SSN COLLEGE OF ENGINEERING
B.E. (Computer Science and Engineering) Semester 4
Unit Test: 1 (28 January 2013)

Time: 8.00–9.30 | **CS2251 Design and Analysis of Algorithms** | Max marks: 50

Part A

$5 \times 1 = 5$

Answer *all* questions (true or false).

1. $101n + 5 \in O(n^2)$ true
2. $100n + 5 \in \Omega(n^2)$ false
3. $n \in o(6n)$ true
4. $f(n) \in \Theta(g(n)) \equiv g(n) \in \Theta(f(n))$ true
5. $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$ true

Part B

$10 \times 2 = 20$

Answer *all* questions.

6. List the important qualities of algorithms.
   precise, executable, finite
7. Compare the order of growth of $n!$ and $2n$.
8. Find the order of growth of the function $10n^2 + 4n + 2$ with suitable values for c and n_0.
   $O(n^2),\ c = 16, n_0 = 0$
9. If $f(x) = \dfrac{x^3}{2}$ and $g(x) = 37x^2 + 120x + 17$, show that $g = O(f)$, but $f \neq O(g)$.

   $$g(x) = 37x^2 + 120x + 17 \leq 174x^3 = 348\frac{x^3}{2} = 348f(x)$$

   $$f(x)\frac{x^3}{2} \leq cx^2 \Rightarrow x \leq 2c$$

10. If $T(n) = \frac{n^2}{2}$, then what is $T(2n)$?

    $$T(2n) = \frac{(2n)^2}{2} = \frac{4n^2}{2} = 2n^2$$

11. Find the order of growth of the sum $\sum_1^n (i^2+1)^2$

$$
\begin{aligned}
&\sum_1^n (i^2+1)^2 \\
&= \sum_1^n i^4 + 2i^2 + 1 \\
&= \sum_1^n i^4 + 2\sum_1^n i^2 + \sum_1^n 1 \\
&= \left\langle \sum_{i=1}^n i^k \approx \frac{1}{k+1} n^{k+1} \right\rangle \\
&\frac{1}{5}n^5 + \frac{1}{3}n^3 + n \\
&= O(n^5)
\end{aligned}
$$

12. How many times the body of the inner loop is executed?

    for $i \leftarrow 1$ **to** m
        for $j \leftarrow 1$ **to** n
            $c[i, j] \leftarrow a[i, j] + b[i, j]$
        end
    end

    mn

13. Find the time complexity of sum(a) using step count method.

    Algorithm: sum(a)

    $s \leftarrow 0$, $i \leftarrow 1$
    while $i \le n$ **do**
        $s \leftarrow s + a[i]$
        $i \leftarrow i + 2$
    end

14. Solve the recurrence relation:

$$
T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + 1 & \text{if } n > 1 \end{cases}
$$

    $T(n) = n$

15. Arrange these functions in increasing order of asymptotic growth: $c^n, n\log n, n^2, \log n, n^3$
    $\log n, n\log n, n^2, n^3, c^n$

Part C $\qquad 5 \times 5 = 25$

Answer any *five* questions.

16. Consider two algorithms A and B for solving the same problem running on two machines 1 and 2. Machine 1 executes 10^9 (1 billion) instructions per second, and machine 2 executes 10^7 (10 million) instructions per second. Algorithm A requires $2n^2$ instructions and runs on machine 1; algorithm B requires $50n \log_{10} n$ instructions and runs on machine 2.

(a) Calculate the running time of the two algorithms for inputs of sizes 100, 1000, 10000.

	$\frac{2n^2}{10^9}$	$\frac{50n \log n}{10^7}$
100	$\frac{2 \times 100 \times 100}{10^9} = \frac{2}{10^5}$	$\frac{50 \times 100 \times 2}{10^7} = \frac{1}{10^3}$
1000	$\frac{2 \times 1000 \times 1000}{10^9} = \frac{2}{10^3}$	$\frac{50 \times 1000 \times 3}{10^7} = \frac{15}{10^3}$
10,000	$\frac{2 \times 10000 \times 10000}{10^9} = \frac{2}{10}$	$\frac{50 \times 10000 \times 4}{10^7} = \frac{2}{10}$

(b) Which is better — algorithm A on machine 1, or algorithm B on machine 2? Why?
Algorithm B. After 10000, its running time is smaller. It grows slowly.

17. (a) Write and solve the recurrence relation for computing factorial of a number.

