COMP 3011
DESIGN AND ANALYSIS OF ALGORITHMS
FALL 2024

Divide and Conquer

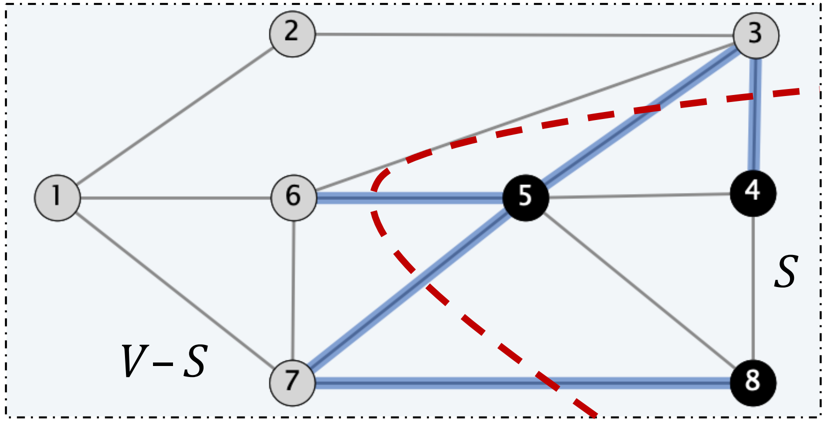
LI Bo
Department of Computing
The Hong Kong Polytechnic University

MINIMUM SPANNING TREES

Basic Definitions

A ***cut*** is a partition of the nodes into two nonempty subsets S and $V - S$, denoted by $(S, V - S)$.

The ***cutset*** of a cut S is the set of edges with exactly one endpoint in S.

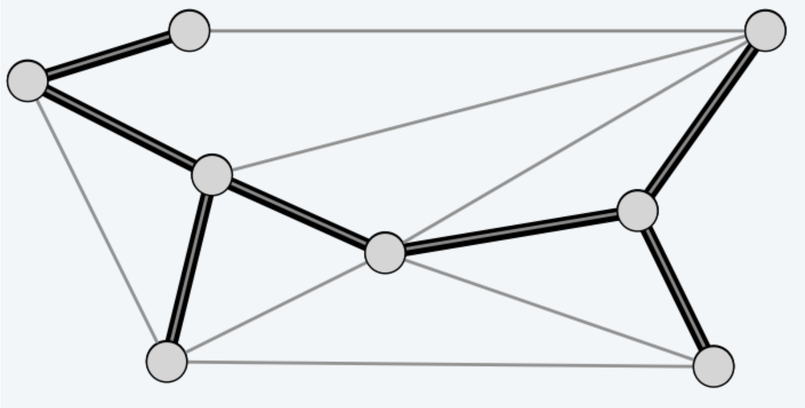


Cut $S = \{4,5,8\}$

Cutset $D = \{(3,4), (3,5), (5,6), (5,7), (8,7)\}$

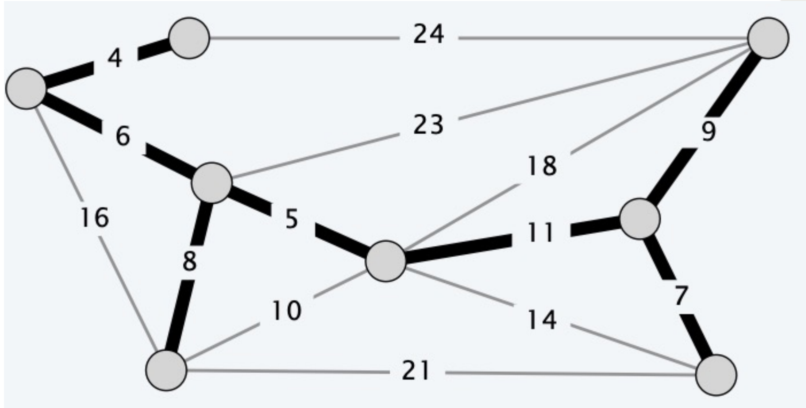
Spanning Tree

Let $H = (V, T)$ be a subgraph of an undirected graph $G = (V, E)$.
H is a ***spanning tree*** of G if H is both acyclic and connected.



Proposition.
Let $H = (V, T)$ be a subgraph of an undirected graph $G = (V, E)$.
Then, the following are equivalent:

- H is a spanning tree of G.
- H is acyclic and connected.
- H is connected and has $|V| - 1$ edges.
- H is acyclic and has $|V| - 1$ edges.
- H is minimally connected: removal of any edge disconnects it.
- H is maximally acyclic: addition of any edge creates a cycle.



