O(g)$:
For $3n \in O(n^3)$, we have to find c and n_0 such that $3n \leq c \cdot n^3, \forall n \geq n_0$. Let us check this condition if: $3n \leq c \cdot n^3$, then $\frac{3}{n^2} \leq c$. Therefore, the Big-O condition holds for $n \geq n_0 = 1$ and $c \geq 3$. Larger values of n_0 result in smaller factors c (e.g., for $n_0 = 10$ $c \geq 0.03$ and so on), but in any case the above statement is valid. $f \in O(g)$

1.2 Secondly, let's show $f \in \Omega(g)$:
For $3n \in \Omega(n^3)$, we have to find c and n_0 such that $c \cdot n^3 \leq 3n, \forall n \geq n_0$. Let us check this condition if: $c \cdot n^3 \leq 3n$, then $c \leq \frac{3}{n^2}$. Since

$$\lim_{n \to \infty} \frac{3}{n^2} = 0,$$

it means that the Big-Omega condition doesn't hold. $f \notin \Omega(g)$

1.3 Now let's consider $f \in \Theta(g)$:
Since $\Theta(g) = O(g) \cap \Omega(g)$ and as claimed in 1.2, it follows that $f \notin \Theta(g)$.

1.4 Let's show $f \in o(g)$:
Since $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{3n}{n^3} = \lim_{n \to \infty} \frac{3}{n^2} = 0$, it means that the function $f(n)$ becomes insignificant relative to $g(n)$ as n approaches infinity. Therefore, $f \in o(g)$.

1.5 And the last, but not least $f \in \omega(g)$:
Since $\lim_{n \to \infty} \frac{3}{n^2} \neq \infty$, $f \notin \omega(g)$.

1.6 $g \in O(f)$:
As claimed in 1.2, $f \notin \Omega(g)$. According to transpose symmetry:

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n)),$$

it follows that $g \notin O(f)$.

1.7 $g \in \Omega(f)$:
As claimed in 1.1, $f \in O(g)$. According to transpose symmetry:

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n)),$$

it follows that $g \in \Omega(f)$.

1.8 Now let's consider $g \in \Theta(f)$:
Since $\Theta(f) = O(f) \cap \Omega(f)$ and as claimed in 1.6, it follows that $g \notin \Theta(f)$.

1.9 Let's show $g \in o(f)$:
Since $\lim_{n \to \infty} \frac{g(n)}{f(n)} = \lim_{n \to \infty} \frac{n^3}{3n} = \lim_{n \to \infty} \frac{n^2}{3} = \infty$, $g \notin o(f)$.

1.10 And the last, but not least $g \in \omega(f)$:
Since $\lim_{n \to \infty} \frac{3}{n^2} = \infty$, $g(n)$ becomes arbitrarily large relative to $f(n)$ as n approaches infinity. Therefore, $g \in \omega(f)$.

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

$$g(n) = n^{1/2}$$

First, let's solve limits

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{7n^{0.7}+2n^{0.2}+13log(n)}{n^{1/2}} = \lim_{n\to\infty}(7n^{0.2} + 2n^{-0.3} + \frac{13log(n)}{\sqrt{n}}) =$$
$$\lim_{n\to\infty} 7n^{0.2} + \lim_{n\to\infty} 2n^{-0.3} + \lim_{n\to\infty} \frac{13log(n)}{\sqrt{n}} = \infty + 0 + 0 = \infty$$

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{n^{1/2}}{7n^{0.7} + 2n^{0.2} + 13log(n)} = 0 \tag{1}$$

2.1 First, let's show $f \in O(g)$:
By definition, $f(n) = O(g(n))$ is equivalent to

$$\exists c \in \mathbb{R} : \lim_{n\to\infty} \frac{f(n)}{g(n)} = c$$

where, $c \geq 0$ (c is finite)
Since $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, in means that $f \notin O(g)$

