Algorithms

Lecture 4
Recursive Time Complexity

Prantik Paul [PNP]
Lecturer
Department of Computer Science and Engineering
BRAC University

Solving Recurrences- Methods

1. Iteration Method
2. Recursion Tree Method
3. Master Theorem

Merge Sort (Algorithm)

$\textsc{Merge-Sort}(A)$ ▷ $A[1\,..\,n]$

if $n = 1$
    then return
    else ▷ recursively sort the two subarrays
        $A_1 = \textsc{Merge-Sort}(A[1\,..\,\lceil n/2 \rceil])$
        $A_2 = \textsc{Merge-Sort}(A[\lceil n/2 \rceil + 1\,..\,n])$
        $A = \textsc{Merge}(A_1, A_2)$ ▷ merge the sorted arrays

Merge (Algorithm)

MERGE(A, B)

INPUT: Two sorted arrays A and B

OUTPUT: Returns C as the merged array

$\triangleright\ n_1 = length[A],\ n_2 = length[B],\ n = n_1 + n_2$

Create $C[1 \,..\, n]$

Initialize two indices to point to A and B

while A and B are not empty

    do Select the smaller of two and add to end of C

        Advance the index that points to the smaller one

if A or B is not empty

  then Copy the rest of the non-empty array to the end of C

return C

Merge Sort: Running Time

$$T(n) = \underbrace{2}_{\text{\# subproblems}} T(\underbrace{n/2}_{\text{subproblem size}}) + \underbrace{\Theta(n)}_{\text{work dividing and combining}}$$

Merge Sort: Running Time

The recurrence for the worst-case running time T(n) is

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

equivalently

$$T(n) = \begin{cases} b & \text{if } n = 1 \\ 2T(n/2) + bn & \text{if } n > 1 \end{cases}$$

Solve this recurrence by

(1) iteratively expansion

Iterative Approach

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

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

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

Merge Sort: Running Time

$$\begin{aligned}T(n) &= 2T(n/2) + bn \\ &= 2(2T(n/2^2)) + b(n/2)) + bn \\ &= 2^2T(n/2^2) + 2bn \\ &= 2^3T(n/2^3) + 3bn \\ &= 2^4T(n/2^4) + 4bn \\ &= \ldots \\ &= 2^iT(n/2^i) + ibn\end{aligned}$$

Refer to notes

- Note that base, $T(n) = b$, case occurs when $2^i = n$.
  That is, $i = \log n$.
- So, $T(n) = bn + bn \log n$
- Thus, $T(n)$ is $O(n \log n)$.

Iteration Method

Try:

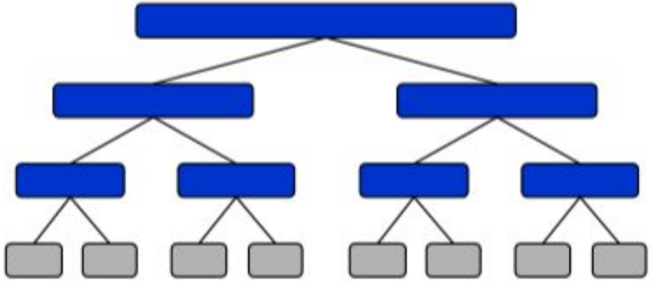
1. $T(n) = 4T(n/2) + bn$
2. $T(n) = 3T(n/2) + bn$

Recursion Tree Method

- Draw the recursion tree for the recurrence relation and look for a pattern:

$$T(n) = \begin{cases} b & \text{if } n = 1 \\ 2T(n/2) + bn & \text{if } n \geq 2 \end{cases}$$

depth	T's	size	time
0	1	$\boldsymbol{n}$	$\boldsymbol{bn}$
1	2	$\boldsymbol{n}/2$	$\boldsymbol{bn}$
$\boldsymbol{i}$	2^i	$\boldsymbol{n}/2^i$	$\boldsymbol{bn}$
...	...	...	...



