Master Theorem: Practice Problems and Solutions

Master Theorem

The Master Theorem applies to recurrences of the following form:

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is an asymptotically positive function.

There are 3 cases:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a} \log^k n)$ with[1] $k \geq 0$, then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ with $\epsilon > 0$, and $f(n)$ satisfies the regularity condition, then $T(n) = \Theta(f(n))$.
   Regularity condition: $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n.

Practice Problems

For each of the following recurrences, give an expression for the runtime $T(n)$ if the recurrence can be solved with the Master Theorem. Otherwise, indicate that the Master Theorem does not apply.

1. $T(n) = 3T(n/2) + n^2$

2. $T(n) = 4T(n/2) + n^2$

3. $T(n) = T(n/2) + 2^n$

4. $T(n) = 2^n T(n/2) + n^n$

5. $T(n) = 16T(n/4) + n$

6. $T(n) = 2T(n/2) + n \log n$

[1]most of the time, $k = 0$

7. $T(n) = 2T(n/2) + n/\log n$

8. $T(n) = 2T(n/4) + n^{0.51}$

9. $T(n) = 0.5T(n/2) + 1/n$

10. $T(n) = 16T(n/4) + n!$

11. $T(n) = \sqrt{2}T(n/2) + \log n$

12. $T(n) = 3T(n/2) + n$

13. $T(n) = 3T(n/3) + \sqrt{n}$

14. $T(n) = 4T(n/2) + cn$

15. $T(n) = 3T(n/4) + n\log n$

16. $T(n) = 3T(n/3) + n/2$

17. $T(n) = 6T(n/3) + n^2\log n$

18. $T(n) = 4T(n/2) + n/\log n$

19. $T(n) = 64T(n/8) - n^2\log n$

20. $T(n) = 7T(n/3) + n^2$

21. $T(n) = 4T(n/2) + \log n$

22. $T(n) = T(n/2) + n(2 - \cos n)$

Solutions

1. $T(n) = 3T(n/2) + n^2 \implies T(n) = \Theta(n^2)$ (Case 3)

2. $T(n) = 4T(n/2) + n^2 \implies T(n) = \Theta(n^2 \log n)$ (Case 2)

3. $T(n) = T(n/2) + 2^n \implies \Theta(2^n)$ (Case 3)

4. $T(n) = 2^n T(n/2) + n^n \implies$ Does not apply (a is not constant)

5. $T(n) = 16T(n/4) + n \implies T(n) = \Theta(n^2)$ (Case 1)

6. $T(n) = 2T(n/2) + n \log n \implies T(n) = n \log^2 n$ (Case 2)

7. $T(n) = 2T(n/2) + n/\log n \implies$ Does not apply (non-polynomial difference between $f(n)$ and $n^{\log_b a}$)

8. $T(n) = 2T(n/4) + n^{0.51} \implies T(n) = \Theta(n^{0.51})$ (Case 3)

9. $T(n) = 0.5T(n/2) + 1/n \implies$ Does not apply ($a < 1$)

10. $T(n) = 16T(n/4) + n! \implies T(n) = \Theta(n!)$ (Case 3)

11. $T(n) = \sqrt{2}T(n/2) + \log n \implies T(n) = \Theta(\sqrt{n})$ (Case 1)

12. $T(n) = 3T(n/2) + n \implies T(n) = \Theta(n^{\lg 3})$ (Case 1)

13. $T(n) = 3T(n/3) + \sqrt{n} \implies T(n) = \Theta(n)$ (Case 1)

14. $T(n) = 4T(n/2) + cn \implies T(n) = \Theta(n^2)$ (Case 1)

15. $T(n) = 3T(n/4) + n \log n \implies T(n) = \Theta(n \log n)$ (Case 3)

16. $T(n) = 3T(n/3) + n/2 \implies T(n) = \Theta(n \log n)$ (Case 2)

17. $T(n) = 6T(n/3) + n^2 \log n \implies T(n) = \Theta(n^2 \log n)$ (Case 3)

18. $T(n) = 4T(n/2) + n/\log n \implies T(n) = \Theta(n^2)$ (Case 1)

19. $T(n) = 64T(n/8) - n^2 \log n \implies$ Does not apply ($f(n)$ is not positive)

20. $T(n) = 7T(n/3) + n^2 \implies T(n) = \Theta(n^2)$ (Case 3)

21. $T(n) = 4T(n/2) + \log n \implies T(n) = \Theta(n^2)$ (Case 1)

22. $T(n) = T(n/2) + n(2 - \cos n) \implies$ Does not apply. We are in Case 3, but the regularity condition is violated. (Consider $n = 2\pi k$, where k is odd and arbitrarily large. For any such choice of n, you can show that $c \geq 3/2$, thereby violating the regularity condition.)

Recurrence Relations

Introduction

Determining the running time of a recursive algorithm often requires one to determine the big-O growth of a function $T(n)$ that is defined in terms of a *recurrence relation*. Recall that a **recurrene relation** for a sequence of numbers is simply an equation that provides the value of the n th number in terms of an algebraic expression involving n and one or more of the previous numbers of the sequence. The most common recurrence relation we will encounter in this course is the **uniform divide-and-conquer recurrence relation**, or **uniform recurrence** for short.