Algorithm: fact (n)

if `n = 0` **then** 1 **else** n * (fact n-1)

$$T(n) = \begin{cases} 1 & \text{if } n = 0 \\ T(n-1) + 1 & \text{if } n > 0 \end{cases}$$

$$\begin{aligned} T(n) &= 1 + T(n-1) \\ &= \langle T(n-1) = 1 + T(n-2) \rangle \\ & \quad\ 1 + 1 + T(n-2) \\ &= 2 + T(n-2) \\ & \quad\ \ldots \\ &= n + T(0) \\ &= n + 1 \\ &= O(n) \end{aligned}$$

(b) Write an iterative algorithm for the same, and calculate the running time.

Algorithm: fact (n)

f, i ← 1, 0

`-- f = i!`

until `i = n` **do**

| f, i ← f * (i+1), i+1

end

Loop iterates n times. Basic step: multiply. $T(n) = O(n)$

18. Define big-Oh, big-Omega, and big-Theta notations. Give an example for each.

    Big-Oh: definition, graph, example

    $$f(n) = O(g(n)) \equiv \quad \text{There exist constants } c \text{ and } n_0 \text{ such that} \\ f(n) \le cg(n) \text{ for all } n \ge n_0$$

    Big-Omega: definition, graph, example

    $$f(n) = \Omega(g(n)) \equiv \quad \text{There exist constants } c \text{ and } n_0 \text{ such that} \\ f(n) \ge cg(n) \text{ for all } n \ge n_0$$

    Big-Theta: definition, graph, example

    $$f(n) = \Theta(g(n)) \equiv \quad \text{There exist constants } c_1, c_2, n_0 \text{ such that} \\ c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0$$

19. Analyse the best-case, the worst-case, and the average-case running times of the algorithm for an array a of size n?

    Algorithm: LinearSearch (a, x)
    $i \leftarrow 0$
    while $i \ne n$ **and** $a[i] \ne x$ **do** $i \leftarrow i + 1$
    return i

    Worst-case: x is not in a. The loop terminates by $i = n$. Loop iterates n times. $T(n) = n$
    Best-case: $x = a[0]$. The loop terminates by $a[0] = x$. Loop does not iterate at all. $T(n) = 0$.
    Average-case: x is probable in any position from 0 to $n - 1$ with probability $\frac{1}{n}$. The loop, on average, iterates $(1 + 2 + \ldots + (n - 1))/n = (n - 1)n/2$ times. $T(n) = (n - 1)/2$.

20. (a) Solve the recurrence relation

    $$T(n) = \begin{cases} 2T\left(\dfrac{n}{2}\right) + n & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$$

Let $n = 2^h$. Then $h = \log_2 n$

$$\begin{aligned}
T(n) &= n + 2T\left(\frac{n}{2}\right) \\
&= \left\langle T\left(\frac{n}{2}\right) = \frac{n}{2} + T\left(\frac{n}{2^2}\right)\right\rangle \\
&\quad n + 2\left[\frac{n}{2} + 2T\left(\frac{n}{2^2}\right)\right] \\
&= n + n + 2^2 T\left(\frac{n}{2^2}\right) \\
&= 2n + 2^2 T\left(\frac{n}{2^2}\right) \\
&\ldots \\
&= nh + 2^h T\left(\frac{n}{2^h}\right) \\
&= \left\langle n = 2^h \right\rangle \\
&\quad nh + 2^h T(1) \\
&= \left\langle h = \log n, n = 2^h, T(1) = 1 \right\rangle \\
&= n \log_2 n + n \\
&= O(n \log n)
\end{aligned}$$

(b) Solve it using Master Theorem.
In $T(n) = aT\left(\frac{n}{b}\right) + n^c$,

$$\begin{aligned}
&a = 2 \\
&b = 2, c = 1, b^c = 2^1 = 2 \\
&a = b^c
\end{aligned}$$

By Master Theorem

$$T(n) = \Theta(n^c \log n) = \Theta(n^1 \log n)$$

(c) What is the order of growth of $T(n)$?

21. A binary search is given as input a sorted array a[1..n] and the output is an index i partitioning the array into two subarrays, a[1..i] $<$ k and $k \leq$ a[i+1..n] where k is the search key. It maintains three subarrays a[1..i-1] $<$ k, a[i..j], and $k \leq$ a[j+1..n] where the search is restricted to a[i..j]. When a[i..j] becomes empty, the search terminates.