2.2 Secondly, let's show $f \in \Omega(g)$:
By definition, $f \in \Omega(g)$ is equivalent to

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$$

where $0 < c \leq \infty$
Since $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, in means that $f \in \Omega(g)$

2.3 Let's show $f \in \Theta(g)$:
By definition, $f \in \Theta(g)$ is equivalent to

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$$

where $0 < c < \infty$
Since $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, in means that $f \notin \Theta(g)$

2.4 Let's show $f \in o(g)$:
By definition, $f \in o(g)$ is equivalent to

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

Since $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, in means that $f \notin o(g)$

2.5 Let's show $f \in \omega(g)$:
By definition, $f \in \omega(g)$ is equivalent to

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

Since $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, in means that $f \in \omega(g)$

2.6 As claimed in 2.2, $f \in \Omega(g)$. According to transpose symmetry:

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n)),$$

it follows that $g \in O(f)$.

2.7 $g \in \Omega(f)$:
As claimed in 2.1, $f \notin O(g)$. According to transpose symmetry:

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n)),$$

it follows that $g \notin \Omega(f)$.

2.8 Now let's consider $g \in \Theta(f)$:
Since $\Theta(f) = O(f) \cap \Omega(f)$ and as claimed in 2.6, it follows that $g \notin \Theta(f)$.

2.9 Let's show $g \in o(f)$:
By definition, $g \in o(f)$ is equivalent to

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

Since $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$, in means that $g \in o(f)$

2.10 Let's show $g \in \omega(f)$:
By definition, $g \in \omega(f)$ is equivalent to

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

Since $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$, in means that $g \notin \omega(f)$

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

First, let's solve limits

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

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{nlog^2(n)}{n^2} = 0 \tag{3}$$

3.1 First, let's show $f \in O(g)$:
By definition, $f(n) = O(g(n))$ is equivalent to

$$\exists c \in \mathbb{R} : \lim_{n\to\infty} \frac{f(n)}{g(n)} = c$$

where, $c \geq 0$ (c is finite)
Since $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, in means that $f \notin O(g)$

3.2 Secondly, let's show $f \in \Omega(g)$:
By definition, $f \in \Omega(g)$ is equivalent to

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$$

where $0 < c \leq \infty$
Since $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, in means that $f \in \Omega(g)$

3.3 Let's show $f \in \Theta(g)$:
By definition, $f \in \Theta(g)$ is equivalent to

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$$

where $0 < c < \infty$
Since $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, in means that $f \notin \Theta(g)$

3.4 Let's show $f \in o(g)$:
By definition, $f \in o(g)$ is equivalent to

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

Since $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, in means that $f \notin o(g)$

3.5 Let's show $f \in \omega(g)$:
By definition, $f \in \omega(g)$ is equivalent to

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

Since $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, in means that $f \in \omega(g)$

3.6 As claimed in 3.2, $f \in \Omega(g)$. According to transpose symmetry:

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n)),$$

it follows that $g \in O(f)$.

3.7 $g \in \Omega(f)$:
As claimed in 3.1, $f \notin O(g)$. According to transpose symmetry:

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n)),$$

it follows that $g \notin \Omega(f)$.

3.8 Now let's consider $g \in \Theta(f)$:
Since $\Theta(f) = O(f) \cap \Omega(f)$ and as claimed in 3.6, it follows that $g \notin \Theta(f)$.

3.9 Let's show $g \in o(f)$:
By definition, $g \in o(f)$ is equivalent to

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

Since $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$, in means that $g \in o(f)$

3.10 Let's show $g \in \omega(f)$:
By definition, $g \in \omega(f)$ is equivalent to

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

Since $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$, in means that $g \notin \omega(f)$

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

First, let's solve limits

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{(log(3n))^3}{9log(n)} = \infty \tag{4}$$

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

4.1 First, let's show $f \in O(g)$:
By definition, $f(n) = O(g(n))$ is equivalent to

$$\exists c \in \mathbb{R} : \lim_{n\to\infty} \frac{f(n)}{g(n)} = c$$

where, $c \geq 0$ (c is finite)
Since $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, in means that $f \notin O(g)$