Minimum spanning tree (MST)
Given a connected, undirected graph $G = (V, E)$ with edge costs c_e, a ***minimum spanning tree*** (V, T) is a spanning tree of G such that the sum of the edge costs in T is minimized.

Tree cost $= 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7$

Minimum spanning tree (MST)

Cayley's theorem. The complete graph on n nodes has n^{n-2} spanning trees.

↓

can't solve by brute force

Both give the optimal solution!

Kruskal's Algorithm

Idea.

- Starts without any edges and insert edges from E in order of increasing cost:

$$c_1 < c_2 < \cdots < c_i < \cdots < c_m$$

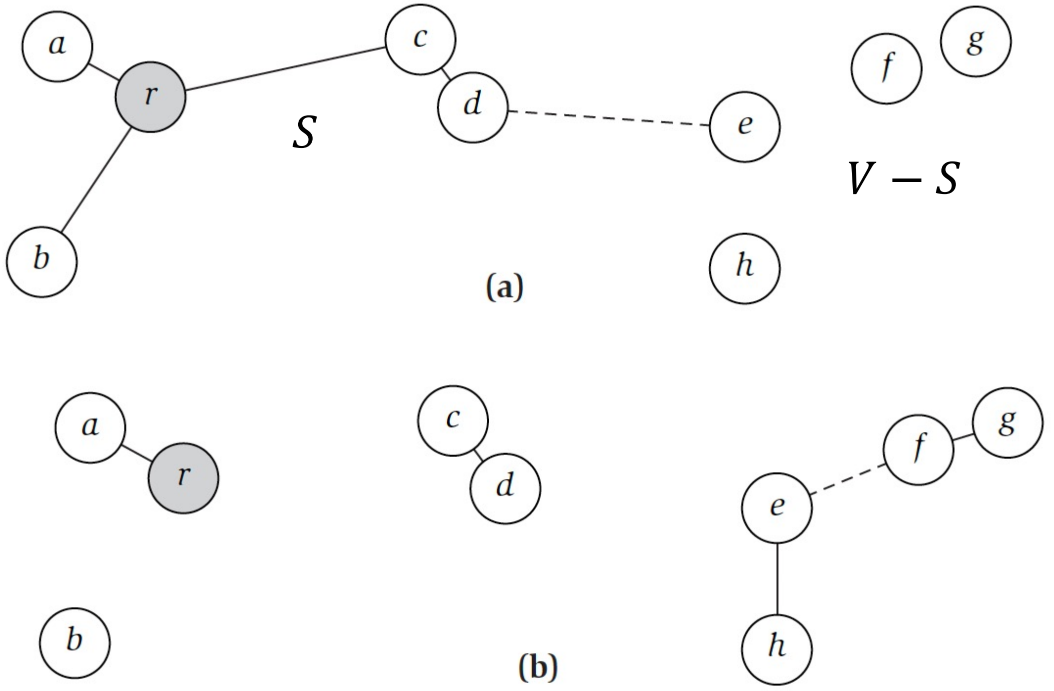
- For edge e_i, insert it if it does not create a cycle with all inserted edges, and discard otherwise.

Prim's Algorithm

Idea (inspired by Dijkstra's Algorithm).

- Start with a root node $S = \{s\}$, and try to greedily grow a tree from S outward.
- At each step, we add the node v connected with S that can be attached as cheaply as possibly.

$$\min_{e=(u,v):u\in S} c_e$$



(a)

(b)

When Is It Safe to Include an Edge in the Minimum Spanning Tree?

Cut Property (Assume that all edge costs are distinct.)
Let S be ***any*** subset of nodes $S \neq V$ or $\emptyset$.
Let edge $e = (v, w)$ be the minimum cost edge with one end in S and the other in $V - S$.
Then ***every*** MST contains the edge e.