Uniform Divide-and-Conquer Recurrence Relation: one of the form

$$T(n) = aT(n/b) + f(n),$$

where $a > 0$ and $b > 1$ are integer constants. This equation is explained as follows.

$T(n)$**:** $T(n)$ denotes the number of steps required by some divide-and-conquer algorithm $\mathcal{A}$ on a problem instance having size n.

Divide $\mathcal{A}$ divides original problem instance into a subproblem instances.

Conquer Each subproblem instance has size n/b and hence is conquered in $T(n/b)$ steps by making a recursive call to $\mathcal{A}$.

Combine $f(n)$ represents the number of steps needed to both divide the original problem instance and combine the a solutions into a final solution for the original problem instance.

For example,

$$T(n) = 7T(n/2) + n^2,$$

is a uniform divide-and-conquer recurrence with $a = 7$, $b = 2$, and $f(n) = n^2$.

It should be emphasized that not every divide-and-conquer algorithm produces a uniform divide-and-conquer recurrence. For example, the *Median-of-Five Find Statistic* algorithm described in the next lecture produces the recurrence

$$T(n) = T(n/5) + T(7n/10) + an,$$

where $a > 0$ is a constant. We refer to such recurrences as **non-uniform divide-and-conquer recurrences**. They are non-uniform in the sense that the created subproblem instances may not all have the same size.

The complexity analysis of a divide-and-conquer algorithm often reduces to determining the big-O growth of a solution $T(n)$ to a divide-and-conquer recurrence. In this lecture we examine three different ways of solving such recurrences, which are summarized as follows.

Master Theorem The **Master Theorem** provides a solution $T(n)$ to a uniform recurrence, provided a, b, and $f(n)$ satisfy certain conditions.

Recursion Tree Given a recursive algorithm, a **recursion tree** is a tree whose nodes are in one-to-one correspondence with the subproblem instances that are created by the algorithm at each depth of the recursion. Moreover, each node of the tree is labeled with either i) the number of algorithm steps that are needed to divide up the corresponding subproblem instance into further subproblems and to combine their solutions to obtain a solution to the subproblem, or ii) the number of steps needed to directly solve the subproblem instance if it represents a base case (in this case the node is a leaf of the recursion tree).

Substitution Method The **substitution method** uses mathematical induction to prove that some candidate function $T(n)$ is a solution to a given divide-and-conquer recurrence. The candidate solution is usually obtained by making an educated guess, or by analyzing the associated recursion tree.

The Master theorem and substitution method represent *proof methods*, meaning that the application of either method will yield a solution beyond doubt. On the other hand, the recursion-tree method represents an *exploratory method* whose solution must be further verified using another method, such as the substitution method.

Recursion Trees and The Master Theorem

Master Theorem. Let $a \geq 1$ and $b > 1$ be constants, $f(n)$ a function, and $T(n)$ be defined on the nonnegative integers by

$$T(n) = aT(n/b) + f(n).$$

Then $T(n)$ can be bounded asymptotically as follows.

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$, then $T(n) = \Theta(f(n))$.

We prove a relaxed version of the Master theorem by assuming that n is always a power of b, instead of always being an integer. The Master Theorem then follows from this theorem by more analysis involving floors and ceilings, which shows that their inclusion does not affect the order of growth of $T(n)$.

Lemma 1. Let $a \geq 1$ and $b > 1$ be constants, and let $f(n)$ be a nonnegative function defined on exact powers of b. Define $T(n)$ on exact powers of b by the recurrence relation

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ aT(n/b) + f(n) & \text{if } n = b^i \end{cases}$$

