CE100 Algorithms and Programming II

Solving Recurrences / The Divide-and-Conquer

Author: Asst. Prof. Dr. Uğur CORUH

## CE100 Algorithms and Programming II

## Week-2 (Solving Recurrences / The Divide-and-Conquer)

#### Spring Semester, 2021-2022

Download [DOC-PDF](ce100-week-2-recurrence.en.md_doc.pdf), [DOC-DOCX](ce100-week-2-recurrence.en.md_word.docx), [SLIDE](ce100-week-2-recurrence.en.md_slide.pdf), [PPTX](ce100-week-2-recurrence.en.md_slide.pptx)

## Solving Recurrences

## Outline (1)

* Solving Recurrences
	+ Recursion Tree
	+ Master Method
	+ Back-Substitution

## Outline (2)

* Divide-and-Conquer Analysis
	+ Merge Sort
	+ Binary Search
	+ Merge Sort Analysis
	+ Complexity

## Outline (3)

* Recurrence Solution

## Solving Recurrences (1)

* Reminder: Runtime $\left(T\left(n\right)\right)$ of *MergeSort* was expressed as a recurrence

$$T\left(n\right)=\left\{\begin{matrix}Θ\left(1\right)&if n=1\\2T\left(n/2\right)+Θ\left(n\right)&otherwise\end{matrix}\right.$$

* Solving recurrences is like solving differential equations, integrals, etc.
	+ Need to learn a few tricks

## Solving Recurrences (2)

**Recurrence:** An equation or inequality that describes a function in terms of its value on smaller inputs.

Example :

$$T\left(n\right)=\left\{\begin{matrix}1&if n=1\\T\left(⌈n/2⌉\right)+1&if n>1\end{matrix}\right.$$

## Recurrence Example

$$T\left(n\right)=\left\{\begin{matrix}1&if n=1\\T\left(⌈n/2⌉\right)+1&if n>1\end{matrix}\right.$$

* Simplification: Assume $n=2^{k}$
* Claimed answer : $T\left(n\right)=lgn+1$
* Substitute claimed answer in the recurrence:

$$lgn+1=\left\{\begin{matrix}1&if n=1\\lg\left(⌈n/2⌉\right)+2&if n>1\end{matrix}\right.$$

* True when $n=2^{k}$

## Technicalities: Floor / Ceiling

Technically, should be careful about the floor and ceiling functions (as in the book).

e.g. For merge sort, the recurrence should in fact be:,

$$T\left(n\right)=\left\{\begin{matrix}Θ\left(1\right)&if n=1\\T\left(⌈n/2⌉\right)+T\left(⌊n/2⌋\right)+Θ\left(n\right)&if n>1\end{matrix}\right.$$

But, it’s usually ok to:

* ignore floor/ceiling
* solve for the exact power of 2 (or another number)

## Technicalities: Boundary Conditions

* Usually assume: $T\left(n\right)=Θ\left(1\right)$ for sufficiently small $n$
	+ Changes the exact solution, but usually the asymptotic solution is not affected (e.g. if polynomially bounded)
* For convenience, the boundary conditions generally implicitly stated in a recurrence
	+ $T\left(n\right)=2T\left(n/2\right)+Θ\left(n\right)$ assuming that
	+ $T\left(n\right)=Θ\left(1\right)$ for sufficiently small $n$

## Example: When Boundary Conditions Matter

Exponential function: $T\left(n\right)=\left(T\left(n/2\right)\right)2$ Assume
$T\left(1\right)=c (where c is a positive constant)$ $T\left(2\right)=\left(T\left(1\right)\right)^{2}=c^{2}$ $T\left(4\right)=\left(T\left(2\right)\right)^{2}=c^{4}$ $T\left(n\right)=Θ\left(c^{n}\right)$ e.g.

$$ However Θ\left(2^{n}\right)\ne Θ\left(3^{n}\right)\left\{\begin{matrix}T\left(1\right)=2&⇒&T\left(n\right)=Θ\left(2^{n}\right)\\T\left(1\right)=3&⇒&T\left(n\right)=Θ\left(3^{n}\right)\end{matrix}\right.$$

The difference in solution more dramatic when:

$$T\left(1\right)=1⇒T\left(n\right)=Θ\left(1^{n}\right)=Θ\left(1\right)$$

## Solving Recurrences Methods

We will focus on 3 techniques

* Substitution method
* Recursion tree approach
* Master method

## Substitution Method

The most general method:

* Guess
* Prove by induction
* Solve for constants

## Substitution Method: Example (1)

Solve $T\left(n\right)=4T\left(n/2\right)+n$ (assume $T\left(1\right)=Θ\left(1\right)$)