Proof. [contradiction + exchange argument]

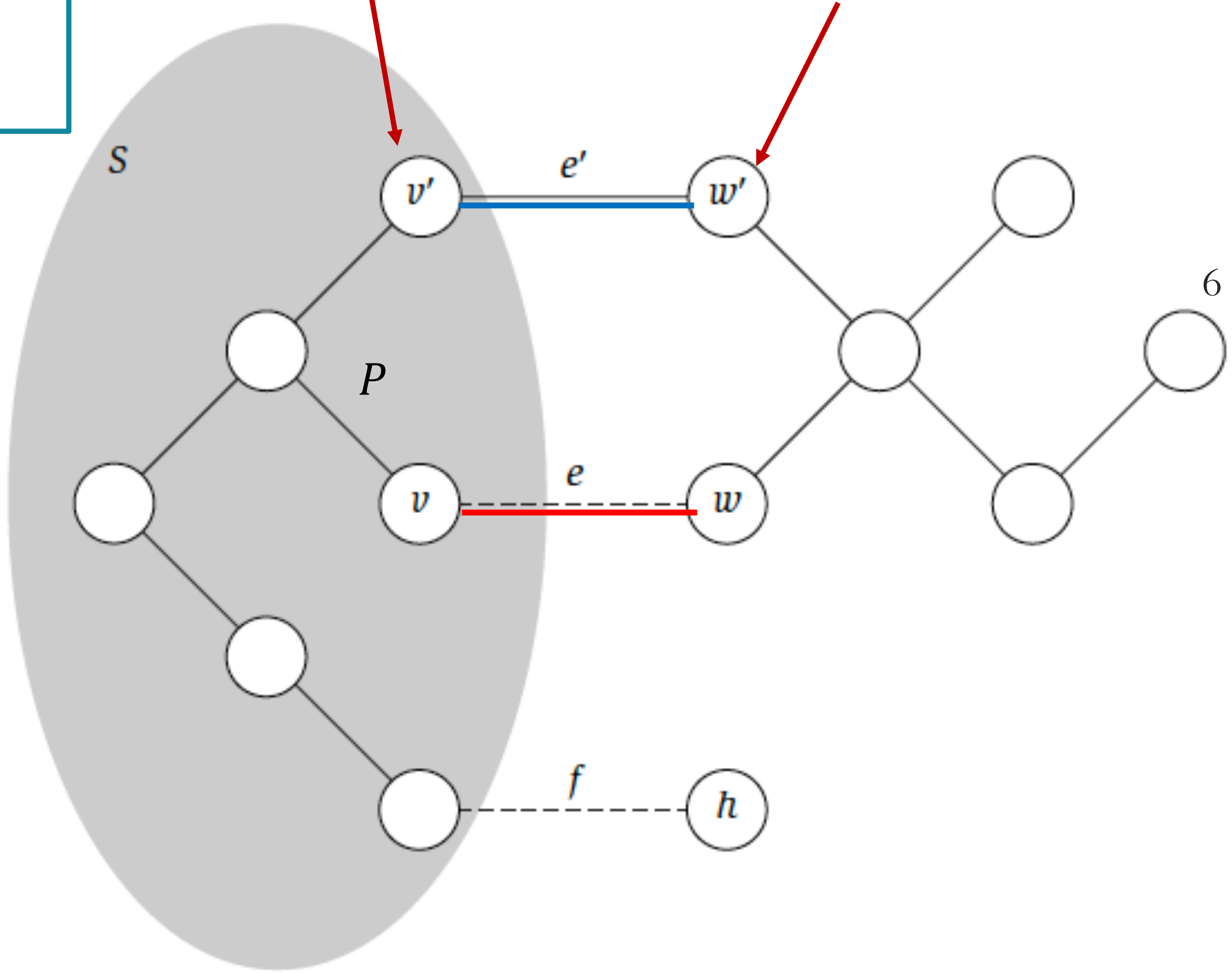
- Let T be an MST that does not contain e.
- There must be a path P in T from v to w.
- Exchange e' for e, get a set of edges

$$T' = T - \{e'\} \cup \{e\}.$$

- T' is a spanning tree:
  - Connected: any path in (V, T) that used e' can now be "rerouted" by using e.
  - Contains $|V| - 1$ edges.
- $c_e < c_e$: cost of T' < cost of T -- a contradiction.

$v' \in S$ is the node just before w' on P

first node w' on P that is in $V - S$.

