Lab01-AlgorithmAnalysis

CS214-Algorithm and Complexity, Xiaofeng Gao, Spring 2020.

∗ If there is any problem, please contact TA Shuodian Yu.

∗ Name:_____ Student ID:_____ Email: _____

1. Please analyze the time complexity of Alg. 1 with brief explanations.

Algorithm 1: PSUM

Input: $n = k^2$, k is a positive integer.
Output: $\sum_{i=1}^{j} i$ for each perfect square j between 1 and n.

```
1 k ← √n;
2 for j ← 1 to k do
3 |   sum[j] ← 0;
4 |   for i ← 1 to j^2 do
5 |   |   sum[j] ← sum[j] + i;
6 return sum[1 ··· k];
```

Solution. The outer loop is executed k times, while the inner is executed $\sum_{j=1}^{n} 1$ times. Thus,

$$T(n) = \sum_{j=1}^{k} \sum_{i=1}^{j^2} 1 = \sum_{j=1}^{\sqrt{n}} j^2 = \frac{\sqrt{n}(\sqrt{n}+1)(2\sqrt{n}+1)}{6} = \Theta(n^{1.5})$$

The time complexity is $\Theta(n^{1.5})$. □

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

Algorithm 2: QuickSort

Input: An array $A[1, \cdots, n]$
Output: $A[1, \cdots, n]$ sorted ~~nondecreasingly~~

```
1 pivot ← A[n]; i ← 1;
2 for j ← 1 to n − 1 do
3 |   if A[j] < pivot then
4 |   |   swap A[i] and A[j];
5 |   |   i ← i + 1;
6 swap A[i] and A[n];
7 if i > 1 then QuickSort(A[1, ··· , i − 1]);
8 if i < n then QuickSort(A[i + 1, ··· , n]);
```

Solution. Each time the function partition an array, assume the pivot position is $i \in \{1, \ldots, n\}$, we will get two subarrays with size $i-1$ and $n-i$, and the number of operations for partition is n.

Thus, the time complexity should be $T(n) = T(i-1) + T(n-i) + O(n)$.

On the average case, the complexity is

$$\begin{aligned} T(n) &= \frac{1}{n}\sum_{i=1}^{n}(T(i-1)+T(n-i)+O(n)) \\ &= \frac{2}{n}+O(n) \\ nT(n) &= 2\sum_{i=0}^{n-1}T(i)+O(n^2) \end{aligned}$$

We substract it with the previous term $(n-1)T(n-1)$ and get

$$\begin{aligned} nT(n)-(n-1)T(n-1) &= 2T(n-1)+O(n) \\ \frac{T(n)}{n+1} &= \frac{T(n-1)}{n}+O(\frac{1}{n+1}) \\ &= \frac{T(0)}{1}+O(\sum_{i=2}^{n+1}\frac{1}{i}) \\ &= O(\log n) \\ T(n) &= O(n\log n) \end{aligned}$$

□

3. The BubbleSort mentioned in class can be improved by stopping in time if there are no swaps during an iteration. An indicator is used thereby to check whether the array is already sorted. Analyze the **average** and **best** time complexity of the improved BubbleSort in Alg. 3.

Algorithm 3: BubbleSort

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

```
1 i ← 1;sorted ← false;
2 while i ≤ n − 1 and not sorted do
3     sorted ← true;
4     for j ← n downto i + 1 do
5         if A[j] < A[j − 1] then
6             interchange A[j] and A[j − 1];
7             sorted ← false;
8     i ← i + 1;
```

Solution. The **best** case is when the array is already sorted and there is no interchange. The algorithm will take $n-1$ comparisoins in the inner for loop. So the best time complexity is $\Omega(n)$.

For the **average** case, the time copmlexity $T(n)$ can be divided into two parts: the comparison and the interchanges. We know that the number of interchanges is no more than the number of comparisons. And the number of interchanges depends on the number of inversions in the initial array, which is a pair of elements $(A[i], A[j])$ satisfying $A[i] > A[j]$ while $1 \le i < j \le n$. Because each swap operation will reduce exactly one inversion.

Moreover, for each pair $(A[i], A[j]), 1 \leq i < j \leq n$, the probability that it is an inversion should be 50% in the average case. Thus, the expected number of inversions in the average case is $\frac{n(n-1)}{4}$. The time complexity $T(n)$ should be not less than $O(n^2)$.

In the lecture we have also discussed the worst case of BubbleSort, the time complexity of which is $O(n^2)$ as well. As long as the complexity of average case can not exceed the worst case, the **average** time complexity is $O(n^2)$. □

4. Rank the following functions by order of growth with brief explanations: that is, find an arrangement $g_1, g_2, \ldots, g_{15}$ of the functions $g_1 = \Omega(g_2), g_2 = \Omega(g_3), \ldots, g_{14} = \Omega(g_{15})$. 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 $\log_2 n$.

$2^{\log n}$	$(\log n)^{\log n}$	n^2	$n!$	$(n+1)!$
2^n	n^3	$\log^2 n$	e^n	2^{2^n}
$\log \log n$	$n \cdot 2^n$	n	$\log n$	$4^{\log n}$

Solution.

$$\log \log n \prec \log n \prec \log^2 n \prec n = 2^{\log n} \prec n^2 = 4^{\log n} \prec n^3$$
$$\prec (\log n)^{\log n} \prec 2^n \prec n \cdot 2^n \prec e^n \prec n! \prec (n+1)! \prec 2^{2^n}$$

□

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