for some positive integer i. Then

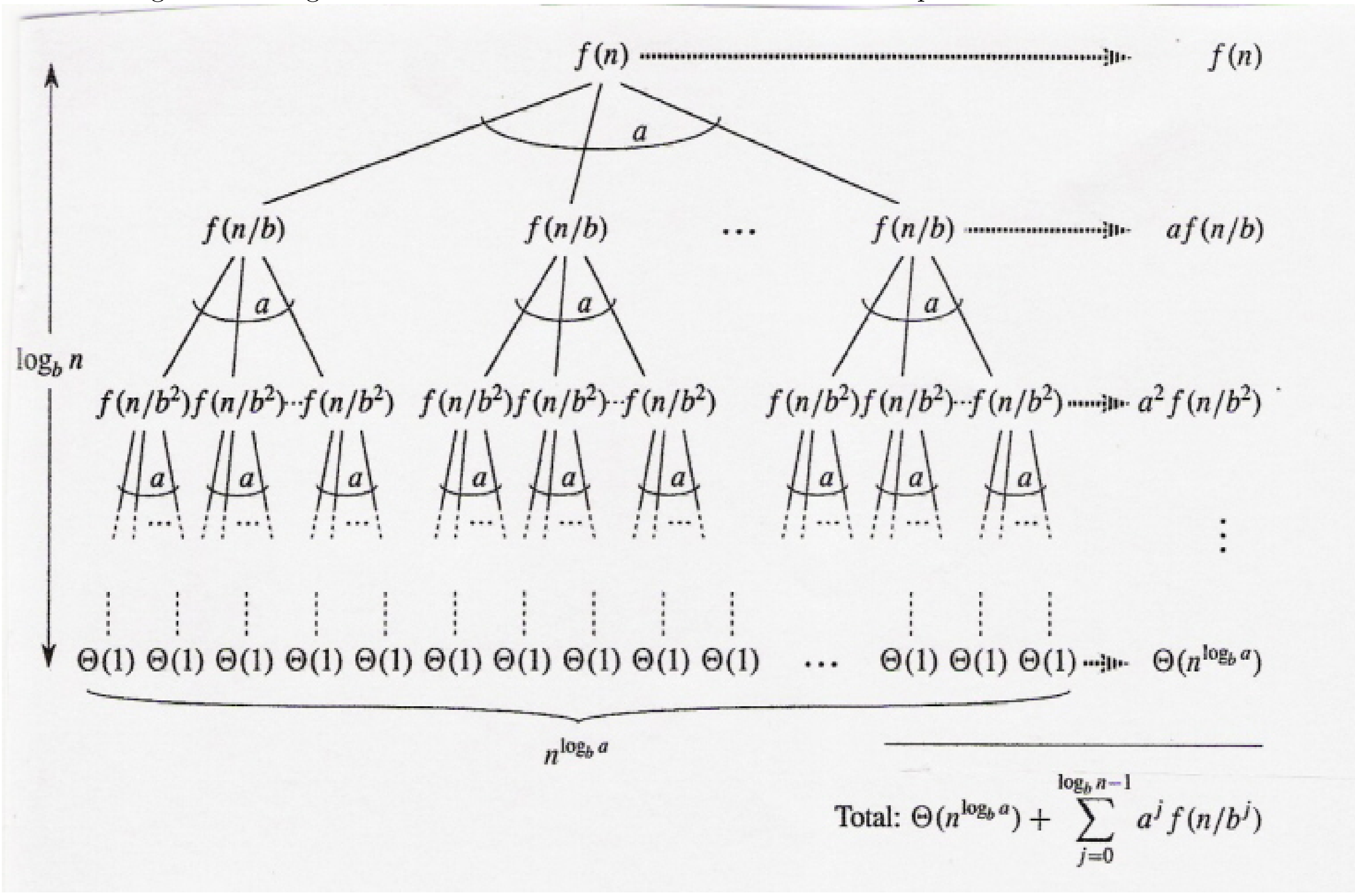
$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j).$$

Proof of Lemma 1. Notice that, since n is a power of b, there are $\log_b n + 1$ levels of recursion: $0, 1, \ldots, \log_b n$. Moreover, level j, $0 \leq j \leq \log_b n - 1$ contributes a total of $a^j f(n/b^j)$ to the total value of $T(n)$, while level $\log_b n$ contributes $\Theta(1) \cdot a^{\log_b n} = \Theta(n^{log_b a})$ (see the general recursion-tree in Figure 1 below). Hence,

$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j).$$

QED

Figure 1: The general recursion tree for uniform divide-and-conquer recurrences



Lemma 2. Let

$$g(n) = \sum_{j=0}^{\log_b n-1} a^j f(n/b^j),$$

where a, b, and $f(n)$ are defined as in Lemma 1. Then

1. If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$, then $g(n) = O(n^{\log_b a})$
2. If $f(n) = \Theta(n^{\log_b a})$ then $g(n) = \Theta(n^{\log_b a} \cdot \log n)$
3. If $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \le cf(n)$ for some constant $c < 1$, then $g(n) = \Theta(f(n))$

Proof of Lemma 2. This is just an exercise in summing the geometric series

$$\sum_{j=0}^{\log_b n-1} a^j f(n/b^j)$$

under varying assumptions about $f(n)$. In Case 1, replace $f(n/b^j)$ with $(\frac{n}{b^j})^{\log_b a-\epsilon}$. In Case 2, replace $f(n/b^j)$ with $(\frac{n}{b^j})^{\log_b a}$. And in Case 3, replace $a^j f(n/b^j)$ with $c^j f(n)$. In all cases, the formula

$$\sum_{j=0}^{k} r^j = \frac{r^{k+1}-1}{r-1}$$

will prove helpful. QED

Example 1. Prove Case 1 of Lemma 2.

Once Lemma 2 is proved, the proof of the relaxed version of the Master Theorem becomes readily available.

Example 2. Finish the proof of Case 1 of the relaxed version of the Master Theorem.

Example 3. Determine the order of growth of the solutions to the following recurrence relations.

1. $T(n) = 16T(n/4) + n$
2. $T(n) = T(n/5) + 20$
3. $T(n) = 3T(n/4) + n \log n$
4. $T(n) = 2T(n/2) + n \log n$

Example 4. Use a recursion tree to estimate the big-O growth of $T(n)$ if it satisfies $T(n) = T(n-2) + n$.