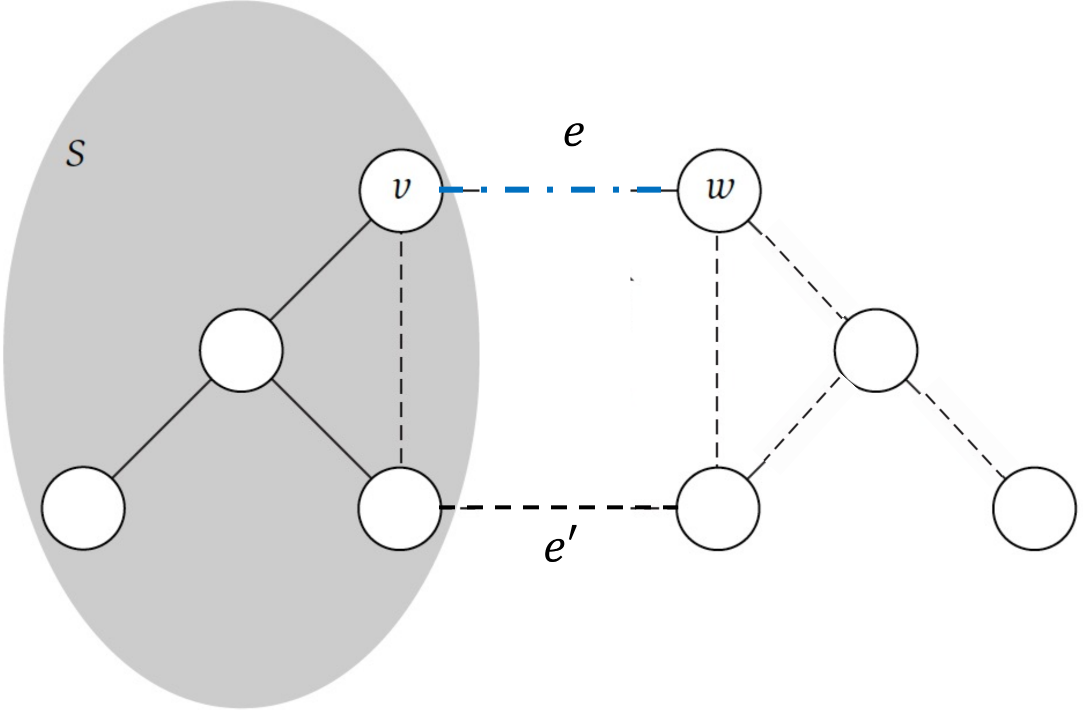


Prim's Algorithm

- Start with a root node $S = \{s\}$, and try to greedily grow a tree from S outward.
- At each step, we add the node v connected with S that can be attached as cheaply as possibly.

$$\min_{e=(u,v):u\in S} c_e$$

Theorem. Prim's Algorithm produces an MST of G.



Prim's Algorithm outputs a spanning tree.

- Contains no cycles: by the design
- Connected: otherwise can add an edge between two components.

Prim's Algorithm outputs an MST.

- At each step, we add the node v connected with S that can be attached as cheaply as possibly.