    (i) Choose an index mid in the interval i..j that is close to the middle of the interval. i..j is not empty.

$$
\begin{aligned}
& i \leq j \\
\equiv\ & 2i \leq i+j \leq 2j \\
\equiv\ & i \leq \frac{i+j}{2} \leq j \\
\equiv\ & \left\langle m = \frac{i+j}{2} \right\rangle \\
\equiv\ & i \leq m \leq j \\
\equiv\ & i \leq m, m \leq j \\
\equiv\ & \langle \lfloor m \rfloor \leq m < \lfloor m \rfloor + 1 \rangle \\
\equiv\ & i \leq m < \lfloor m \rfloor + 1, \lfloor m \rfloor \leq m \leq j \\
\equiv\ & i < \lfloor m \rfloor + 1, \lfloor m \rfloor \leq j \\
\equiv\ & i \leq \lfloor m \rfloor, \lfloor m \rfloor \leq j \\
\equiv\ & i \leq \lfloor m \rfloor \leq j
\end{aligned}
$$

(ii) How is the search interval i..j reduced depending on whether a[m] $<$ k or k $\leq$ a[m]?

$$
\begin{aligned}
& [m] < k \\
\equiv\ & [1..m] < k \\
\equiv\ & \langle \text{invariant} : [1..i-1] < k, \text{progress} \rangle \\
& [1..i-1] = [1..m] \\
\equiv\ & [i..j] = [m+1..j]
\end{aligned}
\qquad
\begin{aligned}
& k \leq [m] \\
\equiv\ & k \leq [m..n] \\
\equiv\ & \langle \text{invariant} : k \leq [j+1..n], \text{progress} \rangle \\
& [j+1..n] = [m..n] \\
\equiv\ & [i..j] = [i..m-1]
\end{aligned}
$$

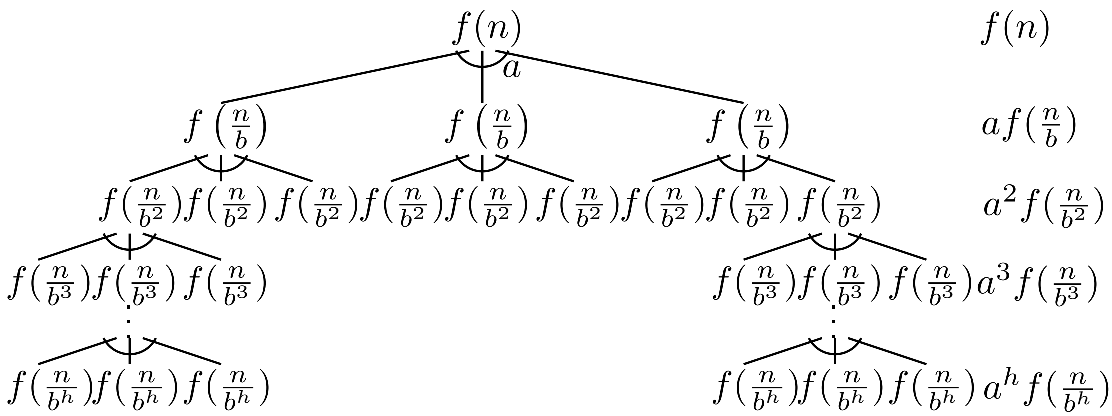
(iii) What is the key operation of the algorithm? Write the recurrence equation for the running time of the algorithm.
Key step: compare.

$$
T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T\left(\dfrac{n}{2}\right) + 1 & \text{if } n > 1 \end{cases}
$$

22. A divide-and-conquer algorithm is called with an input of size n. It does a work of $f(n)$; it divides the problem into a subproblems and to each it passes an input of size $\frac{n}{b}$. The running time of the algorithm is given by

$$
T(n) = aT\left(\frac{n}{b}\right) + f(n)
$$

(i) Draw the recursion tree for the algorithm and find an expression for the running time $T(n)$ of the algorithm, summing the running times of all the recursive calls of the algorithm.



$$
\begin{aligned}
&T(n)\\
&= f(n) + af\left(\frac{n}{b}\right) + a^2 f\left(\frac{n}{b^2}\right) + a^3 f\left(\frac{n}{b^3}\right) + \ldots + a^h f\left(\frac{n}{b^h}\right)\\
&= \sum_{i=0}^{h} a^i f\left(\frac{n}{b^i}\right)
\end{aligned}
$$

where $h = \log_b n$ is the height of the tree.

(ii) Assuming $f(n) = n^c$, rewrite the expression for the running time $T(n)$. In the divide and conquer recurrences, if $f(n)$ is a polynomial n^c,

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{h} a^i \left(\frac{n}{b^i}\right)^c\\
&= \sum_{i=0}^{h} a^i \left(\frac{n^c}{b^{ic}}\right)\\
&= \sum_{i=0}^{h} \left(\frac{a^i}{b^{ic}}\right) n^c\\
&= n^c \sum_{i=0}^{h} \left(\frac{a}{b^c}\right)^i
\end{aligned}
$$

Good luck is another name for tenacity of purpose.
(Ralph Waldo Emerson)

Prepared by | Reviewed by

S Kavitha, R S Milton | HoD, CSE