Total time = $\boldsymbol{bn} + \boldsymbol{bn} \log \boldsymbol{n}$

(last level plus all previous levels)

Quick Sort (Algorithm)

Algorithm

$\textsc{Quicksort}(A, p, r) \triangleright A[p \mathinner{..} r]$

if $p < r$

  then $q \leftarrow \textsc{Partition}(A, p, r)$

    $\textsc{Quicksort}(A, p, q - 1)$

    $\textsc{Quicksort}(A, q + 1, r)$

Partitioning (Algorithm)

Algorithm

$\text{PARTITION}(A, p, q) \triangleright A[p\,..\,q]$

$x \leftarrow A[p]$ $\qquad \triangleright$ pivot $= A[p]$

$i \leftarrow p$

for $j \leftarrow p + 1$ **to** q

  do if $A[j] \leq x$

    then $i \leftarrow i + 1$

      exchange $A[i] \leftrightarrow A[j]$

exchange $A[p] \leftrightarrow A[i]$

return i

Quick Sort: Running Time

Worst-case analysis

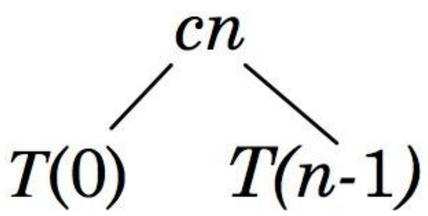
Quick Sort: Recursion Tree Worst Case

$$T(n) = T(0) + T(n-1) + cn$$

$$T(n)$$

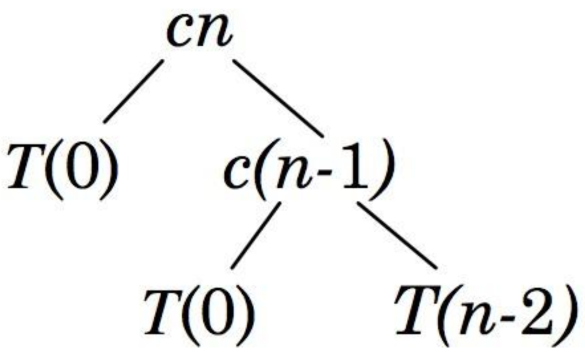
Quick Sort: Recursion Tree Worst Case

$$T(n) = T(0) + T(n-1) + cn$$



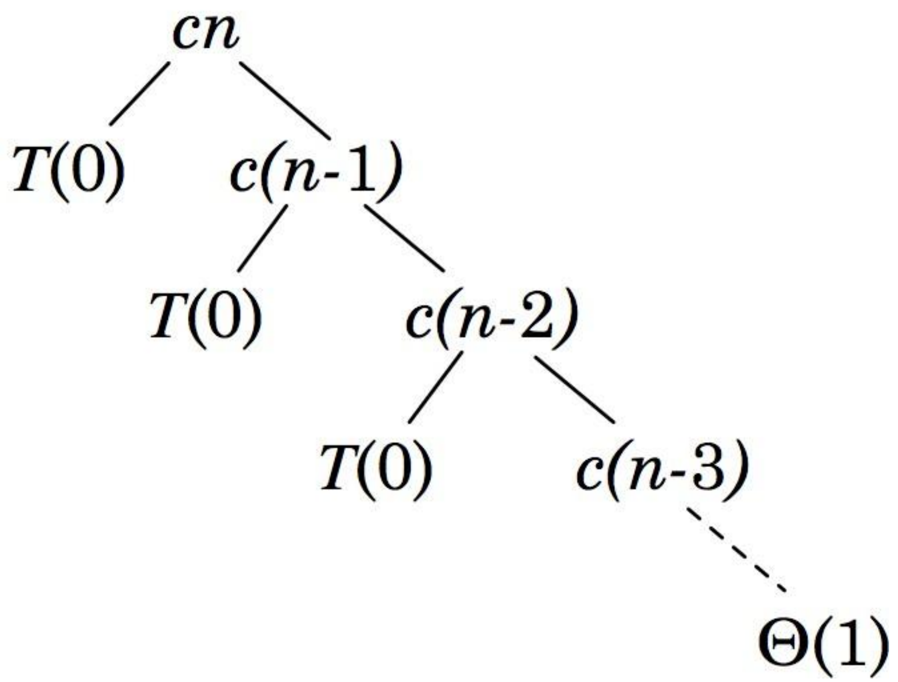
Quick Sort: Recursion Tree Worst Case

$$T(n) = T(0) + T(n-1) + cn$$



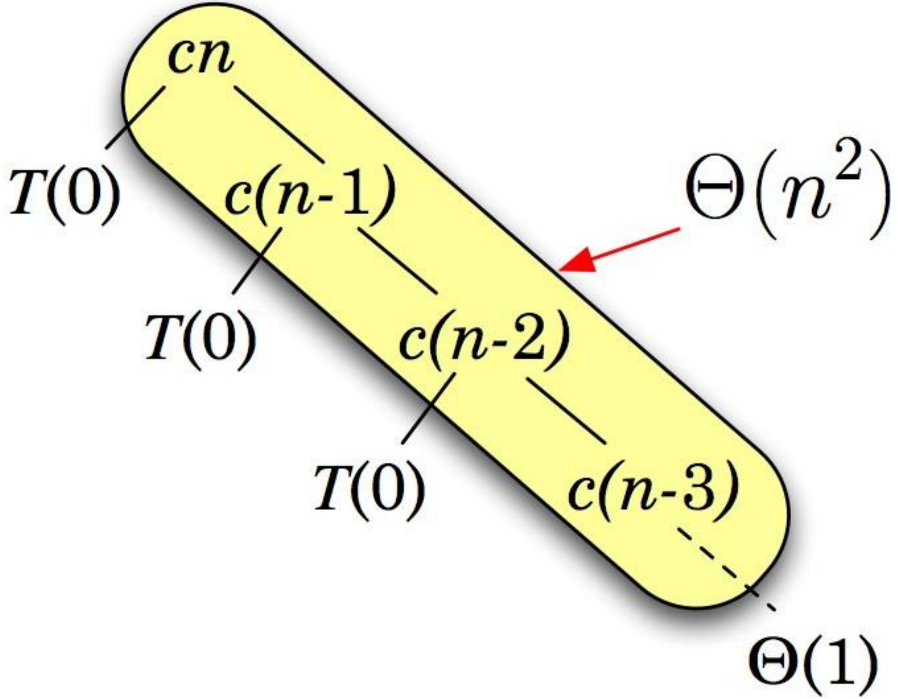
Quick Sort: Recursion Tree Worst Case

$$T(n) = T(0) + T(n-1) + cn$$



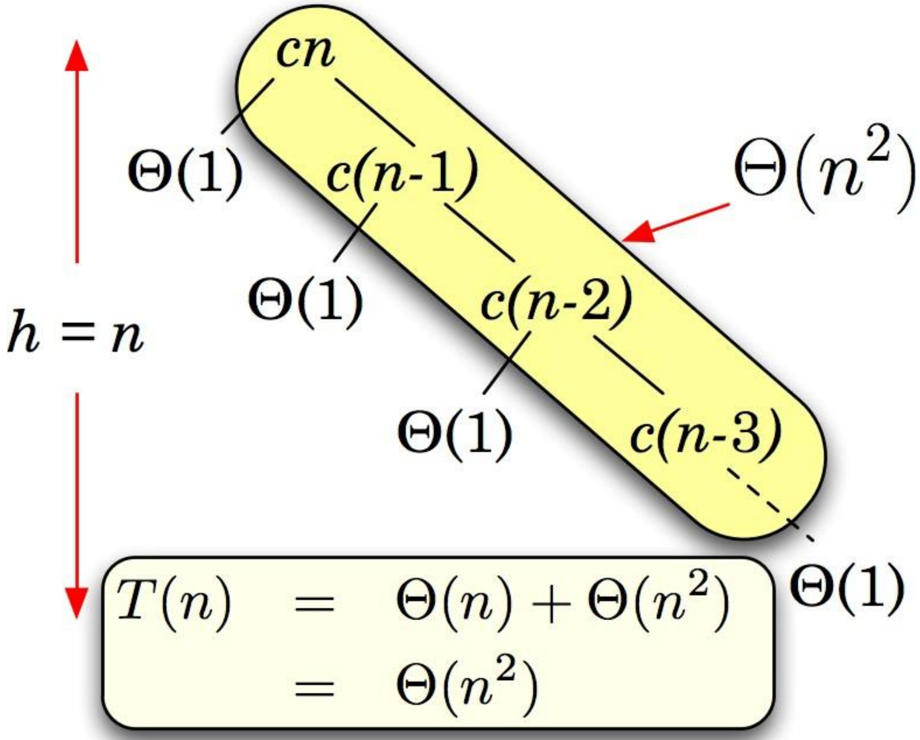
Quick Sort: Recursion Tree Worst Case

$$T(n) = T(0) + T(n-1) + cn$$



Quick Sort: Recursion Tree Worst Case

$$T(n) = T(0) + T(n-1) + cn$$



Quick Sort: Running Time (Best Case)

- Best-case happens when pivot is the median element, creating equal size partitions.

