Lab01-Algorithm Analysis

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

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

∗ Name: Zhenran Xiao   Student ID: 520030910281   Email: xiaozhenran@sjtu.edu.cn

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

   Proof. For smaller integer, obviously the proposition is true:
   $11 = 5 \times 1 + 2 \times 3, 12 = 5 \times 2 + 2 \times 1$;
   For bigger integer, assume that all the integers which do not comfort to the proposition belong to Set $\mathbf{M}$, and k is the smallest integer in $\mathbf{M}$.
   Obviously, $k - 2$ doesn't belong to $\mathbf{M}$. It is an integer that comforts to the propostition. So $k - 2 = 5x + 2y$.
   Then we can easily find that $k = k - 2 + 2 = 5x + 2y + 2 = 5x + 2(y + 1)$. This contradicts the hypothesis.
   So the hypothesis is not true. The original proposition has been proved. □

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 "≺" to order these functions appropriately. Here $\log n$ stands for $\log_2 n$.

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

   Solution. $\log\log n \prec \log(n^2) \prec \log^2 n \prec n \prec n^2 \prec 2^n \prec n \cdot 2^n \prec e^n \prec n! \prec 2^{2^n}$
   Explanations:
   (1) $\log^2 n \prec n$
   According to L'Hopital's rule,

$$\lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = 0$$

   So $\log n \prec \sqrt{n}$. So $\log^2 n \prec n$.
   (2) $n \cdot 2^n \prec e^n$

$$n \cdot 2^n \prec (\frac{e}{2})^n \cdot 2^n = e^n$$

   So $n \cdot 2^n \prec e^n$.
   (3)$e^n \prec n!$

$$\lim_{n\to\infty} \frac{e^n}{n!} = \frac{e}{1} \cdot \frac{e}{2} \cdot \frac{e}{3} \cdot \ldots \cdot \frac{e}{n} = 0$$

   So $e^n \prec n!$.
   The other "≺"s are obvious. □

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

Algorithm 1: Improved BubbleSort

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

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.)*

Solution.

(a) Best Case: $\Theta(n)$. The array has been sorted before BubbleSort.
Worst Case: $\Theta(n^2)$. BubbleSort goes through the whole array.

(b) For partition, the cost is $n-1$.
The results of partition are equally possible. It means that after partition i takes each value in $[0, n-1]$ with equal probability. There are i elements on the left of the pivot and $n-1-i$ elements on the right.
So we can make out the recursive formula:

$$T(n) = (n-1) + \sum_{i=0}^{n-1} \frac{1}{n}[T(i) + T(n-1-i)], n \geq 2$$

And we can easily know $T(1) = T(0) = 0$. And according to the symmetry, $\sum_{i=0}^{n-1} T(i) = \sum_{i=0}^{n-1} T(n-1-i)$.

So the recursive formula can be changed into:

$$T(n) = (n-1) + \frac{2}{n}\sum_{i=0}^{n-1} T(i), n \geq 2$$

We can use dislocation subtraction to work out the final result:

$$nT(n) = n(n-1) + 2\sum_{i=0}^{n-1} T(i), (n-1)T(n-1) = (n-1)(n-2) + 2\sum_{i=0}^{n-2} T(i)$$

$$\Rightarrow nT(n) - (n-1)T(n-1) = 2(n-1) + 2T(n-1)$$
$$\Rightarrow nT(n) = (n+1)T(n-1) + 2(n-1)$$
$$\Rightarrow \frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{2(n-1)}{n(n+1)}$$

Set $X(n) = \frac{T(n)}{n+1}$, thus

$$X(n) = X(n-1) + \frac{2(n-1)}{n(n+1)}, X(1) = X(0) = 0$$

$$\Rightarrow X(n) - \frac{2}{n+1} = X(n-1) - \frac{2}{n} + \frac{2}{n+1}$$

Set $Y(n) = X(n) - \frac{2}{n+1}$, thus

$$Y(n) = Y(n-1) + \frac{2}{n+1}, Y(1) = -1, Y(0) = -2$$

$$\Rightarrow Y(n) = \sum_{i=0}^{n} \frac{2}{i+1} \sim \ln n + c$$

$$\Rightarrow X(n) = Y(n) + \frac{2}{n+1} \sim O(\log n)$$
$$\Rightarrow T(n) = (n+1)X(n) \sim O(n \log n)$$

Therefore, the average time complexity of the QuickSort is O(nlog n).

(c) See it in `Lab01_QuickSort.cpp`.

(d) Assume that the number of comparisons is c and the number of interchanges is i. So the time cost is

$$COST = c + i$$

And we can easily know there must be $c \geq i$. So there is

$$COST \geq 2i$$

About interchanging, according to the BubbleSort algorithm, we can know that its core idea is turning disordered pairs to ordered pairs. In fact, each interchange eliminates a disordered pair.
Assume that there are d disordered pairs and o ordered pairs in a series. We can know that

$$d + o = \binom{n}{2} = \frac{n(n-1)}{2}$$

Considering the possibility of each possible series is equal and the symmetry between ordered pairs and disordered pairs, the average number of disordered pairs in a series should be

$$\bar{d} = \frac{n(n-1)}{4}$$

Because each interchange eliminates a disordered pair, so

$$\bar{i} = \bar{d} = \frac{n(n-1)}{4}$$

Therefore,

$$T(n) = \overline{COST} \geq 2\bar{i} = \frac{n(n-1)}{2}$$

The worst case of the improved BubbleSort is $\Theta(n^2)$, so the average time complexity is $\Theta(n^2)$.

$\square$

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