Lab01-Algorithm Analysis

CS2308-Algorithm and Complexity, Xiaofeng Gao, Spring 2022.

∗ If there is any problem, please contact TA Hongjie Fang.

∗ Name:_________ Student ID:_________ Email: ____________

1. Use minimal counterexample to prove that every integer $n \geq 11$ can be written as $5x + 2y$ where x, y are positive integers.

2. Rank the following functions by order of growth with brief explanations: that is, find an arrangement $g_1, g_2, \ldots, g_{10}$ of the functions $g_1 = \Omega(g_2), g_2 = \Omega(g_3), \ldots, g_9 = \Omega(g_{10})$. Partition your list into equivalence classes such that functions $f(n)$ and $g(n)$ are in the same class if and only if $f(n) = \Theta(g(n))$. Use symbols "=" and "$\prec$" to order these functions appropriately. Here $\log n$ stands for $\log_2 n$.

$$\begin{array}{ccccc} 2^{2^n} & n^2 & n! & 2^n & \log^2 n \\ e^n & \log\log n & n \cdot 2^n & n & \log(n^2) \end{array}$$

3. Here are the pseudo-codes of improved BubbleSort (Alg. 1) and QuickSort (Alg. 2).

Algorithm 1: Improved BubbleSort

Input: An array $A[1, \ldots, n]$

Output: A sorted nondecreasingly

```
i ← 1; sorted ← false;
while i ≤ n − 1 and not sorted do
    sorted ← true;
    for j ← n downto i + 1 do
        if A[j] < A[j − 1] then
            swap A[j] and A[j − 1];
            sorted ← false;
    i ← i + 1;
```

Algorithm 2: QuickSort

Input: An array $A[1, \cdots, n]$

Output: A sorted nondecreasingly

```
i ← 1; pivot ← A[n];
for j ← 1 to n − 1 do
    if A[j] < pivot then
        swap A[i] and A[j];
        i ← i + 1;
swap A[i] and A[n];
if i > 1 then
    QuickSort(A[1, ··· , i − 1]);
if i < n then
    QuickSort(A[i + 1, ··· , n]);
```

(a) The key idea of the improved BubbleSort is that we can stop the iteration if there are no swaps during an iteration. Therefore, we use an indicator *sorted* in Alg. 1 to check whether the array is already sorted. Analyze the **best** and **worst** time complexity of the improved BubbleSort.

(b) Analyze the **average** time complexity of the QuickSort in Alg. 2.

(c) To avoid the worst case of QuickSort from happening too often, in practice we can randomly shuffle the sequence before sorting. Follow this idea and Alg. 2 to implement QuickSort in C++. You only need to complete the `TODO` part in `Lab01-QuickSort.cpp`. *(Hint: you can use the built-in function* ***`random_shuffle(...)`*** *in C++* ***`<algorithm>`*** *library to randomly shuffle the sequence before sorting. Other built-in sorting functions such as* ***`sort(...)`*** *in C++ are* ***NOT*** *allowed to use.)*

(d) *(Bonus)* Analyze the **average** time complexity of the improved BubbleSort in Alg. 1. *(Hint: consider the relation between average number of comparisons and interchanges.)*

Remark: You need to include your .pdf, .tex and .cpp files in your uploaded .rar or .zip file.