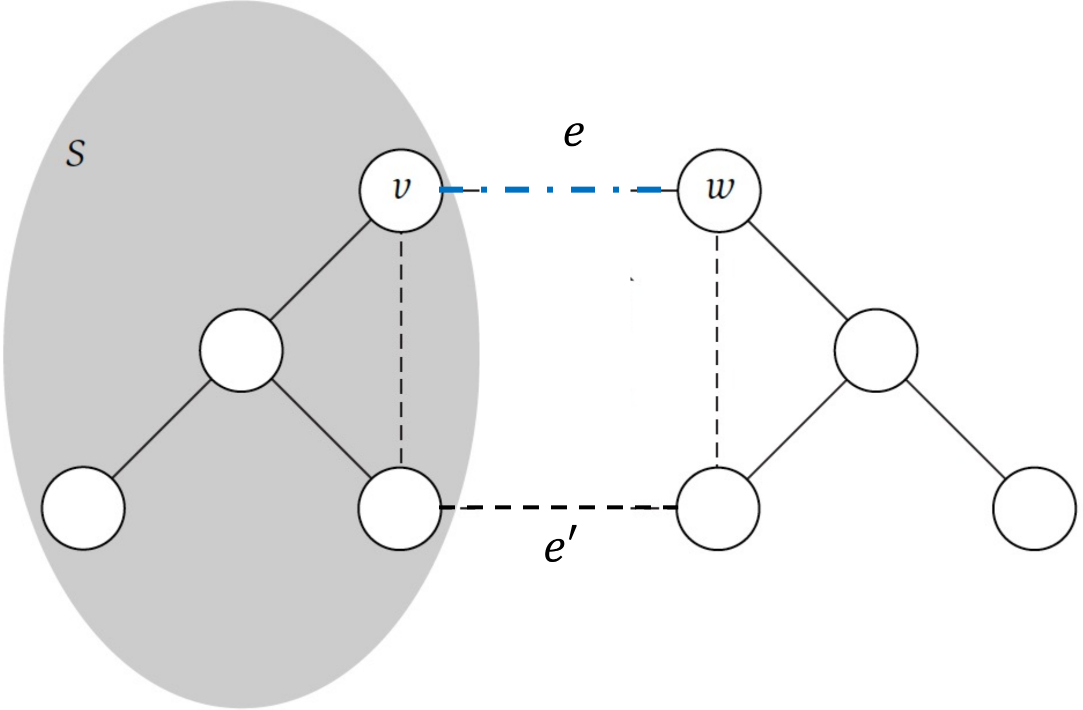
$$\min_{e=(u,v):u\in S} c_e$$

- Thus, e is the cheapest edge connecting S and $V - S$.
- By ***Cut Property***, e belongs to every MST.

Kruskal's Algorithm

- Starts without any edges and insert edges from E in order of increasing cost:
$$c_1 < c_2 < \cdots < c_i < \cdots < c_m$$
- For edge e_i insert it if it does not create a cycle with all inserted edges, and discard otherwise.

Theorem. Kruskal's Algorithm produces an MST of G.



Kruskal's Algorithm outputs a spanning tree.

- Contains no cycles: by the design
- Connected; otherwise can add an edge between two components.

Kruskal's Algorithm outputs an MST.

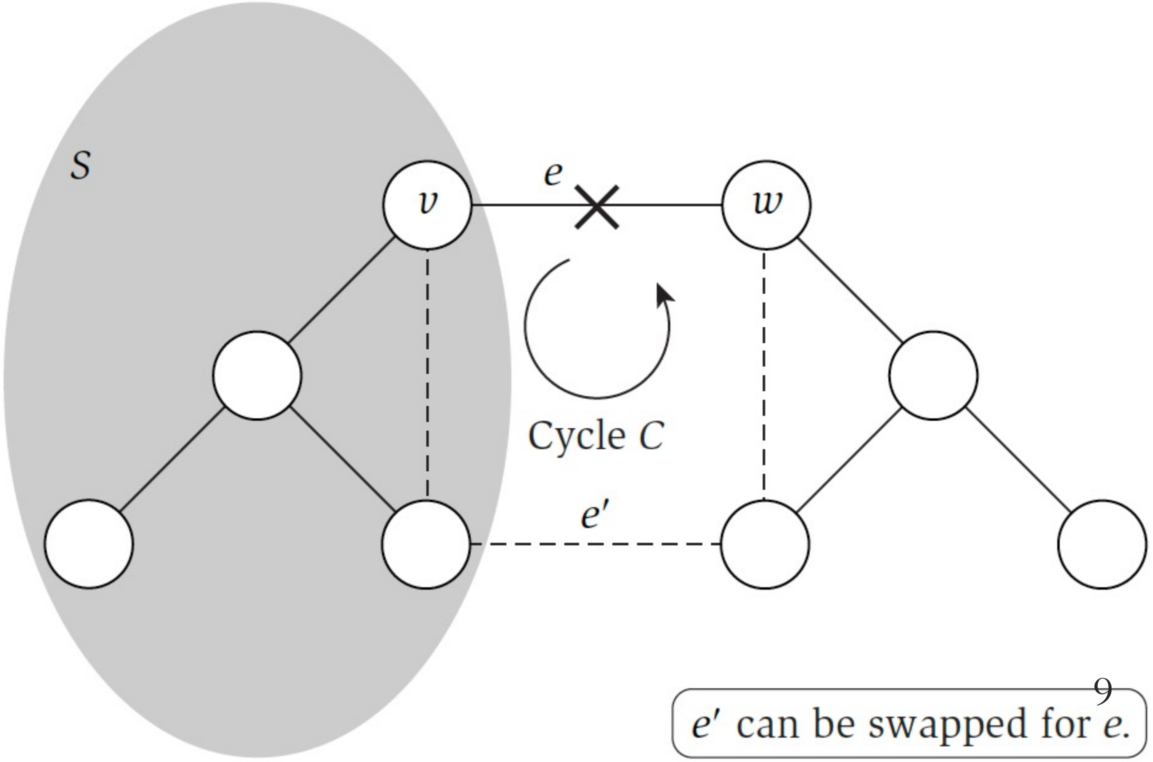
- Consider any edge $e = (v, w)$ added by Kruskal's Algorithm.
- Let S be the set of nodes to which v has a path before e is added. Clearly $v \in S$, but $w \notin S$.
- No edge from S to $V - S$ has been considered: any such edge could have been added without creating a cycle.
- Thus, e is the cheapest edge connecting S and $V - S$.
- By ***Cut Property***, e belongs to every MST.