4.2 Secondly, let's show $f \in \Omega(g)$:
By definition, $f \in \Omega(g)$ is equivalent to

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$$

where $0 < c \leq \infty$
Since $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, in means that $f \in \Omega(g)$

4.3 Let's show $f \in \Theta(g)$:
By definition, $f \in \Theta(g)$ is equivalent to

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$$

where $0 < c < \infty$
Since $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, in means that $f \notin \Theta(g)$

4.4 Let's show $f \in o(g)$:
By definition, $f \in o(g)$ is equivalent to

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

Since $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, in means that $f \notin o(g)$

4.5 Let's show $f \in \omega(g)$:
By definition, $f \in \omega(g)$ is equivalent to

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

Since $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, in means that $f \in \omega(g)$

4.6 As claimed in 4.2, $f \in \Omega(g)$. According to transpose symmetry:

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n)),$$

it follows that $g \in O(f)$.

4.7 $g \in \Omega(f)$:
As claimed in 4.1, $f \notin O(g)$. According to transpose symmetry:

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n)),$$

it follows that $g \notin \Omega(f)$.

4.8 Now let's consider $g \in \Theta(f)$:
Since $\Theta(f) = O(f) \cap \Omega(f)$ and as claimed in 4.6, it follows that $g \notin \Theta(f)$.

4.9 Let's show $g \in o(f)$:
By definition, $g \in o(f)$ is equivalent to

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

Since $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$, in means that $g \in o(f)$

4.10 Let's show $g \in \omega(f)$:
By definition, $g \in \omega(f)$ is equivalent to

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

Since $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$, in means that $g \notin \omega(f)$

Problem 1.2 Selection Sort

(a)

```
int* selectionSort(int* arr_local, int length) {
    int positionOfMin, tmp;

    //go through all elements
    for (int i = 0; i < length - 1; i++) {
        //make current element min and remember its position
        positionOfMin = i;
        //find the min in unsorted part
        for (int j = i + 1; j < length; j++) {
            if (arr_local[j] < arr_local[positionOfMin]) {
                positionOfMin = j;
            }
        }

        //swap min with current element if min is not current element
        if (positionOfMin != i) {
            tmp = arr_local[i];
            arr_local[i] = arr_local[positionOfMin];
            arr_local[positionOfMin] = tmp;
        }
    }
    return arr_local;
}
```

(b) Show that Selection Sort is correct
Algorithm is correct if it can be proved that if the pre-condition is true, the post-condition must be true as well. This algorithm contains two **for** loops. Loops invariants give us the power to reason formally about the loops. We need to prove that an invariant is true for all iterations.

input: array A[1..n] of n unsorted integers
output: same integers in array A now in sorted order

Loop Invariant
Before each loop starts, A[positionOfMin] ≤ A [i..j - 1]
Initialization
positionOfMin = i;
We have that positionOfMin indexes the smallest element in array A[i..j - 1] and hence the loop

invariant is true
Maintenance
Before passing j, we assume that positionOfMin indexes the smallest element in array A[i..j - 1]. During execution of j we have two cases
1. A[j] < A[positionOfMin]
2. A[j] ≥ A[positionOfMin]
If there is the second case, nothing happens. If it is the first acse, then we assign the index of the smallest element from subarray A[i..j-1].
Termination
At termination of the inner loop, min indexes an element less than or equal to all elements in subarray A[i..n] since j = n+1 upon termination. This finds the smallest element in this subarray and is useful to us in the outer loop because we can move that next smallest item into the correct location.

