CS 341 – Algorithms

Lecture 2 – Solving Recurrence

14 May 2021

Today’s Plan

1. Merge Sort
2. Master Theorem
3. More Recurrences ?

Merge Sort

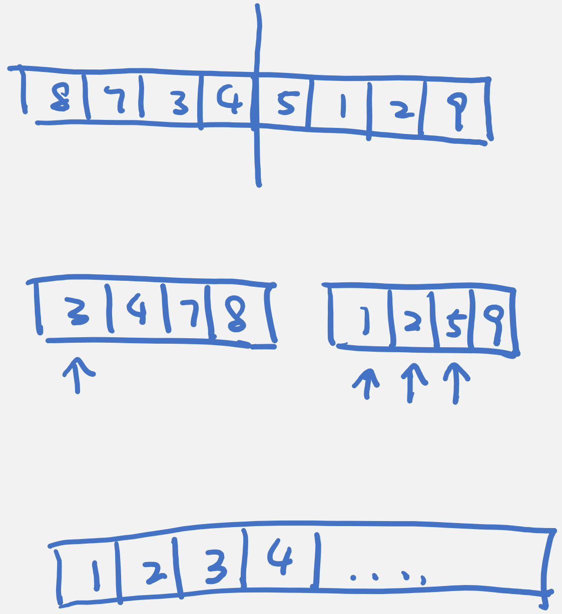
This is a classical algorithm using the idea of divide and conquer.

Input: n numbers $a_1 \dots a_n$

output: sorted list, increasing

Obs: if both halves are sorted,
then it is easy to "merge"

merge in $O(n)$ time.



Recursive Algorithm

How do we assume that each halve is already sorted? The idea is to apply the same procedure **recursively**.

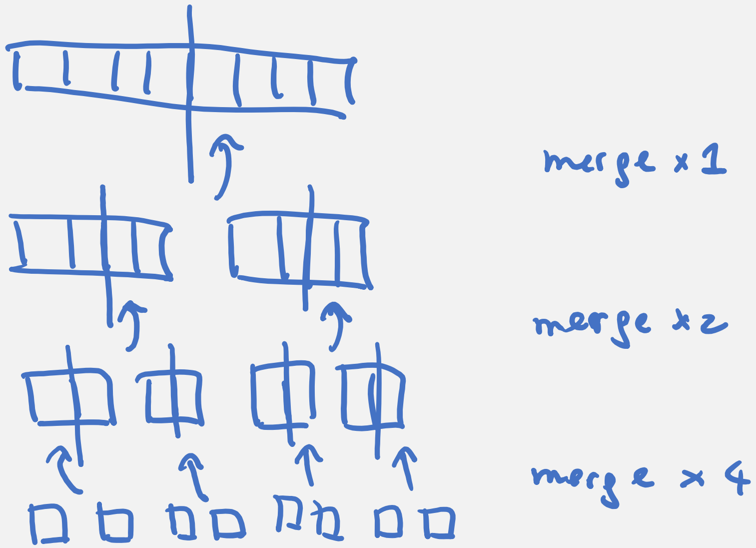
Sort (1, n)

if n=1, return. // base case

Sort $(1, \frac{n}{2})$

sort $(\frac{n}{2}+1, n)$

merge the two sorted lists



Solving the Recurrence Relation

To analyze the time complexity, we need to solve the recurrence relation.

for simplicity, n is a power of 2.