Example 5. Use a recursion tree to estimate the big-O growth of $T(n)$ if it satisfies $T(n) = T(n/2) + 2T(n/4) + n$.

Substitution Method

The substitution method is an inductive method for proving the big-O growth of a function $T(n)$ that satisfies some divide-and-conquer recurrence. It requires that we already have a candidate function $g(n)$ for representing the growth of $T(n)$. For example, suppose we desire to show that $T(n) = \mathrm{O}(g(n))$. Then we peroform the following two steps.

Inductive Assumption Assume that $T(k) \le Cg(k)$, for all $k < n$ and for some constant $C > 0$.

Prove Inductive Step Show that $T(n) \le Cg(n)$. Do this by replacing any term $aT(n/b)$ of the recurrence with $aCg(n/b)$, and show that the resulting non-recurrence expression is bounded above by $Cg(n)$.

Notice that there is no basis step to the above proof method. This is because we only care about the growth of $T(n)$ for sufficiently large n.

Example 6a. Show that if $T(n) = 2T(n/2) + 3n$, then $T(n) = \mathrm{O}(n \log n)$.

Example 6b. Similarly, show that $T(n) = \Omega(n \log n)$, where $T(n)$ satisfies the recurrence from Example 6a.

Sometimes it is necessary to add lesser-degree terms to the inductive assumption. For example, instead of $T(k) \le Ck^2$, one may instead use $T(k) \le Ck^2 + Dk + E$, for some constants $C, D, E > 0$. This is sometimes necessary in order to cancel lesser-degree terms that occur in the recurrence. The following example illustrates this.

Example 7. Show that if $T(n) = 2T(n/2) + n \log n$, then $T(n) = \mathrm{O}(n \log^2 n)$.

Example 8. Prove that if $T(n) = 2T(n/2) + 7$ then $T(n) = \mathrm{O}(n)$.

Exercises

1. Use the Master Theorem to give tight asymptotic bounds for the following recurrences.

   a. $T(n) = 2T(n/4) + 1$
   b. $T(n) = 2T(n/4) + \sqrt{n}$
   c. $T(n) = 2T(n/4) + n$
   d. $T(n) = 2T(n/4) + n^2$

2. Use the Master Theorem to show that the solution to the binary-search recurrence $T(n) = T(n/2) + a$, where $a > 0$ is a constant, is $T(n) = \Theta(\log n)$.

3. Use the Master Theorem to determine the big-O growth of $T(n)$ if it satisfies the recurrence $T(n) = 3T(n/2) + n$.

4. Explain why the Master Theorem cannot be applied to the recurrence $T(n) = 4T(n/2)+n^2 \log n$.

5. Use the Master Equation to estimate the growth of $T(n)$ which satisfies the recurrence from Exercise 4. Note: you should use the substitution method to verify that the estimate is in fact the exact big-O growth of $T(n)$.

6. Solve the recurrence $T(n) = 2T(\sqrt{n}) + \log n$. Hint: Let $S(k) = T(2^k)$ and write a divide-and-conquer recurrence for $S(k)$.

7. Use a recursion tree to show that the solution of $T(n) = T(n-1) + n$ is $T(n) = O(n^2)$.

8. Use a recursion tree to solve the recurrence $T(n) = T(n-2) + n^2$; providing a tight upper and lower bound.

9. Use a recursion tree to estimate the big-O growth of $T(n)$ which satisifies the recurrence $T(n) = 2T(n-1) + 1$.

10. Use a recursion tree to estimate the big-O growth of $T(n)$ which satisifies the recurrence $T(n) = T(n/2) + n$ is $T(n) = \Theta(n)$. Verify your answer using the Master theorem.

11. Use a recursion tree to estimate the big-O growth of $T(n)$ which satisifies the recurrence $T(n) = T(n/2) + n^2$. Verify your answer using the Master theorem.

12. Argue that the solution to the recurrence $T(n) = T(n/3) + T(2n/3) + an$, where $a > 0$ is a constant, is $T(n) = \Theta(n \log n)$, by using an appropriate recursion tree.

13. Suppose $T(n)$ and $g(n)$ are positive functions that satisfy $T(n) \leq Cg(n)$ for all $n \geq k$, and some constant $C > 0$. Prove that There is a constant $C' > 0$ for which $T(n) \leq C'g(n)$, for *all* $n \geq 0$.

14. Use the substitution method to prove that $T(n) = 2T(n/2)+an$, $a > 0$ a constant, has solution $T(n) = \Theta(n \log n)$.

15. Use the substitution method to prove that $T(n) = 4T(n/2) + n$ has solution $T(n) = O(n^2)$.

16. Use the substitution method to prove that $T(n) = T(an)+T(bn)+n$ has solution $T(n) = O(n)$, where a and b are positive constants, with $a + b < 1$.