1. Guess $T\left(n\right)=O\left(n^{3}\right)$ (need to prove $O$ and $Ω$ separately)
2. Prove by induction that $T\left(n\right)\leq cn^{3}$ for large $n$ (i.e. $n\geq n\_{0}$)
	* Inductive hypothesis: $T\left(k\right)\leq ck^{3}$ for any $k<n$
	* Assuming ind. hyp. holds, prove $T\left(n\right)\leq cn^{3}$

## Substitution Method: Example (2)

Original recurrence: $T\left(n\right)=4T\left(n/2\right)+n$

From inductive hypothesis: $T\left(n/2\right)\leq c\left(n/2\right)^{3}$

Substitute this into the original recurrence:

* $T\left(n\right)\leq 4c\left(n/2\right)^{3}+n$
* $=\left(c/2\right)n^{3}+n$
* $=cn^{3}–\left(\left(c/2\right)n^{3}–n\right)$ $⇐$ desired - residual
* $\leq cn^{3}$ when $\left(\left(c/2\right)n^{3}–n\right)\geq 0$

## Substitution Method: Example (3)

So far, we have shown:

$$T\left(n\right)\leq cn^{3} when \left(\left(c/2\right)n^{3}–n\right)\geq 0$$

We can choose $c\geq 2$ and $n\_{0}\geq 1$

But, the proof is not complete yet.

**Reminder: Proof by induction:** *1.Prove the base cases* $⇐$ haven’t proved the base cases yet *2.Inductive hypothesis for smaller sizes* *3.Prove the general case*

## Substitution Method: Example (4)

* We need to prove the base cases
	+ Base: $T\left(n\right)=Θ\left(1\right)$ for small $n$ (e.g. for $n=n\_{0}$)
* We should show that:
	+ $Θ\left(1\right)\leq cn^{3}$ for $n=n\_{0}$ , This holds if we pick $c$ big enough
* So, the proof of $T\left(n\right)=O\left(n^{3}\right)$ is complete
* But, is this a tight bound?

## Example: A tighter upper bound? (1)

* Original recurrence: $T\left(n\right)=4T\left(n/2\right)+n$
* Try to prove that $T\left(n\right)=O\left(n^{2}\right)$,
	+ i.e. $T\left(n\right)\leq cn^{2}$ for all $n\geq n\_{0}$
* **Ind. hyp:** Assume that $T\left(k\right)\leq ck^{2}$ for $k<n$
* **Prove the general case:** $T\left(n\right)\leq cn^{2}$

## Example: A tighter upper bound? (2)

Original recurrence: $T\left(n\right)=4T\left(n/2\right)+n$ Ind. hyp: Assume that $T\left(k\right)\leq ck^{2}$ for $k<n$ Prove the general case: $T\left(n\right)\leq cn^{2}$

$$\begin{matrix}T\left(n\right)&=4T\left(n/2\right)+n\\&\leq 4c\left(n/2\right)^{2}+n\\&=cn^{2}+n\\&=O\left(n2\right)⇐ Wrong! We must prove exactly\end{matrix}$$

## Example: A tighter upper bound? (3)

**Original recurrence:** $T\left(n\right)=4T\left(n/2\right)+n$ **Ind. hyp:** Assume that $T\left(k\right)\leq ck^{2}$ for $k<n$ **Prove the general case:** $T\left(n\right)\leq cn^{2}$

* So far, we have: $T\left(n\right)\leq cn^{2}+n$
* No matter which positive c value we choose, this does not show that $T\left(n\right)\leq cn^{2}$
* Proof failed?

## Example: A tighter upper bound? (4)

* What was the problem?
	+ The inductive hypothesis was not strong enough
* **Idea:** Start with a stronger inductive hypothesis
	+ Subtract a low-order term
* **Inductive hypothesis:** $T\left(k\right)\leq c\_{1}k^{2}–c\_{2}k$ for $k<n$
* **Prove the general case:** $T\left(n\right)\leq c\_{1}n^{2}−c\_{2}n$

## Example: A tighter upper bound? (5)

**Original recurrence:** $T\left(n\right)=4T\left(n/2\right)+n$

**Ind. hyp:** Assume that $T\left(k\right)\leq c\_{1}k^{2}–c\_{2}k$ for $k<n$

Prove the general case: $T\left(n\right)\leq c\_{1}n^{2}–c\_{2}n$

$$\begin{matrix}T\left(n\right)&=4T\left(n/2\right)+n\\&\leq 4\left(c\_{1}\left(n/2\right)^{2}–c\_{2}\left(n/2\right)\right)+n\\&=c\_{1}n^{2}–2c\_{2}n+n\\&=c\_{1}n^{2}–c\_{2}n–\left(c\_{2}n–n\right)\\&\leq c\_{1}n^{2}–c\_{2}n for n\left(c\_{2}–1\right)\geq 0\\&choose c2\geq 1\end{matrix}$$

## Example: A tighter upper bound? (6)

We now need to prove

$$T\left(n\right)\leq c\_{1}n^{2}–c\_{2}n$$

for the base cases.