(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. Briefly describe how you generated the sequences (e.g., with a random sequence generator using your chosen language).
The best case and the worst case are asymptotically equivalent, since the number of comparisons is the same. It is $O(n^2)$ for all cases. The only difference is on swap operation, which is linear.
The best case is when array is already sorted.
The worst case is when array is reversed.

```
/*
    Algorithms and Data Structures
    Spring 2019
    Assignment #1
    selectionSort.cpp
    Purpose: Implementation of the Selection Sort Algorithm

    @author Taiyr Begeyev
    @version 1.0 13/02/19
*/

#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int* generateArrayOfRandomNumbers (int* arr_local, int n);
int* selectionSort(int* arr_local, int length);
int* reverseOrderOfArray(int* arr_local, int length);
int* copyArray(int* arr1, int* arr2, int length);
void printArray(int* arr_local, int length);

int main() {
    srand(time(NULL));
    clock_t t1, t2, t3;

    //generating random number from 1..20
    int randomNumberOfElements = rand() % 10 + 1;

    //Dynamic array allocation
    int* arr = NULL;
    int* arrBestCase = NULL;
    int* arrWorstCase = NULL;
    arr = new int[randomNumberOfElements]; //average case
    arrBestCase = new int[randomNumberOfElements];
    arrWorstCase = new int[randomNumberOfElements];
```

```
cout << "Number of elements: " << randomNumberOfElements << endl;
arr = generateArrayOfRandomNumbers(arr, randomNumberOfElements);
cout << "Generated array: " << endl;
printArray(arr, randomNumberOfElements);

//copy elements from arr1 to arr2
arrBestCase = copyArray(arr, arrBestCase, randomNumberOfElements);
arrWorstCase = copyArray(arr, arrWorstCase, randomNumberOfElements);

// in best case array is already sorted
// in worst case array is reversed
//we use corresponding functions
arrBestCase = selectionSort(arrBestCase, randomNumberOfElements);
arrWorstCase = reverseOrderOfArray(arrWorstCase, randomNumberOfElements);

cout << endl;
cout << "The array out of generated (average case): " << endl;
printArray(arr, randomNumberOfElements);

// ------------------------------------
// COMPUTATION TIME OF AVERAGE CASE
t1 = clock();
arr = selectionSort(arr, randomNumberOfElements);
cout << "Sorted array: " << endl;
printArray(arr, randomNumberOfElements);

t1 = clock() - t1;
//Stop measuring time here
double duration1 = double(t1) / CLOCKS_PER_SEC;
cout << endl;
// ------------------------------------

cout << "The array out of generated (best case): " << endl;
printArray(arrBestCase, randomNumberOfElements);

// ------------------------------------
// COMPUTATION TIME OF BEST CASE
t2 = clock();
arrBestCase = selectionSort(arrBestCase, randomNumberOfElements);
cout << "Sorted array: " << endl;
printArray(arrBestCase, randomNumberOfElements);

t2 = clock() - t2;
//Stop measuring time here
double duration2 = double(t2) / CLOCKS_PER_SEC;
cout << endl;
// ------------------------------------

cout << "The array out of generated (worst case): " << endl;
printArray(arrWorstCase, randomNumberOfElements);

// ------------------------------------
// COMPUTATION TIME OF THE WORST CASE
t3 = clock();
arrWorstCase = selectionSort(arrWorstCase, randomNumberOfElements);
cout << "Sorted array: " << endl;
printArray(arrWorstCase, randomNumberOfElements);