Exercise Solutions

1. Use the Master Theorem to give tight asymptotic bounds for the following recurrences.

   a. Case 1: $T(n) = \Theta(\sqrt{n})$

   b. Case 2: $T(n) = \Theta(\sqrt{n} \log n)$

   c. Case 3: $T(n) = \Theta(n)$

   d. Case 3: $T(n) = \Theta(n^2)$

2. Use Case 2 of the Master Theorem: $T(n) = \Theta(\log n)$.

3. Use the Master Theorem to determine the big-O growth of $T(n)$ if it satisfies $T(n) = 3T(n/2) + n$. Use case 1. $T(n) = \Theta(n^{\log 3})$.

4. $n^{\log_b a} = n^2$ and $f(n) = n^2 \log n = \omega(n^2)$. Hence, Cases 1 and 2 do not apply. Moreover, $n^2 \log n \neq \Omega(n^{2+\epsilon})$, for *all* $\epsilon > 0$. Therefore, Case 3 cannot be applied.

5. From the Master Equation we have

$$\sum_{j=0}^{\log_b n-1} a^j f(n/b^j) = \sum_{j=0}^{\log_2 n-1} 4^j (\frac{n}{2^j})^2 \log(\frac{n}{2^j}) =$$

$$n^2 \sum_{j=0}^{\log_2 n-1} (\log n - j) = n^2(\log^2 n - \log n(\log n - 1)/2) = \Theta(n^2 \log^2 n).$$

   Also, $n^{\log_b a} = n^2$. Therefore, when n is a power of 2, $T(n) = \Theta(n^2 \log^2 n)$.

6. Letting $S(k) = T(2^k)$, we then have the divide-and-conquer recurrence

$$S(k) = T(2^k) = 2T((2^k)^{1/2}) + \log 2^k = 2T(2^{k/2}) + k = 2S(k/2) + k,$$

   which, by the Master Theorem, yields $S(k) = \Theta(k \log k)$. Finally, letting $n = 2^k$, we have $T(n) = \Theta(\log n \log(\log n))$.

7. The recursion tree consists of a single branch of length n, where the work done at depth i is $n - i$. Hence, $T(n) = O(\sum_{i=0}^{n}(n-i)) = O(n^2)$.

8. The recursion tree consists of a single branch of length $n/2$, where the work done at depth i is $(n-2i)^2$. Hence,

$$T(n) = \Theta(\sum_{i=0}^{n/2}(n-2i)^2) = \Theta(n^3).$$

9. The recursion tree is a perfect binary tree having depth n. Thus, it has $2^{n+1} - 1$ nodes. Moreover, since the work at each node is always 1, we see that $T(n) = \Theta(2^n)$.

10. Assuming n is a power of 2, the total work from the recursion tree is

$$n + n/2 + n/4 + \cdots + n/2^{logn}.$$

    Factoring out n and adding the geometric series yields a sum that is $\Theta(n)$.

11. The recursion tree has a single branch, where the work done at depth i is $(n/2^i)^2 = n^2/4^i$. Moreover, since $\sum_{i=0}^{\infty} 1/4^i = 4/3$, we see that $T(n) = \Theta(n^2)$.

12. The recursion tree remains perfect all the way down to depth $\lfloor \log_3 n \rfloor$. Moreover, for each depth $i = 0, 1, \ldots, \lfloor \log_3 n \rfloor$, the work done at that depth always adds to n. Hence, the total work (i.e. $T(n)$) must be $\Omega(n \log n)$. Also, the longest branch has a depth not exceeding $\lfloor \log_{\frac{3}{2}} n \rfloor$. Moreover, the amount of work at each depth does not exceed n. Hence $T(n) = \mathrm{O}(n \log n)$. Therefore, $T(n) = \Theta(n \log n)$.

13. Let C_i, $i = 0, 1, \ldots, k-1$, be a constant for which $T(i) \leq C_i g(i)$. Since $f(i)$ and $g(i)$ are both positive numbers, we know that such a C_i exists. Then choose $C' = \max(C_0, \ldots, C_{k-1}, C)$. Then $T(n) \leq C' g(n)$, for all $n \geq 0$.

14. Assume $T(k) \leq ck \log k$, for all $k < n$. Then

$$T(n) = 2T(n/2) + an \leq 2c(n/2)\log(n/2) + an = cn \log n - cn + an \leq cn \log n$$

iff $c \geq a$. Therefore by induction, $T(n) = \mathrm{O}(n \log n)$. $T(n) = \Omega(n \log n)$ is proved similarly.

15. The hypothesis $T(k) \leq Ck^2$, for all $k < n$ is not sufficient, since it leads to the inequality $n \leq 0$. Using $T(k) \leq Ck^2 + Dk$ yields

$$T(n) = 4T(n/2) + n \leq 4(C(n/2)^2 + Dn/2) + n = Cn^2 + 2Dn + n \leq Cn^2 + Dn \Leftrightarrow$$

$$Dn + n \leq 0 \Leftrightarrow D \leq -1.$$

Therefore, the inequality is established so long as we choose $D \leq -1$, and $C > 0$.