$T\left(n\right)=Θ\left(1\right) for 1\leq n\leq n\_{0}$ (implicit assumption)

$Θ\left(1\right)\leq c\_{1}n^{2}–c\_{2}n$ for $n$ small enough (e.g. $n=n\_{0}$)

* We can choose c1 large enough to make this hold

We have proved that $T\left(n\right)=O\left(n^{2}\right)$

## Substitution Method: Example 2 (1)

For the recurrence $T\left(n\right)=4T\left(n/2\right)+n$,

prove that $T\left(n\right)=Ω\left(n^{2}\right)$

i.e. $T\left(n\right)\geq cn^{2}$ for any $n\geq n\_{0}$

**Ind. hyp:** $T\left(k\right)\geq ck^{2}$ for any $k<n$

**Prove general case:** $T\left(n\right)\geq cn^{2}$

$T\left(n\right)=4T\left(n/2\right)+n$ $\geq 4c\left(n/2\right)^{2}+n$ $=cn^{2}+n$ $\geq cn^{2}$ since $n>0$

Proof succeeded – no need to strengthen the ind. hyp as in the last example

## Substitution Method: Example 2 (2)

We now need to prove that $T\left(n\right)\geq cn^{2}$ for the base cases $T\left(n\right)=Θ\left(1\right)$ for $1\leq n\leq n\_{0}$ (implicit assumption) $Θ\left(1\right)\geq cn^{2}$ for $n=n\_{0}$

$n\_{0}$ is sufficiently small (i.e. constant)

We can choose $c$ small enough for this to hold

We have proved that $T\left(n\right)=Ω\left(n^{2}\right)$

## Substitution Method - Summary

* Guess the asymptotic complexity
* Prove your guess using induction
	+ Assume inductive hypothesis holds for $k<n$
	+ Try to prove the general case for $n$
		- Note: $MUST$ prove the $EXACT$ inequality $CANNOT$ ignore lower order terms, If the proof fails, strengthen the ind. hyp. and try again
* Prove the base cases (usually straightforward)

## Recursion Tree Method

* A recursion tree models the runtime costs of a recursive execution of an algorithm.
* The recursion tree method is good for generating guesses for the substitution method.
* The recursion-tree method can be unreliable.
	+ Not suitable for formal proofs
* The recursion-tree method promotes intuition, however.

## Solve Recurrence (1) : $T\left(n\right)=2T\left(n/2\right)+Θ\left(n\right)$



alt:“alt” height:250px center

## Solve Recurrence (2) : $T\left(n\right)=2T\left(n/2\right)+Θ\left(n\right)$



alt:“alt” height:450px center

## Solve Recurrence (3) : $T\left(n\right)=2T\left(n/2\right)+Θ\left(n\right)$



alt:“alt” height:500px center

## Example of Recursion Tree (1)

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



alt:“alt” height:500px center

## Example of Recursion Tree (2)

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

## Example of Recursion Tree (3)

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

## The Master Method

* A powerful black-box method to solve recurrences.
* The master method applies to recurrences of the form
	+ $T\left(n\right)=aT\left(n/b\right)+f\left(n\right)$
* where $a\geq 1,b>1$, and $f$ is **asymptotically positive**.

## The Master Method: 3 Cases

(TODO : Add Notes )

* Recurrence: $T\left(n\right)=aT\left(n/b\right)+f\left(n\right)$
* Compare $f\left(n\right)$ with $n^{log\_{b}^{a}}$
* Intuitively:
	+ **Case 1:** $f\left(n\right)$ grows polynomially slower than $n^{log\_{b}^{a}}$
	+ **Case 2:** $f\left(n\right)$ grows at the same rate as $n^{log\_{b}^{a}}$
	+ **Case 3:** $f\left(n\right)$ grows polynomially faster than $n^{log\_{b}^{a}}$