Reverse-Delete Algorithm

- Start with the full graph (V, E) and begin deleting edges in order of decreasing cost.

$$c_1 > c_2 > \cdots > c_i > \cdots > c_m$$

- As we get to each edge e (starting from the most expensive), we delete it as long as doing so would not actually disconnect the graph we currently have.



Theorem.
The Reverse-Delete Algorithm produces an MST of G.

Cycle Property
(Assume that all edge costs are distinct.)
Let C be any cycle in G.
Let edge $e = (v, w)$ be the most expensive edge on C.
Then e does not belong to any MST of G.

Proof [by contradiction].

- Let T be an MST that contains $e = (v, w)$.
- Deleting e from T and partition the nodes into S and $V - S$.
- There is another edge e' crosses from S to $V - S$.
- Consider the set of edges

$$T = T - \{e\} \cup \{e'\}$$

which is a spanning tree of G with smaller cost.

DIVIDE AND CONQUER

Divide and Conquer

- ➢ Divide up problem into several subproblems (of the same kind).
- ➢ Solve (conquer) each subproblem recursively.
- ➢ Combine solutions to subproblems into overall solution

Most common usage:

- ➢ Divide problem of size n into two subproblems of size $n/2$.
- ➢ Solve (conquer) each subproblem recursively.
- ➢ Combine two solutions into overall solution.

Consequence:

- ➢ Brute force: $\Theta(n^2)$.
- ➢ Divide-and-conquer: $O(n\log n)$.

Brute-force algorithm may already be polynomial time, and the divide and conquer strategy is to reduce the running time to a lower polynomial.