Quick Sort: Running Time (Almost Worst Case)

- What if the split is always $\frac{1}{10} : \frac{9}{10}$?

Refer to book

Master Theorem

The Master Theorem is a tool used to solve recurrence relations that arise in the analysis of divide-and-conquer algorithms. The Master Theorem provides a systematic way of solving recurrence relations of the form:

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

where a, b, and f(n) are positive functions and n is the size of the problem. The Master Theorem provides conditions for the solution of the recurrence to be in the form of O(n^k) for some constant k, and it gives a formula for determining the value of k.

Master Theorem

The Master Theorem is a tool used to solve recurrence relations that arise in the analysis of divide-and-conquer algorithms. The Master Theorem provides a systematic way of solving recurrence relations of the form:

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

Not all recurrence relations can be solved with the use of the master theorem i.e. if

- $T(n)$ is not monotone, ex: $T(n) = \sin n$
- $f(n)$ is not a polynomial, ex: $T(n) = 2T(n/2) + 2^n$

Master Theorem

This theorem is an advance version of master theorem that can be used to determine running time of divide and conquer algorithms if the recurrence is of the following form :-

$$T(n) = aT(n/b) + \theta(n^k \log^p n)$$

Master Theorem

$$T(n) = aT(n/b) + \theta(n^k \log^p n)$$

1. if $a > b^k$, then $T(n) = \theta(n^{\log_b a})$
2. if $a = b^k$, then
   (a) if $p > -1$, then $T(n) = \theta(n^{\log_b a} \log^{p+1} n)$
   (b) if $p = -1$, then $T(n) = \theta(n^{\log_b a} \log\log n)$
   (c) if $p < -1$, then $T(n) = \theta(n^{\log_b a})$
3. if $a < b^k$, then
   (a) if $p >= 0$, then $T(n) = \theta(n^k \log^p n)$
   (b) if $p < 0$, then $T(n) = \theta(n^k)$

Master Theorem

- **Example-2: Merge Sort** – $T(n) = 2T(n/2) + O(n)$

  $a = 2, b = 2, k = 1, p = 0$

  $b^k = 2$. So, $a = b^k$ and $p > -1$ [Case 2.(a)]

  $T(n) = \theta(n^{\log_b a} \log^{p+1} n)$

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

Master Theorem

- **Example-3:** $T(n) = 3T(n/2) + n^2$

  $a = 3, b = 2, k = 2, p = 0$

  $b^k = 4$. So, $a < b^k$ and $p = 0$ [Case 3.(a)]

  $T(n) = \theta(n^k \log^p n)$

  $T(n) = \theta(n^2)$

Master Theorem

- **Example-4:** $T(n) = 3T(n/2) + \log^2 n$

  $a = 3, b = 2, k = 0, p = 2$

  $b^k = 1$. So, $a > b^k$ [Case 1]

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

  $T(n) = \theta(n^{\log_2 3})$

Master Theorem

- **Example-5:** $T(n) = 2T(n/2) + n\log^2 n$

  $a = 2, b = 2, k = 1, p = 2$

  $b^k = 2$. So, $a = b^k$ [Case 2.(a)]

  $T(n) = \theta(n^{\log_b a}\log^{p+1} n\ )$

  $T(n) = \theta(n^{\log_2 2}\log^3 n)$

  $T(n) = \theta(n\log^3 n)$

Master Theorem

- **Example-6:** $T(n) = 2^nT(n/2) + n^n$

  This recurrence can't be solved using above method since function is not of form $T(n) = aT(n/b) + \theta(n^k \log^p n)$

Master Theorem

$$T(n) = 2T(n/2) + n \lg n$$

Not solvable by general theorem,
but solvable by advanced version

Master Theorem

Exercises

4.5-1

Use the master method 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. $T(n) = 2T(n/4) + n^2$.

Resources

- CLRS Book:
  - Chapter 2.3.2 - Merge Sort Analysis
  - Chapter 4.4 - Recursion Tree Method
  - Chapter 4.5 - Master Method
  - Chapter 7.4 - Quicksort Analysis
- Recursive Time Complexity Notes
- Master Theorem Explanation