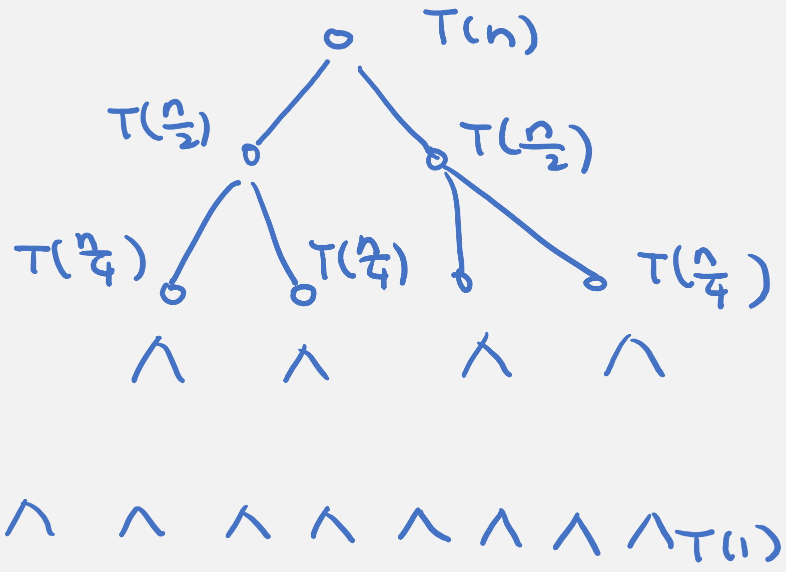
$T(n)$: time for merge sort with n elements.

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + cn$$ where c constant.

recursion tree

how many levels?

$\log_2 n$



merge cn $= cn$

merge $2 \times c\left(\frac{n}{2}\right)$ $= cn$

$4 \times c\left(\frac{n}{4}\right)$ $= cn$

⋮ ⋮

$n \times c(1)$ $= cn$

$\Rightarrow$ total time complexity $= cn \log_2 n$.

Proving by Induction

induction hypothesis : $T(n) = c\,n\log_2 n$

induction step : $T(m) = 2T(\frac{m}{2}) + cm$

$= 2(c(\frac{m}{2})\log_2(\frac{m}{2})) + cm$

$= cm(\log_2 m - 1) + cm$

$= cm\log_2 m.$

hypothesis $T(n) = cn$

$T(m) = 2T(\frac{m}{2}) + cm$

$= 2c(\frac{m}{2}) + cm$

$= 2cm$

induction hypothesis : $T(n) = O(n)$

induction step: $T(m) = 2T(\frac{m}{2}) + cm$

$= 2\,O(\frac{m}{2}) + cm$

$= O(m).$

size $O(1)$

$O(1)$ time ✓

Today’s Plan

1. Merge Sort
2. Master Theorem
3. More Recurrences

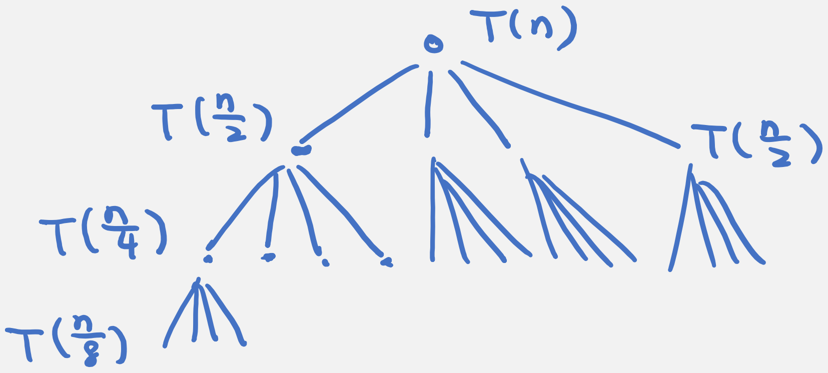
1. Homework 1 posted
2. TA office hours, Tuesdays 8:30-9:30pm
   Thursdays 3:30-4:30pm
3. Tutorials, Mondays 1-2pm
   First week on May 25 because of Victoria Day

