Homework A

DD2440 Advanced Algorithms

Jim Holmström F-08 890503-7571

14 oktober 2011

1 Problem 1

$$\text{Given: } a, b \geq 0 \land \neg(a = b = 0) \tag{1}$$

$$\text{Definition of gcd: } \gcd(a, b) = c \iff \arg\max_c \{c|a \land c|b\} \tag{2}$$

1.1 Validate

In order from the top.

$$\texttt{if (b > a) return gcd(b, a)} \tag{3}$$

Sorts the arguments resulting in : $a \geq b$ and trivially holds from (2) with $a \leftrightarrow b$ and using the commutative property of "$\land$"

$$\texttt{else if (b == 0) return a} \tag{4}$$

Acts as base case. $i \neq 0$ from (1) $\gcd(i, 0) = i$ since $c = i$ is the largest $c : (c|i \land c|0)$

$$\texttt{else if (a and b are even) return 2*gcd(a/2, b/2)} \tag{5}$$

Since both a and b are even both must have at least the factor 2 in common. More generally:

$$\gcd(pm, pn) = p \cdot \gcd(m, n) \tag{6}$$

In our case $a = 2m$ and $b = 2n$

$$\texttt{else if (a is even) return gcd(a/2, b)} \tag{7}$$

Since we now that we have passed the above statement we have that b is odd thus 2 cannot be a factor, and we can thus remove the factor 2 from a without influencing the result.

```
else if (b is even) return gcd(a, b/2)
```
(8)

The same way as above but $a \leftrightarrow b$

```
else return gcd(b,a-b)
```
(9)

(3) $\Rightarrow a - b \geq 0$ and (4) $\Rightarrow b \neq 0$ and thus the parameters will at least be within the domain.

$\text{gcd}(b, a-b) = \arg\max_c \{b|c \wedge (a-b)|c\} =$
$\{(a - \underbrace{b}_{c|b})|c \Rightarrow \{a \text{ must have a factor } c, c|c(a/c + b/c)\} \Rightarrow c|a\}$
$= \arg\max_x \{a|c \wedge b|c\} = \text{gcd}(a, b)$ and thus the statement holds.

All the recursive calls holds, has arguments within the domain, the argumentsum is strictly smaller for all calls (except (3) but it can only be called once in a row) $\Rightarrow$ will always land in the basecase (4) and return the correct gcd. $\square$

1.2 Number of recursive calls

In (9) $a, b \in Odd$ so the gcd will be called with $b \in Odd$ and $a - b \in Even$. We now know from above that there can only be a constant number of calls to the statments "not dividing by 2" before "dividing by 2" is called $\Rightarrow$ number of calls $O(log(a) + log(b)) = O(log(ab)$

1.3 Bit complexity

$n = max(n_a, n_b)$ where n_a, n_b is the number of bits in a, b

Assuming the number is represented in the base 2, since we are asked for bit complexity.
Operation-cost:
$2a$ and $a/2$ is $O(n_a)$ "*" and "/" with it's base you can just move all digits (bits in this case) one step in the representation (you "shift" the digits).
$a > b, a - b$ trivially is $O(n)$

The number of recursive calls on the number of bits: $O(log(ab)) = O(log(2^{n_a+n_b}) = O(n_a + n_b))$

Total-bitcomplexity $= \sum \#calls_i * operationcost_i = O(n_a+n_b)*O(n) = \underline{O((n_a + n_b)max(n_a, n_b))}$

2 Problem 2

$$N = 8905037571, a_1 = 123456789, a_2 = 987654321$$

Simple relations used throughout this problem:

$$c \geq 0 \Rightarrow \gcd(c+1, c) = 1 \tag{10}$$

$$\textit{Chinese Remainder Theorem} \tag{11}$$

$$c \cdot d + a \equiv a \ (mod\ d) \tag{12}$$

$$\text{Find a positive } x < N(N+1) : \begin{cases} x \equiv a_1 \ (mod\ N) \\ x \equiv a_2 \ (mod\ N+1) \end{cases} \tag{13}$$

Put x as a smart linear combination of $a_{1:2}$. $x = a_1 b_1 (N+1) + a_2 b_2 N$