## The Master Method: Case 1 (Bigger)

* Recurrence: $T\left(n\right)=aT\left(n/b\right)+f\left(n\right)$
* *Case 1:* $\frac{n^{log\_{b}^{a}}}{f\left(n\right)}=Ω\left(n^{ε}\right)$ for some constant $ε>0$
* i.e., $f\left(n\right)$ grows polynomialy slower than $n^{log\_{b}^{a}}$ (by an $n^{ε}$ factor)
* **Solution:** $T\left(n\right)=Θ\left(n^{log\_{b}^{a}}\right)$

## The Master Method: Case 2 (Simple Version) (Equal)

* Recurrence: $T\left(n\right)=aT\left(n/b\right)+f\left(n\right)$
* *Case 2:* $\frac{f\left(n\right)}{n^{log\_{b}^{a}}}=Θ\left(1\right)$
* i.e., $f\left(n\right)$ and $n^{log\_{b}^{a}}$ grow at similar rates
* **Solution:** $T\left(n\right)=Θ\left(n^{log\_{b}^{a}}lgn\right)$

## The Master Method: Case 3 (Smaller)

* *Case 3:* $\frac{f\left(n\right)}{n^{log\_{b}^{a}}}=Ω\left(n^{ε}\right)$ for some constant $ε>0$
* i.e., $f\left(n\right)$ grows polynomialy faster than $n^{log\_{b}^{a}}$ (by an $n^{ε}$ factor)
* and the following regularity condition holds:
	+ $af\left(n/b\right)\leq cf\left(n\right)$ for some constant $c<1$
* Solution: $T\left(n\right)=Θ\left(f\left(n\right)\right)$

## The Master Method Example (case-1) : $T\left(n\right)=4T\left(n/2\right)+n$

* $a=4$
* $b=2$
* $f\left(n\right)=n$
* $n^{log\_{b}^{a}}=n^{log\_{2}^{4}}=n^{log\_{2}^{2^{2}}}=n^{2log\_{2}^{2}}=n^{2}$
* $f\left(n\right)=n$ grows polynomially slower than $n^{log\_{b}^{a}}=n^{2}$
	+ $\frac{n^{log\_{b}^{a}}}{f\left(n\right)}=\frac{n^{2}}{n}=n=Ω\left(n^{ε}\right)$
* CASE-1:
	+ $T\left(n\right)=Θ\left(n^{log\_{b}^{a}}\right)=Θ\left(n^{log\_{2}^{4}}\right)=Θ\left(n^{2}\right)$

## The Master Method Example (case-2) : $T\left(n\right)=4T\left(n/2\right)+n^{2}$

* $a=4$
* $b=2$
* $f\left(n\right)=n^{2}$
* $n^{log\_{b}^{a}}=n^{log\_{2}^{4}}=n^{log\_{2}^{2^{2}}}=n^{2log\_{2}^{2}}=n^{2}$
* $f\left(n\right)=n^{2}$ grows at similar rate as $n^{log\_{b}^{a}}=n^{2}$
	+ $f\left(n\right)=Θ\left(n^{log\_{b}^{a}}\right)=n^{2}$
* CASE-2:
	+ $T\left(n\right)=Θ\left(n^{log\_{b}^{a}}lgn\right)=Θ\left(n^{log\_{2}^{4}}lgn\right)=Θ\left(n^{2}lgn\right)$

## The Master Method Example (case-3) (1) : $T\left(n\right)=4T\left(n/2\right)+n^{3}$

* $a=4$
* $b=2$
* $f\left(n\right)=n^{3}$
* $n^{log\_{b}^{a}}=n^{log\_{2}^{4}}=n^{log\_{2}^{2^{2}}}=n^{2log\_{2}^{2}}=n^{2}$
* $f\left(n\right)=n^{3}$ grows polynomially faster than $n^{log\_{b}^{a}}=n^{2}$
	+ $\frac{f\left(n\right)}{n^{log\_{b}^{a}}}=\frac{n^{3}}{n^{2}}=n=Ω\left(n^{ε}\right)$

## The Master Method Example (case-3) (2) : $T\left(n\right)=4T\left(n/2\right)+n^{3}$ (con’t)

* Seems like CASE 3, but need to check the regularity condition
* Regularity condition $af\left(n/b\right)\leq cf\left(n\right)$ for some constant $c<1$
* $4\left(n/2\right)^{3}\leq cn^{3}$ for $c=1/2$
* CASE-3:
	+ $T\left(n\right)=Θ\left(f\left(n\right)\right)$ $⇒$ $T\left(n\right)=Θ\left(n^{3}\right)$

## The Master Method Example (N/A case) : $T\left(n\right)=4T\left(n/2\right)+n^{2}lgn$

* $a=4$
* $b=2$
* $f\left(n\right)=n^{2}lgn$
* $n^{log\_{b}^{a}}=n^{log\_{2}^{4}}=n^{log\_{2}^{2^{2}}}=n^{2log\_{2}^{2}}=n^{2}$
* $f\left(n\right)=n^{2}lgn$ grows slower than $n^{log\_{b}^{a}}=n^{2}$
	+ but is it polynomially slower?
	+ $\frac{n^{log\_{b}^{a}}f\left(n\right)}{=}\frac{n^{2}}{\frac{n^{2}}{lgn}}=lgn\ne Ω\left(n^{ε}\right)$ for any $ε>0$
		- is not CASE-1
		- Master Method does not apply!