Exercises

1. $T(n) = 4T\left(\frac{n}{2}\right) + n$

$T(n) = O(n^2)$

$\log_2 n$ levels



$n = n$

$4 \times \frac{n}{2} = 2n$

$16 \times \frac{n}{4} = 4n$

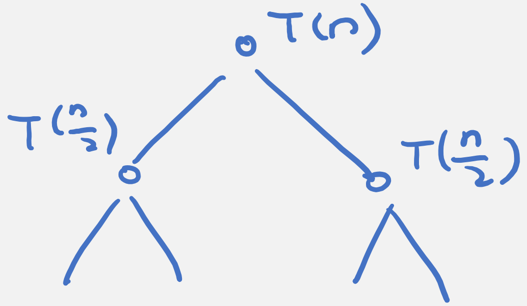
$64 \times \frac{n}{8} = 8n$

$\leftarrow \quad 4^{\log_2 n} \times 1 = \left(2^{\log_2 n}\right)^2 = n^2$

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

$T(n) = O(n^2)$

$\log_2 n$ levels



$n^2 = n^2$

$2 \times \left(\frac{n}{2}\right)^2 = \frac{n^2}{2}$

$4 \times \left(\frac{n}{4}\right)^2 = \frac{n^2}{4}$

$8 \times \left(\frac{n}{8}\right)^2 = \frac{n^2}{8}$

Recursion Tree in the General Setting

Consider the recurrence relation $T(n) = a \cdot T\left(\frac{n}{b}\right) + n^c$ where $a \geq 1, b > 0, c \geq 0$.

integers

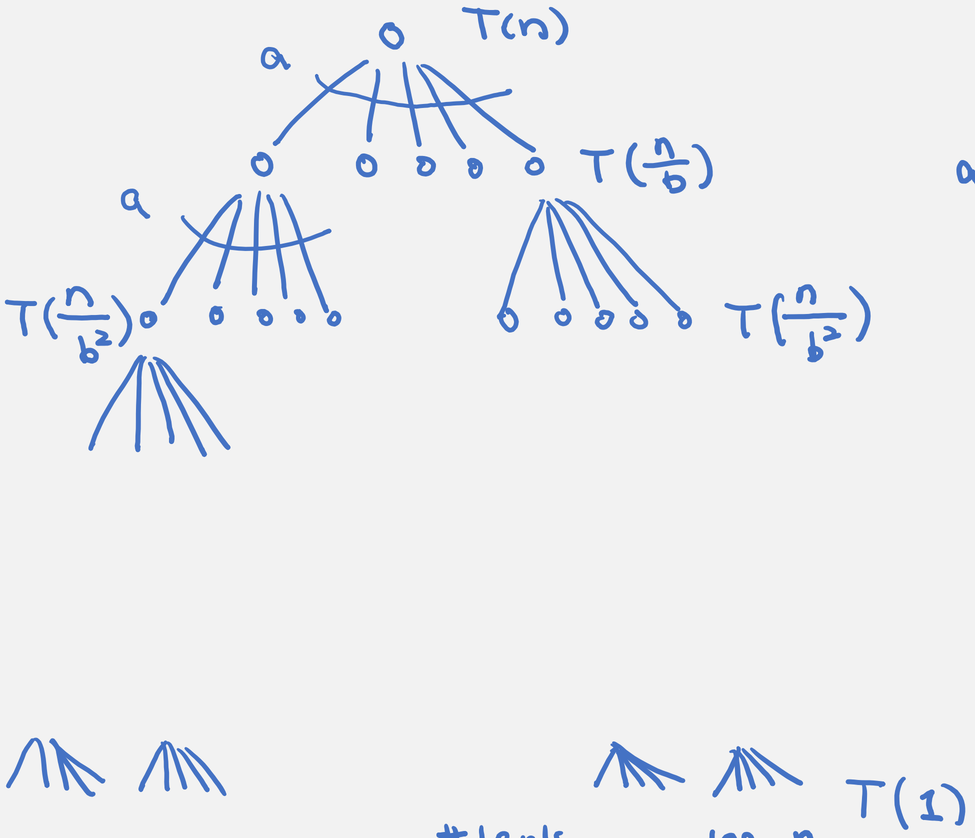
n is a power of b

levels is $\leq \log_b n$

$\frac{n}{b^i} \leq c$

$n \leq c\, b^i$

$\log_b n \leq \log_b c + i$



$\#\text{ leaves} = a^{\#\text{levels}} = a^{\log_b n}$

$= n^{\log_b a}$

$n^c = n^c$

$a \cdot \left(\frac{n}{b}\right)^c = n^c \left(\frac{a}{b^c}\right)$

$a^2 \left(\frac{n}{b^2}\right)^c = n^c \cdot \left(\frac{a}{b^c}\right)^2$

$a^3 \left(\frac{n}{b^3}\right)^c = n^c \cdot \left(\frac{a}{b^c}\right)^3$

$\vdots$

geometric sequence

ratio $= \frac{a}{b^c}$

Master Theorem

There are three cases to consider, depending on the ratio in the geometric sequence.

merge sort

① if $\frac{a}{b^c} = 1$, every level has the same work

total work = (# levels)·(work at each level) = $n^c \cdot \log_b n$.

