Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen
Dmytro Chibisov
Sandeep Sadanandan

Winter Semester 2007/08
Solution Sheet 1
October 31, 2007

Fundamental Algorithms

Problem 1 (10 Points)

Calculate the cost of calculating n^{th} Fibonacci number, using the recursive algorithm $F(n) = F(n-1) + F(n-2)$

Solution

First let's try to solve it using trial and error method. Let's examine the first few numbers of the series.

n	1	2	3	4	5	6	7	8
F(n)	1	1	2	3	5	8	13	21
T(n) = T(F(n))	0	0	3	6	12	21	36	60

From a careful analysis, we can see that $T(n) = 3F(n) - 3$. Let's propose this to be the value of $T(n)$ and see whether we can prove this. We use induction to prove this.

Case $n = 1$ We can see that $T(0) = 3F(0) - 3 = 0$

Case $n = 2$ We can see that $T(1) = 3F(1) - 3 = 0$

Case $n > 2$ Assume that $T(n) = 3F(n) - 3$ is true for all $m < n$.

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + 3 \\ &= 3F(n-1) - 3 + 3F(n-2) - 3 + 3 \\ &= 3(F(n-1) + F(n-2)) + (3 - 3 - 3) \\ &= 3F(n) - 3 \end{aligned}$$

Hence proved. So, the cost of calculation of n^{th} Fibonacci number is $3F(n) - 3$.

Problem 2 (10 Points)

Show: $\left\lfloor 2^{\frac{n-1}{2}} \right\rfloor \leq F(n) \leq \left\lfloor 2^{\frac{n+1}{2}} \right\rfloor$

Solution

As in the above exercise, we can use induction to prove this.

Case $n = 1$: $\lfloor 2^0 \rfloor \leq 1 \leq \lfloor 2^1 \rfloor$

Case $n = 2$: $\lfloor 2^0 \rfloor \leq 1 \leq \left\lfloor 2^{\frac{3}{2}} \right\rfloor$

Case $n > 2$: Assume that $\left\lfloor 2^{\frac{n-1}{2}} \right\rfloor \leq F(n) \leq \left\lfloor 2^{\frac{n+1}{2}} \right\rfloor$ is true for all $m < n$.

1. $\left\lfloor 2^{\frac{n-1}{2}} \right\rfloor \leq F(n)$

$$\begin{aligned}
F(n) &= F(n-1) + F(n-2) \\
&\geq \left\lfloor 2^{\frac{n-1-1}{2}} \right\rfloor + \left\lfloor 2^{\frac{n-2-1}{2}} \right\rfloor \\
&= \left\lfloor 2^{\frac{n-2}{2}} \right\rfloor + \left\lfloor 2^{\frac{n-3}{2}} \right\rfloor \\
&\geq \left\lfloor 2^{\frac{n-3}{2}} \right\rfloor (\left\lfloor 2^{\frac{1}{2}} \right\rfloor + 1) \\
&= \left\lfloor 2^{\frac{n-3}{2}} \right\rfloor (1 + 1) \\
&= \left\lfloor 2^{\frac{n-1}{2}} \right\rfloor
\end{aligned}$$

2. $F(n) \leq \left\lfloor 2^{\frac{n+1}{2}} \right\rfloor$ (Very similar to the above)

$$\begin{aligned}
F(n) &= F(n-1) + F(n-2) \\
&\leq \left\lfloor 2^{\frac{n-1+1}{2}} \right\rfloor + \left\lfloor 2^{\frac{n-2+1}{2}} \right\rfloor \\
&= \lfloor 2^{\frac{n}{2}} \rfloor + \left\lfloor 2^{\frac{n-1}{2}} \right\rfloor \\
&\leq \lfloor 2^{\frac{n}{2}} \rfloor (1 + \left\lfloor 2^{\frac{-1}{2}} \right\rfloor) \\
&= \lfloor 2^{\frac{n}{2}} \rfloor (1 + 0) \\
&= \lfloor 2^{\frac{n}{2}} \rfloor (\left\lfloor 2^{\frac{1}{2}} \right\rfloor) \\
&\leq \left\lfloor 2^{\frac{n+1}{2}} \right\rfloor
\end{aligned}$$

Problem 3 (10 Points)

Let SuperComputer be a very fast computer which can perform 10^9 operations per second, for some problems of size n the table below lists the number of operations necessary.

More specifically, the i^{th} algorithm needs $t_i(n)$ operations.

$$\begin{aligned}
t_1(n) &= 2 \cdot n \\
t_2(n) &= n \lg(n) \\
t_3(n) &= 2.5n^2 \\
t_4(n) &= \frac{1}{1000} \cdot n^3 \\
t_5(n) &= 3^n
\end{aligned}$$

Determine, for which maximal input sizes each algorithm needs at most 1 second, 1 minute, 1 hour. How do these values change, if the computer is upgraded to be 10 times faster (i.e., can do 10^{10} operations)?

Solution

If N is the number of operations which the computer can do in time t (which is actually $10^9 \cdot t$ here), we need to find the value of n for each of the algorithms which will need $t_i(n) \leq N$.

If we take the first case, the algorithm needs $2 \cdot n$ operations for an input size of n.