(11) $\Rightarrow$

$$\begin{cases} x \equiv a_1 b_1 (N+1) + a_2 b_2 N \ \equiv \{(12)\} \equiv a_1 b_1 (N+1) \ (mod\ N) \\ x \equiv a_1 b_1 (N+1) + a_2 b_2 N \ \equiv \{(12)\} \equiv a_2 b_2 N \ (mod\ N+1) \end{cases} \tag{14}$$

(13) and (14) gives:

$$\begin{cases} a_1 b_1 (N+1) \equiv a_1 \ (mod\ N) \\ a_2 b_2 N \equiv a_2 \ (mod\ N+1) \end{cases} \Rightarrow \begin{cases} b_1 (N+1) \equiv 1 \ (mod\ N) \\ b_2 N \equiv 1 \ (mod\ N+1) \end{cases} \tag{15}$$

x satisfies (13) if $b_{1:2}$ satisfies (15)

$$b_1(N+1) \equiv b_1 N + b_1 \equiv \{(12)\} \equiv b_1 \equiv 1 \ (mod\ N)$$

$$b_2 N \equiv b_2(N+1) - b_2 \equiv \{(12)\} \equiv -b_2 \equiv 1 \ (mod\ N+1) \Rightarrow b_2 \equiv 1 \cdot (-1) \equiv N \ (mod\ N+1)$$

$$\begin{aligned} \therefore x &\equiv a_1(N+1) + a_2 N^2 \equiv \{(\text{using pythons native big-integer support})\} \equiv \\ &\equiv \underline{71603982658724599629} \ (mod\ N(N+1)) \end{aligned}$$

The x found solves (13) and $x < N(N+1)$ $\square$

3 Problem 3

An element is denote by m. With unitcost RAM you can't allocate to much RAM (memory allocated $\leq 2^w = n$) making so for example straight up bucketsort will not do it.

Intending to use radix sort using $\lceil log_2(n+1) \rceil$ at a time. We know that since $m = O(n^{10})$ m will have $O(10 * log(n))$ number of bits. Sorting with $\lceil log_2(n+1) \rceil$ at a time the largest number occuring is $O(n)$ and we can do this sorting step by bucket-sort (assuming you have enough space to allocate, else you just lower the sorting size by a constant, the following arguments will still hold) in $O(n)$ and repeat this 10 times. The resuling sort will take $O(10n)$ and since 10 is a constant we get $O(n)$ $\square$

4 Problem 4

Use a balanced search tree (ex. red-black tree) where each node consists of a tuple of the value i and a bucket b_i consisting of an arraylist. The tree will have m nodes.
Put all elements the their corresponding bucket b_i in the balanced tree, each put is $O(log(m))$ (arraylist insert is $O(1)$) and you do this foreach element $O(n)$ gives us $O(nlog(m))$ for this. Stitch the buckets b_i together in order by repeatly taking the minimum i-bucket from the balanced search tree $O(log(m))$ since you do this m times this will take $O(mlog(m))$
Since the number of unique elements can't exceed to number elements we have $m \leq n$ giving us $O(mlog(m)) \leq O(nlogm)$

Resulting complexity will be $O(nlog(m))$ $\square$

5 Problem 5

Resolution Rule: $\frac{C \vee x \qquad D \vee \neg x}{C \vee D}$ where $C, D \in \bigvee c_i$ in our case $i = 1$ because of the `2-CNF` criteria.
This is need so that $C \vee D \in$ `2-CNF` that is; closed under resolution. (Note. this doesn't hold for `3-CNF` or higher making that problem much harder) The main idea in the proof is that taking all valid combinations of the $2n$ variables in the start expression will result in maximally $(2n)^2 = O(n^2)$ number of valid resoluted new clauses and since these are all possible `2-CNF` from this statement can't be further resoluted (not without resulting in duplications since they are already represented). Foreach resolution you check for the resolution clause to return either () or $(x \vee x)$ (where x can be negated variable). () will result in `false` termination and $(x \vee x)$-clause is resulting in "$x == 1$" for the orignal statement to hold. Each resolution and resolution check will take $O(1)$ but for safety reason one can say that it's polynomal and still fullfill the criterias as a solution for this

assignment. The same goes for the container of clauses, answer etc, as long as you go with a structure which has polynomal access time.

All the operators are of polynomal order and since polynoms are closed under both addition and multiplication ($n^2 Poly(n) = Poly(n)$) we have a resulting polynomal running time $\square$