16. Assume $T(k) \leq Ck$, for all $k < n$ and some constant $C > 0$. Then

$$T(n) = T(an) + T(bn) + n \leq Can + Cbn + n = Cn(a+b) + n \leq Cn \Leftrightarrow$$

$$Cn(1 - a - b) \geq n \Leftrightarrow C \geq 1/(1 - a - b).$$

Since $a+b < 1$, we have $1-a-b > 0$, and thus the inequality holds, so long as $C \geq 1/(1-a-b)$.

POSTAL Book Package 2021

Computer Science & IT

Objective Practice Sets

Algorithms

Contents

MADE EASY Publications

2 CHAPTER Recurrence Relations

Q.1 Let $T(n) = [n(\log(n^3) - \log n) + \log n]n + \log n$
Find complexity of $T(n)$.
(a) $O(n \log n)$ (b) $O(n^2)$
(c) $O(n^2 \log n)$ (d) $O(n^3)$

Q.2 Let T_n be the recurrence relation is defined as follows:

$$T_n = \begin{cases} 0, & n = 0 \\ T_{n-1} + n, & n \geq 1 \end{cases}$$

Find the value of T_n?
(a) n (b) n^2
(c) $\frac{n^2+n}{2}$ (d) $\frac{n^2-n}{2}$

Q.3 What is time complexity of recurrence equation

$$T(n) = T\left(\sqrt{n}\right) + 1$$

(a) $O(\log n)$ (b) $O(n^2)$
(c) $O(n \log \log n)$ (d) $O(\log \log n)$

Q.4 What is complexity of recurrence equation
$T(n)$ = sqrt(2) + $(n/2)$ + sqrt (n)
(a) $\theta(n(\log n + 1))$ (b) $\theta(n^2(\log n + 1))$
(c) $\theta(\sqrt{n}(\log n + 1))$ (d) $\theta(n^3(\log n + 1))$

Q.5 $T(n) = T(2n/3) + 1$ then $T(n)$ is equal to
(a) $\Theta(\log_2 n)$ (b) $\Theta(n \log_2 n)$
(c) $\Theta(n^2)$ (d) $\Theta(n)$

Q.6 The running time of an algorithm is given by
$T(n) = T(n-1) + T(n-2) - T(n-3)$, if $n > 3$
n, otherwise
The order of this algorithms is
(a) $O(n)$ (b) $O(\log n)$
(c) $O(n^n)$ (d) $O(n^2)$

Q.7 What should be the relation between $T(1)$, $T(2)$ and $T(3)$, so that in above question gives an algorithm whose order is constant?
(a) $T(1) = T(2) = T(3)$ (b) $T(1) + T(3) = 2T(2)$
(c) $T(1) - T(3) = T(2)$ (d) $T(1) + T(2) = T(3)$

Q.8 What is complexity of recurrence eq.

$$T(n) = 2T(n/4) + \sqrt{3}$$

(a) $O(\sqrt{n})$ (b) $O(n)$
(c) $O(n^2)$ (d) $O(n \log n)$

Q.9 Which recurrence relation satisfy the sequence: 2, 3, 4, ..., for $n \geq 1$.
(a) $T(N) = 2T(N-1) - T(N-2)$
(b) $T(N) = 2T(N-1) + T(N-2)$
(c) $T(N) = N + 1$
(d) None of these

Q.10 Which one of the following correctly determines the solution of the recurrence relation with $T(1) = 1$?

$$T(n) = 2T\left(\frac{n}{2}\right) + \log n$$

(a) $\Theta(n)$ (b) $\Theta(n \log n)$
(c) $\Theta(n^2)$ (d) $\Theta(\log n)$

Q.11 The time complexity of computing the transitive closure of a binary relation on a set of n elements is known to be
(a) $O(n \log n)$ (b) $O(n^{3/2})$
(c) $O(n^3)$ (d) $O(n)$

Q.12 Let $T(n) = T(n-1) + \frac{1}{n}$; $T(1) = 1$; Then $T(n) = ?$
(a) $O(n^2)$ (b) $O(n \log n)$
(c) $O(\log n)$ (d) $O(n^2 \log n)$

Q.13 A certain problem is having an algorithm with the following recurrence relation.

$$T(n) = T(\sqrt{n}) + 1$$

How much time would the algorithm take to solve the problem?
(a) $O(\log n)$ (b) $O(n \log n)$
(c) $O(\log \log n)$ (d) $O(n)$

Q.49 What is complexity of recurrence relation

$$T(n) = 3T\left(\frac{n}{2}+47\right) + 2n^2 + 10n - \frac{1}{2}$$

(a) $O(n^2)$ (b) $O(n^{3/2})$
(c) $O(n \log n)$ (d) none of these

Q.50 Consider the following recurrance relation

$$T(n) = 7T(n/2) + an^2$$

what will be the solution of above recurrence?
(a) $0(n^2)$ (b) $0(n^3)$
(c) $0(n^2 \log n)$ (d) $0(n^{2.81})$

Q.51 In Tower of Hanoi there are three towers: left, right and middle. There are n discs in left tower in a particular sequence (ascending order), we have to transfer these discs into right tower having same sequence using middle tower (consider that disc with smaller sequence no. is always at top of large sequence no. i.e. 1 is above 2 but 2 above 1 is not allowed). If Towers of Hanoi is implemented with recursive function then total discs moves to move 9 discs from left to right are _______.

