Lab01-Algorithm Analysis

Exercises for Algorithms by Xiaofeng Gao, 2019 Autumn Semester.

∗ If there is any problem, please contact TA Wenyi Xu.
Name:Hua Qin Student ID:019033910022 Email: huaqin@sjtu.edu.cn

1. Use the minimal counterexample principle to prove that for any integer $n > 17$, there exist integers $i_n \geq 0$ and $j_n \geq 0$, such that $n = i_n \times 4 + j_n \times 7$.

   Proof. If $n = i_n \times 4 + j_n \times 7$, $i_n \geq 0$, $j_n \geq 0$ is not true for every $n > 17$, then there are values of n for which $n = i_n \times 4 + j_n \times 7$, $i_n \geq 0$, $j_n \geq 0$ is false, and there must be a smallest such value, say $n = k$. Since $18 = 1 \times 4 + 2 \times 7$, which is true for this proposition. Then we have $k \geq 19$, and $k - 1 \geq 18$.

   Since k is the smallest value for which $k = i_k \times 4 + j_k \times 7$, $i_k \geq 0$, $j_k \geq 0$ is false, then $k - 1 = i_{k-1} \times 4 + j_{k-1} \times 7$, $i_{k-1} \geq 0$ and $j_{k-1} \geq 0$ is true.

   However, we have

   $$\begin{aligned} k &= k - 1 + 1 \\ &= i_{k-1} \times 4 + j_{k-1} \times 7 + 2 \times 4 - 1 \times 7 \\ &= (i_{k-1} + 2) \times 4 + (j_{k-1} - 1) \times 7 \end{aligned}$$

   Since $i_{k-1} \geq 0$, then $i_{k-1} + 2 \geq 0$. As for j_{k-1}, if j_{k-1}>0, then $k = i_k \times 4 + j_k \times 7$ is true and $i_k = i_{k-1} + 2$, $j_k = j_{k-1} - 1$. In the case of $j_{k-1} = 0$, we have found that $k - 1 \geq 18$. Since $j_{k-1} = 0$, $i_{k-1} \geq 5$. We have

   $$\begin{aligned} k &= k - 1 + 1 \\ &= i_{k-1} \times 4 + 1 \\ &= (i_{k-1} - 5) \times 4 + 4 \times 5 + 1 \\ &= (i_{k-1} - 5) \times 4 + 3 \times 7 \end{aligned}$$

   As we can see, $k = i_k \times 4 + j_k \times 7$, $i_k \geq 0$, $j_k \geq 0$ is true if $k - 1 = i_{k-1} \times 4 + j_{k-1} \times 7$, $i_{k-1} \geq 0$ and $j_{k-1} \geq 0$ is true. We have derived a contradiction, which allows us to conclude that our original assumption is false. Then we have proofed that for any integer $n > 17$, there exist integers $i_n \geq 0$ and $j_n \geq 0$, such that $n = i_n \times 4 + j_n \times 7$. □

2. Suppose $a_0 = 1$, $a_1 = 2$, $a_2 = 3$, and $a_k = a_{k-1} + a_{k-2} + a_{k-3}$ for $k \geq 3$. Use the strong principle of mathematical induction to prove that $a_n \leq 2^n$ for any integer $n \geq 0$.

   Proof. Induction hypothesis. For $k \geq 0$ and $0 \leq n \leq k$, $a_n \leq 2^n$ is true.

   Proof of induction step. We need to prove $a_{k+1} \leq 2^{k+1}$ is true.

   If k<3, since $a_0 = 1 \leq 2^0$, $a_1 = 2 \leq 2^1$ and $a_2 = 3 \leq 2^2$, which means that induction hypothesis is true in this case, then we can prove that $a_3 = 1 + 2 + 3 = 6 \leq 2^3$ is true. We can prove that $a_{k+1} \leq 2^{k+1}$ is true when k<3.

   If $k \geq 3$, according to induction hypothesis, we have

   $$\begin{aligned} a_{k+1} &= a_k + a_{k-1} + a_{k-2} \\ &\leq 2^k + a_{k-1} + a_{k-2} + a_{k-3} - a_{k-3} \\ &= 2^k + a_k - a_{k-3} \\ &\leq 2^k + 2^k - a_{k-3} \\ &= 2^{k+1} - a_{k-3} \\ &\leq 2^{k+1} \end{aligned}$$

Obviously, $a_n \geq 0$, $n \geq 0$, and $a_n \leq 2^n$, $0 \leq n \leq k$ then we can prove that $a_{k+1} \leq 2^{k+1}$. Thus we can prove that $a_n \leq 2^n$ for any integer $n \geq 0$. □

3. For Algorithm 1 and Algorithm 2 shown below, what is the time complexity these two Algorithms? Express in O, Ω and Θ notation.

Algorithm 1: COUNT1

```
   Input: n
1  count ← 0;
2  for i ← 1 to n do
3  |   j ← i;
4  |   while j ≠ 0 do
5  |   |   j ← j − (j and (j xor (j − 1)));
6  |   |   count ← count + 1;
```

Algorithm 2: COUNT2

```
   Input: n
1  count ← 0;
2  if n is even then
3  |   for i ← 1 to n do
4  |   |   j ← i;
5  |   |   while j ≠ 0 do
6  |   |   |   j ←
   |   |   |    j − (j and (j xor (j − 1)));
7  |   |   |   count ← count + 1;
8  else
9  |   j ← ⌊n/2⌋;
10 |   while j ≥ 1 do
11 |   |   count ← count + 1;
12 |   |   j ← ⌊j/2⌋;
```

Solution. Algorithm 1. For j, we have:

$$j = \sum_{k=0}^{\lfloor \log 2j \rfloor} a_k \times 1 \tag{1}$$

In eq.1, a_k can be 0 or 1, while $a_{\lfloor \log 2k \rfloor} = 1$. Since we can express j as eq.1, we have:

$$j - (j \text{ and } (j \text{ xor } (j-1))) = j - (j \mod 2^{l+1}) \tag{2}$$

In eq.2, $a_l = 1$ and for each $m<l$, $a_m = 0$. As we can see, the times of while loop's execution equals the number of a_k, which $a_k = 1$ in eq.1. For each $j \leq i$, the number of a_k, which $a_k = 1$ $0<N(j) \leq \lfloor \log 2j \rfloor$.

The time complexity can be Expressed in $O(n \times \log 2n)$, $\Omega(n \times \log 2n)$ and $\Theta(n \times \log 2n)$.

Algorithm 2.

If n is even the time complexity is same as **Algorithm 1**. However, if n is odd, the time complexity is $\Theta(\log 2n)$.

Above all, the time complexity is $\Omega(\log 2n)$ and $O(n \times \log 2n)$.

□