## The Master Method : Case 2 (General Version)

* Recurrence : $T\left(n\right)=aT\left(n/b\right)+f\left(n\right)$
* Case 2: $\frac{f\left(n\right)}{n^{log\_{b}^{a}}}=Θ\left(lg^{k}n\right)$ for some constant $k\geq 0$
* Solution : $T\left(n\right)=Θ\left(n^{log\_{b}^{a}}lg^{k+1}n\right)$

## General Method (Akra-Bazzi)

$T\left(n\right)=\sum\_{i=1}^{k}a\_{i}T\left(n/b\_{i}\right)+f\left(n\right)$

Let $p$ be the unique solution to

$\sum\_{i=1}^{k}\left(a\_{i}/b\_{i}^{p}\right)=1$

Then, the answers are the same as for the master method, but with $n^{p}$ instead of $n^{log\_{b}^{a}}$ *(Akra and Bazzi also prove an even more general result.)*

## Idea of Master Theorem (1)

Recursion Tree: 

## Idea of Master Theorem (2)

CASE 1 : The weight increases geometrically from the root to the leaves. The leaves hold a constant fraction of the total weight.

$n^{log\_{b}^{a}}T\left(1\right)=Θ\left(n^{log\_{b}^{a}}\right)$

## Idea of Master Theorem (3)

CASE 2 : $\left(k=0\right)$ The weight is approximately the same on each of the $log\_{b}n$ levels.

$n^{log\_{b}^{a}}T\left(1\right)=Θ\left(n^{log\_{b}^{a}}lgn\right)$

## Idea of Master Theorem (4)

CASE 3 : The weight decreases geometrically from the root to the leaves. The root holds a constant fraction of the total weight.

$n^{log\_{b}^{a}}T\left(1\right)=Θ\left(f\left(n\right)\right)$

## Proof of Master Theorem: Case 1 and Case 2

* Recall from the recursion tree (note $h=lg\_{b}n=tree height$)

$Leaf Cost=Θ\left(n^{log\_{b}^{a}}\right)$ $Non-leaf Cost=g\left(n\right)=\sum\_{i=0}^{h−1}a^{i}f\left(n/b^{i}\right)$

$T\left(n\right)=Leaf Cost+Non-leaf Cost$

$T\left(n\right)=Θ\left(n^{log\_{b}^{a}}\right)+\sum\_{i=0}^{h−1}a^{i}f\left(n/b^{i}\right)$

## Proof of Master Theorem Case 1 (1)

* $\frac{n^{log\_{b}^{a}}}{f\left(n\right)}=Ω\left(n^{ε}\right)$ for some $ε>0$
* $\frac{n^{log\_{b}^{a}}}{f\left(n\right)}=Ω\left(n^{ε}\right)⇒O\left(n^{−ε}\right)⇒f\left(n\right)=O\left(n^{log\_{b}^{a−ε}}\right)$
* $g\left(n\right)=\sum\_{i=0}^{h−1}a^{i}O\left(\left(n/b^{i}\right)^{log\_{b}^{a−ε}}\right)=O\left(\sum\_{i=0}^{h−1}a^{i}\left(n/b^{i}\right)^{log\_{b}^{a−ε}}\right)$
* $O\left(n^{log\_{b}^{a−ε}}\sum\_{i=0}^{h−1}a^{i}b^{iε}/b^{ilog\_{b}^{a−ε}}\right)$

## Proof of Master Theorem Case 1 (2)

* $\sum\_{i=0}^{h−1}\frac{a^{i}b^{iε}}{b^{ilog\_{b}^{a}}}=\sum\_{i=0}^{h−1}a^{i}\frac{\left(b^{ε}\right)^{i}}{\left(b^{log\_{b}^{a}}\right)^{i}}=∑a^{i}\frac{b^{iε}}{a^{i}}=\sum\_{i=0}^{h−1}\left(b^{ε}\right)^{i}$