t3 = clock() - t3;
//Stop measuring time here
```

```
    double duration3 = double(t3) / CLOCKS_PER_SEC;
    cout << endl;
    // ----------------------------------

    /*
        Execution times
    */
    cout << "Execution time for worst case: "<< fixed << duration3 << endl;
    cout << "Execution time for average case: "<< fixed << duration1 << endl;
    cout << "Execution time for best case: "<< fixed << duration2 << endl;

    delete [] arr;
    delete [] arrWorstCase;
    delete [] arrBestCase;
    arr = NULL;
    arrWorstCase = NULL;
    arrBestCase = NULL;
    return 0;
}

/*
    @brief generates array of random numbers in a certain range
    @param array and its length
    @returns randomized array
*/

int* generateArrayOfRandomNumbers (int* arr_local, int n) {
    for(int i = 0; i < n; i++) {
        arr_local[i] = rand() % 200 + 1;
    }
    return arr_local;
}

/*
    @brief implementation of the Selection Sort Algorithm
        At every step, algorithm finds minimal element in the unsorted part and
        adds it to the end of the sorted one.
        When unsorted part becomes empty, algorithm stops.
    @param array and its length
    @returns sorted array
*/

int* selectionSort(int* arr_local, int length) {
    int positionOfMin, tmp;

    //go through all elements
    for (int i = 0; i < length - 1; i++) {
        //make current element min and remember its position
        positionOfMin = i;
        //find tshe min in unsorted part
        for (int j = i + 1; j < length; j++) {
            if (arr_local[j] < arr_local[positionOfMin]) {
                positionOfMin = j;
            }
        }

        //swap min with current element if min is not current element
        if (positionOfMin != i) {
            tmp = arr_local[i];
            arr_local[i] = arr_local[positionOfMin];
```

```
            arr_local[positionOfMin] = tmp;
        }
    }
    return arr_local;
}

/*
    @brief make a reversed order of an array for the worst case
    @param array and its length
    @returns reversed array
*/

int* reverseOrderOfArray(int* arr_local, int length) {
    int positionOfMax, tmp;

    //go through all elements
    for (int i = 0; i < length - 1; i++) {
        //make current element min and remember its position
        positionOfMax = i;
        //find tshe min in unsorted part
        for (int j = i + 1; j < length; j++) {
            if (arr_local[j] > arr_local[positionOfMax]) {
                positionOfMax = j;
            }
        }

        //swap min with current element if min is not current element
        if (positionOfMax != i) {
            tmp = arr_local[i];
            arr_local[i] = arr_local[positionOfMax];
            arr_local[positionOfMax] = tmp;
        }
    }
    return arr_local;
}

/*
    @brief copies elements from Arr1 to Arr 2
    @param arr1
    @param arr2
    @param length of both arrays
    @returns the array where elements from arr1 were copied to
*/

int* copyArray(int* arr1, int* arr2, int length) {
    for (int i = 0; i < length; i++) {
        arr2[i] = arr1[i];
    }
    return arr2;
}

/*
    @brief prints all elements of the array
    @param array and its length
*/

void printArray(int* arr_local, int length) {
    for(int i = 0; i < length; i++) {
        cout << arr_local[i] << " ";
    }
```

```
    cout << endl;
}
```

