**Algorithms and Data Structures**

**CH08-320201**

**Homework 1**

**Asymptotic Analysis**

**Problem 1.1 Asymptotic Analysis (8 points)**

Considering the following pairs of functions f and g, show for each pair whether 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).

1. (2 points) f(n) = 3n and g(n) = n 3 ,
2. (2 points) f(n) = 7n 0.7 + 2n 0.2 + 13 log n and g(n) = √ n,
3. (2 points) f(n) = n 2/ log n and g(n) = n log n,
4. (2 points) f(n) = (log(3n))3 and g(n) = 9 log n

Please check attached handwritten sheet

**Problem 1.2 Selection Sort (16 points)**

In the lecture Insertion Sort was discussed. Selection Sort is similar to Insertion Sort and works as follows. Given an array of elements, you always take the current element and exchange it with the smallest element that can be found on the right hand side of the current element. In doing so, you will gradually build up a sorted sequence on the left side (like in Insertion Sort), and in each iteration “attach” the smallest element from the remaining unsorted right side to it.

1. **(2 points) Implement Selection Sort.**

*Check executable program file*

1. **(5 points) Show that Selection Sort is correct (consider the loop invariant).**

void selectionSort(int arr[], std::size\_t size)

{

for(int i = 0; i < (size - 1); ++i)

{

int mincount = i;

for(int j = (i + 1); j < size; ++j)

{

if(arr[j] < arr[mincount])

mincount = j;

}

if(mincount != i) std::swap(arr[i], arr[mincount]);

}

}

An array (arr) of n integers is sorted by repetitively selecting the smallest among the yet unselected integers. This is done using a single array by keeping the yet unselected integers at the front of the array, and the selected integers at the back of the array.

A swap function is used to move the selected integer from the front part of the array to the back part of the array. The selected integer is swapped with the integer in location i - 1.

We make use of a simple nested if loop which returns the index of the minimum value in array (arr).

The loop invariant is in terms of mincount, the number of elements already sorted in the array.

“Mincount = 0;“ immediately makes the loop invariant true.

The loop increments “mincount” after every iteration. Before doing this, the smallest from the front of the array needs to be selected and swapped to the back of the array,

{

if(arr[j] < arr[mincount])

mincount = j;

}

if(mincount != i) std::swap(arr[i], arr[mincount]);

Terminate when mincount == i.

1. **(3 points) 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).**

Random Number Generator implemented within code.

We use a number generator in the main function; this would require all my already defined functions to use a size which is an input that is asked from the user. Once the desired size is recorded, it is then loaded into a number generator which then generates x random numbers to populate the array with.

A randomly generated sequence would be the average case for this algorithm since it provides a situation which is completely generated without human consciousness. However, for the best case, we simply take the sorted array that has already gone through the selectionSort function once and give it as input by a function invocation at the main function. This would prove to be the best case.

For the worst case, we make use of the C++ function that reverses any given input. For this, we write a function that can take an array as input and we invoke the array through this reverseArray function before calling upon selectionSort again. This would be the worst case for the Selection Sort algorithm.

Note that the time complexity of Selection Sort is all cases remains constant, and thus, the graphs of the time taken for each iteration of this algorithm will grow in a similar manner.

1. **(5 points) 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.).**

|  |  |  |  |
| --- | --- | --- | --- |
| Input Length (x) | Time Taken for best case (s) | Time Taken for average case (s) | Time Taken for worst case (s) |
| 50 | 0.03999 | 0.01 | 0.0526 |
| 100 | 0.0376 | 0.018 | 0.0686 |
| 150 | 0.0583 | 0.026 | 0.0909 |
| 250 | 0.0686 | 0.0516 | 0.1585 |

1. **(1 point) Interpret the plots from (d) with respect to asymptotic behavior and constants.**

According to the graph plotted, the selection sort is bound asymptotically by 𝑛2.

f(n) = ϴ(𝑛2)