Algorithms and Data Structures (WS15/16)

Example Solutions for Unit 4

Problem 1

skipped

Problem 2

In class we always chose b to be a constant. Now we will vary b depending on n and show that this would improve the running time.

We show the statement using induction.

Base Case $k = 2$

The statement becomes $T(2^{\binom{2}{2}}) \leq C \cdot 2 \cdot 2^{\binom{3}{2}}$, which is obviously true if we pick $C \geq \frac{T(2)}{16}$

Induction Step

Assume the statement is true for all $m < k$, and we prove it for k.

Let $n = 2^{\binom{k}{2}}$ and $b = 2^{k-1}$. Then we compute $\frac{n}{b}$ to be $2^{\binom{k}{2}-k+1} = 2^{\binom{k-1}{2}}$

Now we use the following equation we have proven in class:

$$T(n) \leq \alpha \cdot b \cdot n + (2b-1)T(\frac{n}{b})$$

Plugging in the values we have chosen for n and b gives us:

$$T(2^{\binom{k}{2}}) \leq \alpha \cdot 2^{k-1} \cdot 2^{\binom{k}{2}} + (2^k - 1)T(2^{\binom{k-1}{2}})$$

Here we use the induction hypothesis for $T(2^{\binom{k-1}{2}})$ and get:

$$T(2^{\binom{k}{2}}) \leq \alpha \cdot 2^{k-1} \cdot 2^{\binom{k}{2}} + (2^k - 1)(C \cdot (k-1) \cdot 2^{\binom{k}{2}}) =$$

$$= \alpha \cdot 2^{k-1} \cdot 2^{\binom{k}{2}} + C \cdot 2^{\binom{k}{2}} \cdot (2^k \cdot k - 2^k - k + 1)$$

$$= \alpha \cdot 2^{k-1} \cdot 2^{\binom{k}{2}} + C \cdot k \cdot 2^{\binom{k+1}{2}} + C \cdot 2^{\binom{k}{2}} \cdot (-2^k - k + 1)$$

In the last step we used that $2^{\binom{k}{2}} \cdot 2^k = 2^{\binom{k+1}{2}}$, the rest was just rearranging the terms.

Therefore in order to prove the statement we need to establish the following:

$$\alpha \cdot 2^{k-1} \cdot 2^{\binom{k}{2}} \leq C \cdot 2^{\binom{k}{2}} \cdot (2^k + k - 1)$$

We put this in the following form:

$$\frac{\alpha}{2} \cdot 2^{\binom{k}{2}+k} \leq C \cdot 2^{\binom{k}{2}+k} + C \cdot (k-1)2^{\binom{k}{2}}$$

It is clear this equation holds for $C \geq \frac{\alpha}{2}$.

Therefore our statement is true for any $C \geq \max\left(\frac{\alpha}{2}, \frac{T(2)}{16}\right)$

Exercises for Unit 6

(3.) multiplying two n digit integers

- we will assume that our base is 10 but all arguments hold for any base b

$$a = \overline{a_{n-1}a_{n-2}\dots a_1a_0}\ ,\quad b = \overline{b_{n-1}b_{n-2}\dots b_1b_0}$$

$$a(x) := a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_1x + a_0$$

$$b(x) := b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \dots + b_1x + b_0$$

- note that all a_i, b_j are one-digit-coeff. and $a = a(10)$, $b = b(10)$
- if we define $c(x) := a(x)\cdot b(x) = c_{2n-2}x^{2n-2} + \dots + c_1x + c_0$, then the number we want to compute is equal to $c(10)$
- we can compute $c(x)$ in $O(n^{1+\varepsilon})$ time $\rightarrow$ the only problem is that the coeff. c_i might not be one-digit numbers

$\Rightarrow$ we cannot just read off what $c(10)$ is, because

$$c(10) \neq \overline{c_{2n-2}c_{2n-1}\dots c_1c_0}$$

- how many digits (in base $b=10$) can c_i have?

$$c_i = \underbrace{\sum_{\substack{k+l=i\\k\geq 0\\l\geq 0}} \underbrace{\underbrace{a_k}_{<10}\cdot\underbrace{b_l}_{<10}}_{<10^2}}_{<n\cdot 10^2}$$

$\Rightarrow O(\log n)$ digits

- we consider how much time we need to compute one digit of $c = c(10)$

$\Rightarrow O(\log n)$ because at most $O(\log n)$ c_i's can influence it and we only need to add up that many one-digit numbers

- since c has $O(n)$ digits, we need in total $O(n \log n)$ time to compute $c = c(10)$ once we have $c(x)$ $\rightarrow$ this is dominated by the time we need for the polynomial multiplication

$$\Rightarrow T(n) = O(n^{1+\varepsilon})$$