② if $\frac{a}{b^c} < 1$, root level dominates

total work = $O(n^c)$ hidden constant $\leq \frac{1}{1-r} = \frac{1}{1-\frac{a}{b^c}}$

if assume a, b, c constants

③ if $\frac{a}{b^c} > 1$, leaf level dominates

total work = $O(n^{\log_b a})$.

Thm a, b, c constants.

$T(n) = aT(\frac{n}{b}) + n^c$

$$T(n) = \begin{cases} O(n^c \cdot \log_b n) & \frac{a}{b^c} = 1 \\ O(n^{\log_b a}) & \frac{a}{b^c} > 1 \\ O(n^c) & \frac{a}{b^c} < 1 \end{cases}$$

Today’s Plan

1. Merge Sort
2. Master Theorem
3. More Recurrences

Single Subproblem

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

e.g. binary search

$T(n)$	1
$T(\frac{n}{2})$	1
$T(\frac{n}{4})$	1
$T(\frac{n}{8})$	1
$\vdots$	
$T(1)$	1

total work

$= \#$ levels

$\leq \log_2 n$

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

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

$T(n)$	n
$T(\frac{n}{2})$	$\frac{n}{2}$
$T(\frac{n}{4})$	$\frac{n}{4}$
$\vdots$	$\vdots$
$T(1)$	1

total work

$= O(n)$

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

i-th level

$n^{\frac{1}{2^i}} \leq c$

$T(n)$	1
$T(\sqrt{n})$	1
$T(n^{\frac{1}{4}})$	1
$T(n^{\frac{1}{8}})$	1
$\vdots$	

total work

$= \#$ levels

if $i = \log_2 \log_2 n$

then $n^{\frac{1}{\log_2 n}} = \Theta(1)$

$\frac{1}{2^i} \log n \leq \log c$

$\log n \leq 2^i \cdot \log c$

$i \geq \log \log n$

Uneven Subproblems

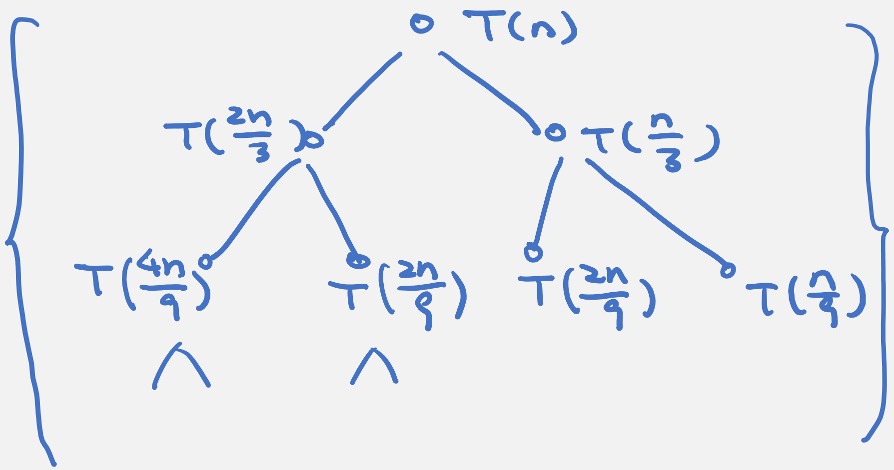
1. $T(n) = T\left(\frac{2n}{3}\right) + T\left(\frac{n}{3}\right) + n$

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

not tight.