Q.52 Let m, n be positive integers. Define $Q(m, n)$ as
$Q(m, n) = 0$, if $m < n$
$Q(m - n, n) + p$, if $m \geq n$
Then $Q(m, 3)$ is (a div b, gives the quotient when a is divided by b)
(a) a constant (b) $p \times (m \bmod 3)$
(c) $p \times (m \text{ div } 3)$ (d) $3 \times p$

Q.53 Find complexity of Recurrence Relation
$T(n) = T(n - 1) + T(n - 2) + C$
(a) $O(2^n)$ (b) $O(n^2)$
(c) $O(n)$ (d) $O(n^n)$

Q.54 The recurrence relation capturing the optimal execution time of the Towers of Hanoi problem with n discs is
(a) $T(n) = 2T(n - 2) + 2$
(b) $T(n) = 2T(n - 1) + n$
(c) $T(n) = 2T(n/2) + 1$
(d) $T(n) = 2T(n - 1) + 1$

■■■■

Answers Recurrence Relations

1.	(c)	2.	(c)	3.	(d)	4.	(c)	5.	(a)	6.	(a)	7.	(a)	8.	(a)	9.	(a)
10.	(a)	11.	(c)	12.	(b)	13.	(c)	14.	(c)	15.	(b)	16.	(b)	17.	(a)	18.	(b)
19.	(c)	20.	(d)	21.	(b)	22.	(a)	23.	(a)	24.	(b)	25.	(a)	26.	(a)	27.	(a)
28.	(a)	29.	(2.32)	30.	(d)	31.	(b)	32.	(d)	33.	(b)	34.	(b)	35.	(a)	36.	(a)
37.	(a)	38.	(a)	39.	(c)	40.	(c)	41.	(c)	42.	(a)	43.	(c)	44.	(a)	45.	(d)
46.	(d)	47.	(a)	48.	(a)	49.	(a)	50.	(d)	51.	(511)	52.	(c)	53.	(a)	54.	(d)

Explanations Recurrence Relations

1. (c)

$[n(\log (n^3) - \log n) + \log n]\, n + \log n$

$$= \left[n\left(\log\frac{n^3}{n}\right) + \log n\right] n + \log n$$

$= [n \log n^2 + \log n]n + \log n$
$= n^2 . 2 \log n + n \log n + \log n$
$= 2n^2 \log n + n \log n + \log n = O(n^2 \log n)$
So option (c) is correct.

2. (c)

$T(n) = T(n - 1) + n$

By Repetitive Substitution

$T(n) = [T(n - 2) + n - 1] + n$
$= [(n - 2) + T(n - 3] + (n - 1) + n$
$= 0 + 1 + 2 + ... + n$

$$= \frac{n(n+1)}{2} = \frac{n^2+n}{2}$$

So option (c) is correct.

3. (d)

Since, we have a sqrt term, considering only perfect squares and those which are multiple of 2 as that can take care of log.

$$T(2) = 1 \quad //\text{Assume}$$
$$T(2^2) = T(2) + 1 = 2$$
$$T\left(2^{2^2}\right) = T(4) + 1 = 3$$
$$T\left(2^{2^3}\right) = T(16) + 1 = 4$$
$$T\left(2^{2^4}\right) = T(256) + 1 = 5$$

So, we are getting

$$T(n) = \log\log n + 1$$
$$\Rightarrow \quad T(n) = O(\log\log n)$$

4. (c)

From master theorem, case 2

Here, $a = \sqrt{2}$, $b = 2$, $k = 1/2$, $p = 0$

$a = b$ ^ k which is true.

$p > -1$

So, complexity is theta (n ^ (log a base to b) log ^ (p + 1)n

$$= \theta(\sqrt{n}(\log n + 1))$$

5. (a)

$$T(n) = T\left(\frac{2n}{3}\right) + 1$$

By using recursive tree method:

$$T(n)$$
$$\downarrow$$
$$T\left(\frac{2n}{3}\right) = O(1)$$
$$\downarrow$$
$$T\left(\frac{2^2 n}{3^2}\right) = O(1)$$
$$\downarrow$$
$$T\left(\frac{2^3 n}{3^3}\right) = O(1)$$
$$\vdots$$
$$T\left(\frac{2^k n}{3^k}\right) = O(1)$$

So, height of tree is $\log_{(3/2)} n$.