= An increasing geometric series since $b>1$

$\frac{b^{hε}−1}{b^{ε}−1}=\frac{\left(b^{h}\right)^{ε}−1}{b^{ε}−1}=\frac{\left(b^{log\_{b}^{n}}\right)^{ε}−1}{b^{ε}−1}=\frac{n^{ε}−1}{b^{ε}−1}=O\left(n^{ε}\right)$

## Proof of Master Theorem Case 1 (3)

* $g\left(n\right)=O\left(n^{log\_{b}a−ε}O\left(n^{ε}\right)\right)=O\left(\frac{n^{log\_{b}^{a}}}{n^{ε}}O\left(n^{ε}\right)\right)=O\left(n^{log\_{b}^{a}}\right)$
* $T\left(n\right)=Θ\left(n^{log\_{b}^{a}}\right)+g\left(n\right)=Θ\left(n^{log\_{b}^{a}}\right)+O\left(n^{log\_{b}^{a}}\right)=Θ\left(n^{log\_{b}^{a}}\right)$

**Q.E.D.** (Quod Erat Demonstrandum)

## Proof of Master Theorem Case 2 (limited to k=0)

* $\frac{f\left(n\right)}{n^{l}og\_{b}^{a}}=Θ\left(lg^{0}n\right)=Θ\left(1\right)⇒f\left(n\right)=Θ\left(n^{log\_{b}^{a}}\right)⇒f\left(n/b^{i}\right)=Θ\left(\left(n/b^{i}\right)^{log\_{b}^{a}}\right)$
* $g\left(n\right)=\sum\_{i=0}^{h−1}a^{i}Θ\left(\left(n/b^{i}\right)^{log\_{b}^{a}}\right)$
* $=Θ\left(\sum\_{i=0}^{h−1}a^{i}\frac{n^{log\_{b}^{a}}}{b^{ilog\_{b}^{a}}}\right)$
* $=Θ\left(n^{log\_{b}^{a}}\sum\_{i=0}^{h−1}a^{i}\frac{1}{\left(b^{log\_{b}^{a}}\right)^{i}}\right)$
* $=Θ\left(n^{log\_{b}^{a}}\sum\_{i=0}^{h−1}a^{i}\frac{1}{a^{i}}\right)$
* $=Θ\left(n^{log\_{b}^{a}}\sum\_{i=0}^{log\_{b}^{n−1}}1\right)=Θ\left(n^{log\_{b}^{a}}log\_{b}n\right)=Θ\left(n^{log\_{b}^{a}}lgn\right)$
* $T\left(n\right)=n^{log\_{b}^{a}}+Θ\left(n^{log\_{b}^{a}}lgn\right)$
* $=Θ\left(n^{log\_{b}^{a}}lgn\right)$

**Q.E.D.**

## The Divide-and-Conquer Design Paradigm (1)



alt:“alt” height:500px center

## The Divide-and-Conquer Design Paradigm (2)

1. **Divide** we divide the problem into a number of subproblems.
2. **Conquer** we solve the subproblems recursively.
3. **BaseCase** solve by Brute-Force
4. **Combine** subproblem solutions to the original problem.

## The Divide-and-Conquer Design Paradigm (3)

* $a=subproblem$
* $1/b=each size of the problem$

$$T\left(n\right)=\left\{\begin{matrix}Θ\left(1\right)&if&n\leq c&\left(basecase\right)\\aT\left(n/b\right)+D\left(n\right)+C\left(n\right)&otherwise\end{matrix}\right.$$

Merge-Sort

$$T\left(n\right)=\left\{\begin{matrix}Θ\left(1\right)&&n=1\\2T\left(n/2\right)+Θ\left(n\right)&if&n>1\end{matrix}\right.$$

$T\left(n\right)=Θ\left(nlgn\right)$

## Selection Sort Algorithm

SELECTION-SORT(A)
 n = A.length;
 for j=1 to n-1
 smallest=j;
 for i= j+1 to n
 if A[i]<A[smallest]
 smallest=i;
 endfor
 exchange A[j] with A[smallest]
 endfor

## Selection Sort Algorithm

$$T\left(n\right)=\left\{\begin{matrix}Θ\left(1\right)&&n=1\\T\left(n−1\right)+Θ\left(n\right)&if&n>1\end{matrix}\right.$$

* Sequential Series

$$cost=n\left(n+1\right)/2=1/2n^{2}+1/2n$$

* Drop low-order terms
* Ignore the constant coefficient in the leading term

$$T\left(n\right)=Θ\left(n^{2}\right)$$

## Merge Sort Algorithm (initial setup)

Merge Sort is a recursive sorting algorithm, for initial case we need to call Merge-Sort(A,1,n) for sorting $A\left[1..n\right]$

initial case

A : Array
p : 1 (offset)
r : n (length)
Merge-Sort(A,1,n)