total work $= \Theta(n \log n)$

\# levels $\le \log_{1.5} n$



n

$\frac{2n}{3} + \frac{n}{3} = n$

$\frac{4n}{9} + \frac{2n}{9} + \frac{2n}{9} + \frac{n}{9} = n$

$\Rightarrow$ \# full levels $= \log_3 n$

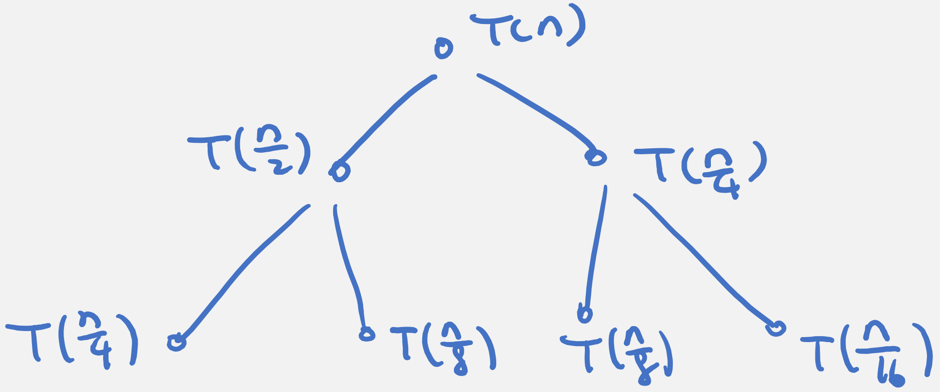
2. $T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + n$

$T(n) = O(n)$

induction hypothesis $T(n) \le 4n$

e.g. $T(n) \le 2T\left(\frac{n}{2}\right) + n$

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



n

$\frac{n}{2} + \frac{n}{4} = \frac{3n}{4}$

ratio $= \frac{3}{4}$

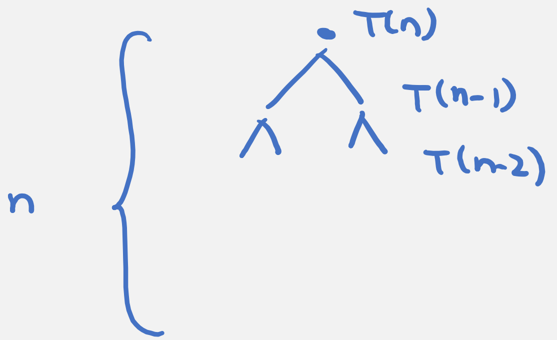
$\frac{n}{4} + \frac{n}{8} + \frac{n}{8} + \frac{n}{16} = \frac{9n}{16}$

We will see an interesting problem later with this kind of recurrence relation.

Exponential Time

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

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



1
2
4
8
⋮
2^n

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

Fibonacci sequence

MATH 239 $x^2 - x - 1 = 0$

$T(n) = \left(\frac{1+\sqrt{5}}{2}\right)^n \approx 1.618^n$

Analyzing Maximum Independent Set (Optional)

Input: Graph $G = (V, E)$

NP-complete

Output: An independent set $S \subseteq V$ of maximum size (where S is independent if $uv \notin E$ for all $u, v \in S$)

naive : enumerate all S : $\Omega(2^n)$ iterations

slightly better exhaustive search

v

not choosing v in S

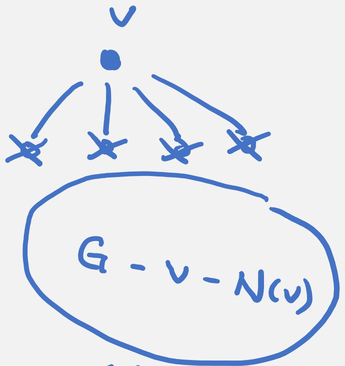
max ind set in $G-v$

$T(n-1)$

choose v in S

max ind set in $G - v - N(v)$

$\leq T(n-2)$



$T(n) \leq T(n-1) + T(n-2) + n^c$

$T(n) \leq O(1.618^n \cdot n^c)$.

Summary

Understand the recursion tree method.

This will be accepted as a correct solution. Of course, inductive solution will also be accepted.

We will apply this to analyzing time complexity of divide and conquer algorithms.