So, $\quad T(n) = \text{Height} \times \text{Work at each level}$

$$= \log_{3/2} n \times O(1)$$
$$= O(\log_{3/2} n)$$

6. (a)

$$T(n) = \begin{cases} T(n-1) + T(n-2) - T(n-3) & n > 3 \\ n & \text{otherwise} \end{cases}$$

$$T(4) = T(3) + T(2) - T(1)$$
$$= 3 + 2 - 1 = 4$$
$$T(5) = T(4) + T(3) - T(2)$$
$$= 4 + 3 - 2 = 5$$

So by induction,

$$T(n) = T(n-1) + T(n-2) - T(n-3)$$
$$= n - 1 + n - 2 - (n-3)$$
$$= 2n - 3 - n + 3 = n$$

So,

$$T(n) = O(n)$$

7. (a)

Let, $\quad T(1) = T(2)$

$$= T(3) = k \text{ (say)}$$

Then, $\quad T(4) = k + k - k = k$

$$T(5) = k + k - k = k$$

By mathematical induction it can be proved that $T(n) = k$, a constant.

8. (a)

In these type of questions, where master theorem does not apply you can do the following

$$T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{3}$$

Reframe it like this, by putting i in the specified positions.

$$T(n) = 2^i T\left(n/4^i\right) + i\sqrt{3}$$

Now, for $\quad n/4^i = 1$, put $i = \log_4 n$

So, the equation becomes

$$T(n) = 2^{\log_4 n} + \log_4 n\sqrt{3}$$

or

$$T(n) = n^{\log_4 2} + \log_4 n\sqrt{3}$$
$$T(n) = \sqrt{n} + \log_4 n\sqrt{3}$$

and therefore, $\quad T(n) = O(\sqrt{n})$

9. (a)
Take base conditions as $T(0) = 1$ and $T(-1) = 0$.

$$\begin{aligned} T(-1) &= 0 \\ T(1) &= 2T(0) - T(-1) = 2 \\ T(2) &= 2T(1) - T(0) \\ &= 3 \text{ ... and so on} \end{aligned}$$

So, option (a) is correct.

10. (a)

$$T(n) = 2.T\left(\frac{n}{2}\right) + \log n$$

[By Master's Theorem case1]

$$= \Theta(n)$$

11. (c)
The time complexity of computing the transitive closure of a binary relation on a set of n-element will take $O(n^3)$ using Floyd-Warshall's algorithm.

12. (b)

$$T(n) = T(n-1) + \frac{1}{n}$$

$$T(n-1) = T(n-2) + \frac{1}{n-1}$$

$$T(n-2) = T(n-3) + \frac{1}{n-2}$$

$$T(n-3) = T(n-4) + \frac{1}{n-3}$$

$$T(n) = T(n-2) + \frac{1}{n-1} + \frac{1}{n}$$

$$= T(n-3) + \frac{1}{(n-2)} + \frac{1}{(n-1)} + \frac{1}{n}$$

$$T(n) = T(n-4) + \frac{1}{(n-3)} + \frac{1}{(n-2)} + \frac{1}{(n-1)} + \frac{1}{n}$$

$$T(n) = T(n-k) + \frac{1}{(n-(k-1))} + \frac{1}{(n-(k-1))} + \frac{1}{(n-(k-3))} + \frac{1}{(n-(k-4))} + \ldots + \frac{1}{n}$$

$$T(n) = T(n-k) + \frac{1}{(n-k+1)} + \frac{1}{(n-k+2)} + \frac{1}{(n-k+3)} + \ldots + \frac{1}{n-1} + \frac{1}{n}$$

$$T(n) = T(n-k) + \frac{1}{n-k+1} + \frac{1}{n-k+2} + \frac{1}{n-k+3} + \ldots + \frac{1}{n-3} + \frac{1}{n-2} + \frac{1}{n-1} + \frac{1}{n}$$

Thing that the one condition is given if ($n > 1$)
So, we can substitute ($k = n - 1$)

$$T(n) = T(n-(n-1)) + \frac{1}{n-(n-1)+1} + \ldots + \frac{1}{n}$$

$$T(n) = T(1) + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \ldots + \frac{1}{n}$$

Assume that $T(1) = 1$.

$$T(n) = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \ldots. + \frac{1}{n}$$

$$T(n) = O(\log n)$$

13. (c)

$$T(n) = T(\sqrt{n}) + 1$$

$$\text{Put } n = 2^m, \quad \because \quad m = \log n$$

$$T(2^m) = T(\sqrt{2^m}) + 1$$

$$\underbrace{\boxed{T(2}^m)}_{S} = \underbrace{\boxed{T(2}^{m/2}}_{S}) + 1$$

$$S(m) = S(m/2) + 1$$

Using Master's Theorem

$$S(m) = \log m \quad (\because \quad m = \log n)$$

$$T(n) = O(\log \log n)$$

14. (c)
Using Master Theorem

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

$$n^{\log_b^a} = n^{\log_2^8} = n^3$$

$$f(n) = n^2 \log n$$

If $\quad n^{\log_b^a} > f(n)$ then

$$T(n) = O(n^{\log_b^a})$$

$\Rightarrow \quad n^3 > n^2$

$\Rightarrow \quad T(n) = O(n^3)$

So option (c) is correct.

15. (b)
According to Master theorem
$a = 7,\ b = 3,\ k = 2$
So, its case of $a < b^k$
So, $O(N^2)$ will be complexity
So, option (b) is correct.