So we need a value n such that, $2 \cdot n \leq 10^9 \cdot t$. Which will be $5 \cdot 10^8 \cdot t$. Now, let's calculate this for all the algorithms

$$2 \cdot n \leq 10^9 \cdot t \quad \Rightarrow \quad n \leq 5 \cdot 10^8 \cdot t$$

$$n \lg(n) \leq 10^9 \cdot t \quad \Rightarrow \quad n \leq 3.522134445 \cdot 10^7$$

$$\begin{aligned}
2.5 \cdot n^2 \leq 10^9 \cdot t \quad &\Rightarrow \quad n \leq \sqrt{4 \cdot 10^8 \cdot t} \\
&\Rightarrow \quad n \leq 2 \cdot 10^4 \cdot \sqrt{t}
\end{aligned}$$

$$\frac{1}{1000} \cdot n^3 \leq 10^9 \cdot t \quad \Rightarrow \quad n \leq (10^{12} \cdot t)^{\frac{1}{3}} = 10^4 \cdot t^{\frac{1}{3}}$$

$$3^n \leq 10^9 \cdot t \quad \Rightarrow \quad n \leq \log_3(10^9 \cdot t) = 9 \log_3(10) + \log_3(t) \approx 18.8 + \log_3(t)$$

Given these relations, if we know the value of t, finding out the maximum size of input is just a matter of solving the equations. In case of t_2 one has to calculate the values separately for different values of t, where as for the other algorithms, we can simply use it as a formula.

	1s	1m = 60s	1h = 3600s
$t_1(n)$	$5 \cdot 10^8$	$3 \cdot 10^{10}$	$1.8 \cdot 10^{12}$
$t_2(n)$	$\approx 3.96 \cdot 10^7$	$\approx 1.94 \cdot 10^9$	$\approx 9.86 \cdot 10^{10}$
$t_3(n)$	20000	$\approx 1.55 \cdot 10^5$	$1.2 \cdot 10^6$
$t_4(n)$	10000	≈ 39149	$\approx 1.53 \cdot 10^5$
$t_4(n)$	≈ 18	≈ 22	≈ 26

Now if we increase the processing power by a factor of 10, it is very evident that the input size can be multiplied by 10 in the case of t_1.

Let's see what happens with t_5. The following was valid when the processing power was 10^9.

$$3^n \leq 10^9 \cdot t \Rightarrow n \leq \log_3(10^9 \cdot t) = 9\log_3(10) + \log_3(t) \approx 18.8 + \log_3(t)$$

When the power is 10^{10}, the relation will change to:

$$3^n \leq 10^{10} \cdot t \Rightarrow n \leq \log_3(10^{10} \cdot t) = 10\log_3(10) + \log_3(t) \approx \log_3(10) + 18.8 + \log_3(t)$$

It is clear that the size of n can be increased by a value of $\log_3(10)$. [1] Now if we continue to analyse the same with other algorithms, we get the following.

t_1	t_2	t_3	t_4	t_5
$\cdot 10$	$\approx \cdot 10$	$\cdot\sqrt{10}$	$\cdot 10^{\frac{1}{3}}$	$+\log_3 10$

Problem 4 (20 Points)

Design iterative and recursive algorithms to compute 2^n. Show that there exists a recursive algorithm which performs better than the iterative naive algorithm.

Solution

Let's try to make two algorithms of which one is iterative and other is recursive.

Iterative algorithm

We multiply 2 n times

Algorithm *PowerOfTwoIterative*(n)
(∗ The iterative algorithm for 2^n ∗)
returnval ← 1
if $n = 0$
  then return *returnval*
while $n > 0$
    $returnval = returnval * 2$
    $n = n - 1$
return *returnval*

It is easily seen that the number of operations needed for this algorithm is $n - 1$.

[1]Note: NOT by a factor

Recursive Algorithm

The main idea of recursive algorithm is from the fact that $2^n = 2^{\frac{n}{2}} * 2^{\frac{n}{2}}$

Algorithm *PowerOfTwoRecursive*(n)
(* The recursive algorithm for 2^n *)
1. **if** $n = 1$
2. **then return** 2
3. **if** n is $EVEN$
4. **then**
5. $PartialResult = PowerOfTwoRecursive(\frac{n}{2})$
6. **return** $PartialResult * PartialResult$
7. **else**
8. **return** $2 * PowerOfTwoRecursive(n - 1)$

Analysis

We can assume that n is greater than one. Let's consider the values of n in a sequence of recursive calls which would happen once $PowerOfTwoRecursive(n)$ is called. It could be:

1. All the values are $EVEN$

   In the case of sequence of all n being $EVEN$, we will be dividing n by 2 in all the calls. The maximum number of this calls can be $\lg n$.

   In every call, we have 2 operations. Hence the number of operations will be $2 \cdot \lg n$.

2. A sequence with alternate ODD and $EVEN$ values of n. In this case, the maximum number of recursive calls will be $2 * \lg n$ since the operations $n = n - 1$ and division by 2 will come alternatively.

   In every call, we have 2 operations. So the number of operations is $2*2 \cdot \lg n = 4 \lg n$.

3. We cannot have a sequence with two consecutive ODD values. Any other sequence will have number of recursive calls varying between $\lg n$ and $2 \cdot \lg n$. So the number of operations will be definitely less than the second case.

The maximum number of operations needed with the recursive algorithm is $4 * \lg n$. As seen in the graph, for any $n > 16$, the recursive algorithm has a better performance than the iterative one.