## Merge Sort Algorithm (internal iterations)

internal iterations

$p=start−point$

$q=mid−point$

$r=end−point$

A : Array
p : offset
r : length
Merge-Sort(A,p,r)
 if p=r then (CHECK FOR BASE-CASE)
 return
 else
 q = floor((p+r)/2) (DIVIDE)
 Merge-Sort(A,p,q) (CONQUER)
 Merge-Sort(A,q+1,r) (CONQUER)
 Merge(A,p,q,r) (COMBINE)
 endif

## Merge Sort Combine Algorithm (1)

Merge(A,p,q,r)
 n1 = q-p+1
 n2 = r-q

 //allocate left and right arrays
 //increment will be from left to right
 //left part will be bigger than right part

 L[1...n1+1] //left array
 R[1...n2+1] //right array

 //copy left part of array
 for i=1 to n1
 L[i]=A[p+i-1]

 //copy right part of array
 for j=1 to n2
 R[j]=A[q+j]

 //put end items maximum values for termination
 L[n1+1]=inf
 R[n2+1]=inf

 i=1,j=1
 for k=p to r
 if L[i]<=R[j]
 A[k]=L[i]
 i=i+1
 else
 A[k]=R[j]
 j=j+1

## Example : Merge Sort

1. **Divide:** Trivial.
2. **Conquer:** Recursively sort 2 subarrays.
3. **Combine:** Linear- time merge.
* $T\left(n\right)=2T\left(n/2\right)+Θ\left(n\right)$
	+ Subproblems $⇒2$
	+ Subproblemsize $⇒n/2$
	+ Work dividing and combining $⇒Θ\left(n\right)$

## Master Theorem: Reminder

* $T\left(n\right)=aT\left(n/b\right)+f\left(n\right)$
	+ Case 1: $\frac{n^{log\_{b}^{a}}}{f\left(n\right)}=Ω\left(n^{ε}\right)⇒T\left(n\right)=Θ\left(n^{log\_{b}^{a}}\right)$
	+ Case 2: $\frac{f\left(n\right)}{n^{log\_{b}^{a}}}=Θ\left(lg^{k}n\right)⇒T\left(n\right)=Θ\left(n^{log\_{b}^{a}}lg^{k+1}n\right)$
	+ Case 3: $\frac{n^{log\_{b}^{a}}}{f\left(n\right)}=Ω\left(n^{ε}\right)⇒T\left(n\right)=Θ\left(f\left(n\right)\right)$ and $af\left(n/b\right)\leq cf\left(n\right)$ for $c<1$

## Merge Sort: Solving the Recurrence

$T\left(n\right)=2T\left(n/2\right)+Θ\left(n\right)$ $a=2,b=2,f\left(n\right)=Θ\left(n\right),n^{log\_{b}^{a}}=n$

Case-2: $\frac{f\left(n\right)}{n^{log\_{b}^{a}}}=Θ\left(lg^{k}n\right)⇒T\left(n\right)=Θ\left(n^{log\_{b}^{a}}lg^{k+1}n\right)$ holds for $k=0$

$T\left(n\right)=Θ\left(nlgn\right)$

## Binary Search (1)

Find an element in a sorted array:

**1. Divide:** Check middle element. **2. Conquer:** Recursively search 1 subarray. **3. Combine:** Trivial.

## Binary Search (2)

$$PARENT=⌊i/2⌋$$

$$LEFT-CHILD=2i, 2i>n$$

$$RIGHT-CHILD=2i+1, 2i>n$$

## Binary Search (3) : Iterative

ITERATIVE-BINARY-SEARCH(A,V,low,high)
 while low<=high
 mid=floor((low+high)/2);
 if v == A[mid]
 return mid;
 elseif v > A[mid]
 low = mid + 1;
 else
 high = mid - 1;
 endwhile
 return NIL

## Binary Search (4): Recursive

RECURSIVE-BINARY-SEARCH(A,V,low,high)
 if low>high
 return NIL;
 endif

 mid = floor((low+high)/2);

 if v == A[mid]
 return mid;
 elseif v > A[mid]
 return RECURSIVE-BINARY-SEARCH(A,V,mid+1,high);
 else
 return RECURSIVE-BINARY-SEARCH(A,V,low,mid-1);
 endif

## Binary Search (5): Recursive

$$T\left(n\right)=T\left(n/2\right)+Θ\left(1\right)⇒T\left(n\right)=Θ\left(lgn\right)$$