So I generated an array with a random number as an input filled with random elements. I applied *generateArrayOfRandomNumbers()* function to *arr*. Just to ensure that I use the same array of numbers, I copied arr to *arrBestCase* and *arrWorstCase*. Since the best case is the case with already sorted numbers and the worst case with numbers in decreasing orders, I have sorted them in prior. Then I called *insertionSort* fiunction to all arrays and measured execution time.

(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 the best, average, and worst 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 in your plot. You can use a plotting tool/software of your choice (Gnuplot, R, Matlab, Excel, etc.).

GnuPlot Script

```
set terminal pdf
set output "plot.pdf"

# x and y labels
set xlabel "Input"
set ylabel "Computation time"
set title "Problem 1.2 (d)"

# 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

# x and y ranges
set xrange [0:7000]
set yrange [0:300]

set grid

plot "output" using 5:6 title "Worst Case" with lines linestyle 1,\
"output" using 1:2 title "Average Case" with lines linestyle 2,\
"output" using 3:4 title "Best Case"with lines linestyle 3
```

There is the graph in the end

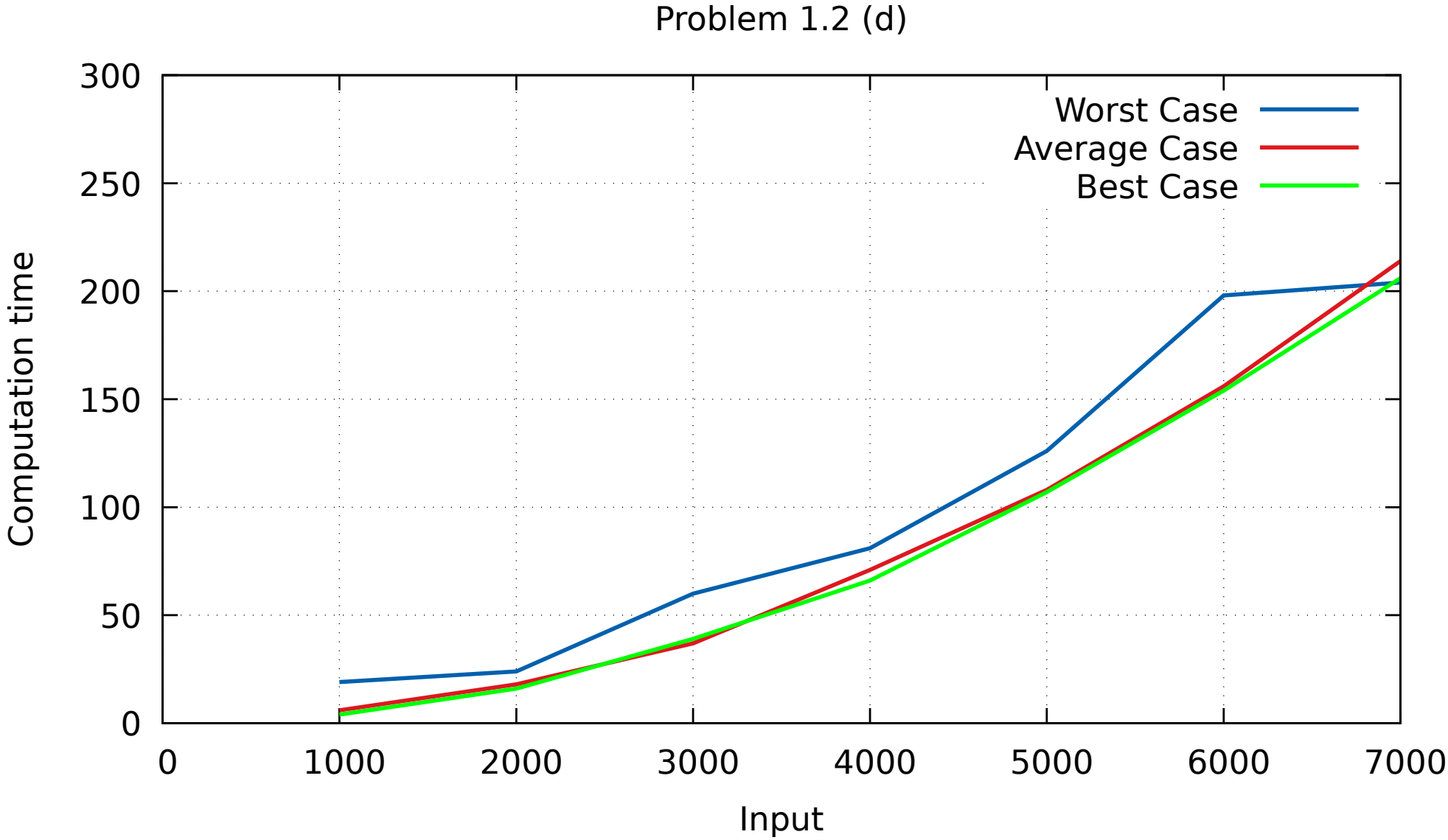
(e) Interpret the plots from (d) with respect to asymptotic behavior and constants

There is practically no difference between all three cases. Selecting the lowest element requires scanning all n elements (this takes n - 1 comparisons) and then swapping it into the first position.

Finding the next lowest element requires scanning the remaining n - 1 elements and so on, therefore

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

It is what we can see from the graph.


Problem 1.2 (d)
Worst Case
Average Case
Best Case
Computation time
Input
0
50
100
150
200
250
300
0
1000
2000
3000
4000
5000
6000
7000