## Binary Search (6): Example (Find 9)



alt:“alt” height:450px center

## Recurrence for Binary Search (7)

$T\left(n\right)=1T\left(n/2\right)+Θ\left(1\right)$

* Subproblems $⇒1$
* Subproblemsize $⇒n/2$
* Work dividing and combining $⇒Θ\left(1\right)$

## Binary Search: Solving the Recurrence (8)

* $T\left(n\right)=T\left(n/2\right)+Θ\left(1\right)$
* $a=1,b=2,f\left(n\right)=Θ\left(1\right)⇒n^{log\_{b}^{a}}=n^{0}=1$
* **Case 2:** $\frac{f\left(n\right)}{n^{log\_{b}^{a}}}=Θ\left(lg^{k}n\right)⇒T\left(n\right)=Θ\left(n^{log\_{b}^{a}}lg^{k+1}n\right)$ holds for $k=0$
* $T\left(n\right)=Θ\left(lgn\right)$

## Powering a Number: Divide & Conquer (1)

**Problem**: Compute an, where n is a natural number

NAIVE-POWER(a, n)
 powerVal = 1;
 for i = 1 to n
 powerVal = powerVal \* a;
 endfor
return powerVal;

* What is the complexity? $⇒T\left(n\right)=Θ\left(n\right)$

## Powering a Number: Divide & Conquer (2)

* Basic Idea:

$$a^{n}=\left\{\begin{matrix}a^{n/2}\*a^{n/2}&if n is even\\a^{\left(n−1\right)/2}\*a^{\left(n−1\right)/2}\*a&if n is odd\end{matrix}\right.$$

## Powering a Number: Divide & Conquer (3)

POWER(a, n)
 if n = 0 then
 return 1;
 else if n is even then
 val = POWER(a, n/2);
 return val \* val;
 else if n is odd then
 val = POWER(a,(n-1)/2)
 return val \* val \* a;
 endif

## Powering a Number: Solving the Recurrence (4)

* $T\left(n\right)=T\left(n/2\right)+Θ\left(1\right)$
* $a=1,b=2,f\left(n\right)=Θ\left(1\right)⇒n^{log\_{b}^{a}}=n^{0}=1$
* **Case 2:** $\frac{f\left(n\right)}{n^{log\_{b}^{a}}}=Θ\left(lg^{k}n\right)⇒T\left(n\right)=Θ\left(n^{log\_{b}^{a}}lg^{k+1}n\right)$ holds for $k=0$
* $T\left(n\right)=Θ\left(lgn\right)$

## Correctness Proofs for Divide and Conquer Algorithms

* **Proof by induction** commonly used for Divide and Conquer Algorithms
* **Base case:** Show that the algorithm is correct when the recursion bottoms out (i.e., for sufficiently small n)
* **Inductive hypothesis:** Assume the alg. is correct for any recursive call on any smaller subproblem of size $k$, $\left(k<n\right)$
* **General case:** Based on the inductive hypothesis, prove that the alg. is correct for any input of size n

## Example Correctness Proof: Powering a Number

* **Base Case:** $POWER\left(a,0\right)$ is correct, because it returns $1$
* **Ind. Hyp:** Assume $POWER\left(a,k\right)$ is correct for any $k<n$
* **General Case:**
	+ In $POWER\left(a,n\right)$ function:
		- If $n$ is $even$:
			* $val=a^{n/2}$ (due to ind. hyp.)
			* it returns $val\*val=a^{n}$
		- If $n$ is $odd$:
			* $val=a^{\left(n−1\right)/2}$ (due to ind. hyp.)
			* it returns $val\*val\*a=a^{n}$
* The correctness proof is complete

## References

* [Introduction to Algorithms, Third Edition | The MIT Press](https://mitpress.mit.edu/books/introduction-algorithms-third-edition)
* [Bilkent CS473 Course Notes (new)](http://nabil.abubaker.bilkent.edu.tr/473/)
* [Bilkent CS473 Course Notes (old)](http://cs.bilkent.edu.tr/~ugur/teaching/cs473/)
* [Insertion Sort - GeeksforGeeks](https://www.geeksforgeeks.org/insertion-sort/)
* [NIST Dictionary of Algorithms and Data Structures](https://xlinux.nist.gov/dads/)
* [NIST - Dictionary of Algorithms and Data Structures](https://xlinux.nist.gov/dads/)
* [NIST - big-O notation](https://xlinux.nist.gov/dads/HTML/bigOnotation.html)
* [NIST - big-Omega notation](https://xlinux.nist.gov/dads/HTML/omegaCapital.html)

$−End−Of−Week−2−Course−Module−$