THE MERGESORT ALGORITHM

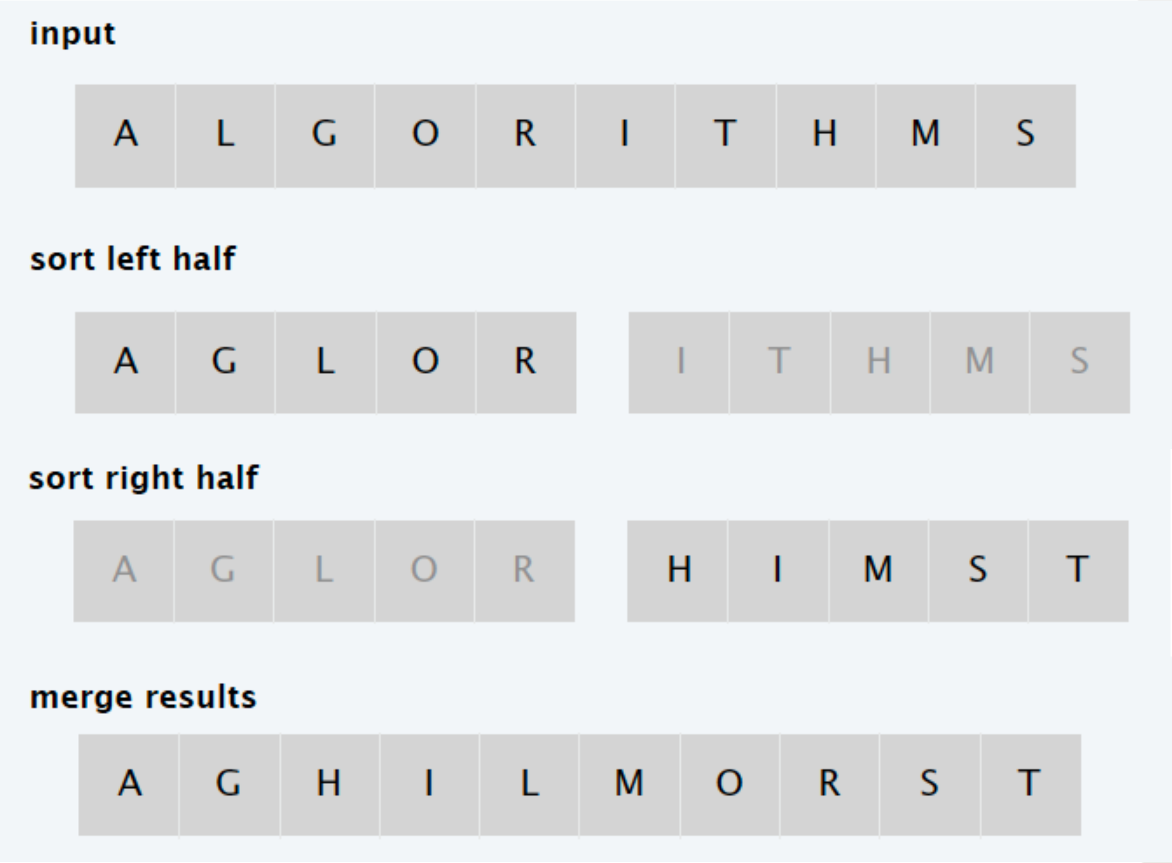
The Mergesort Algorithm

Problem. Given a list L of n elements from an ordered universe, rearrange them in ascending order.

The algorithm

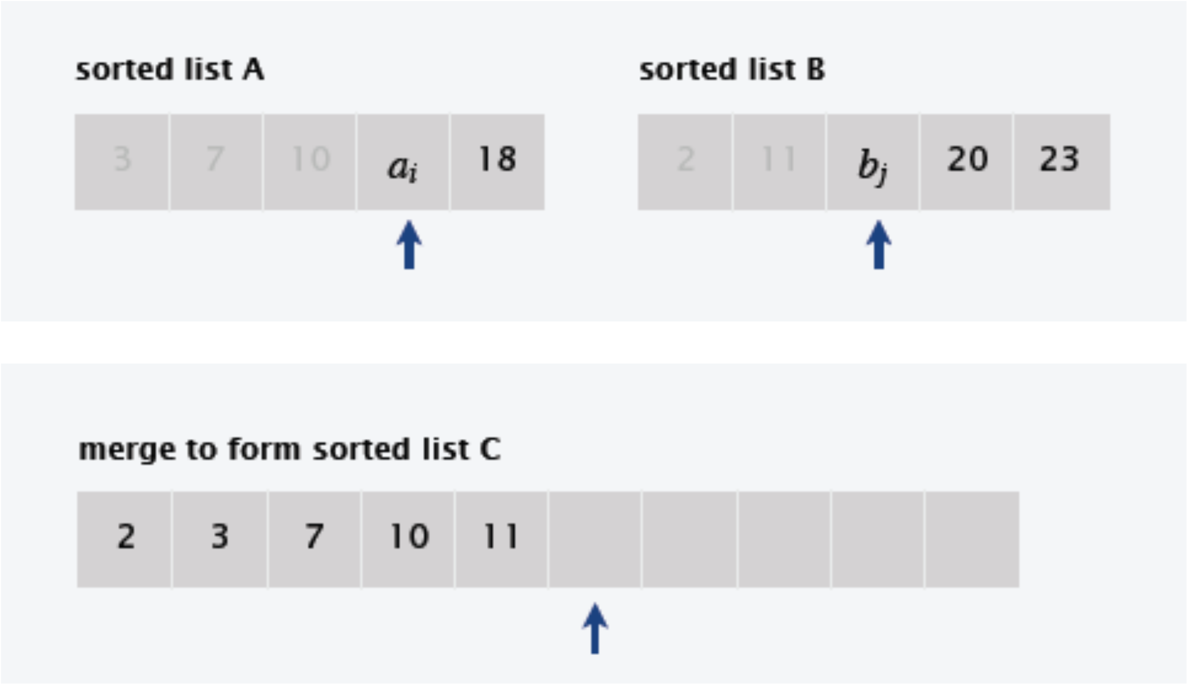
- Divide into left and right smaller problems.
- Recursively sort left half.
- Recursively sort right half.
- Merge two halves to make sorted whole.

How to do this?



The Mergesort Algorithm

Goal. Combine two sorted lists A and B into a sorted whole C.



The algorithm

- Scan A and B from left to right.
- Compare a_i and b_j.
- If $a_i < b_j$, append a_i to C (no larger than any remaining element in B).
- If $a_i > b_j$, append b_j to C (smaller than every remaining element in A).

$\Theta(n)$

The Mergesort Algorithm

Definition. $T(n)$ = max number of compares to Mergesort a list of length n.

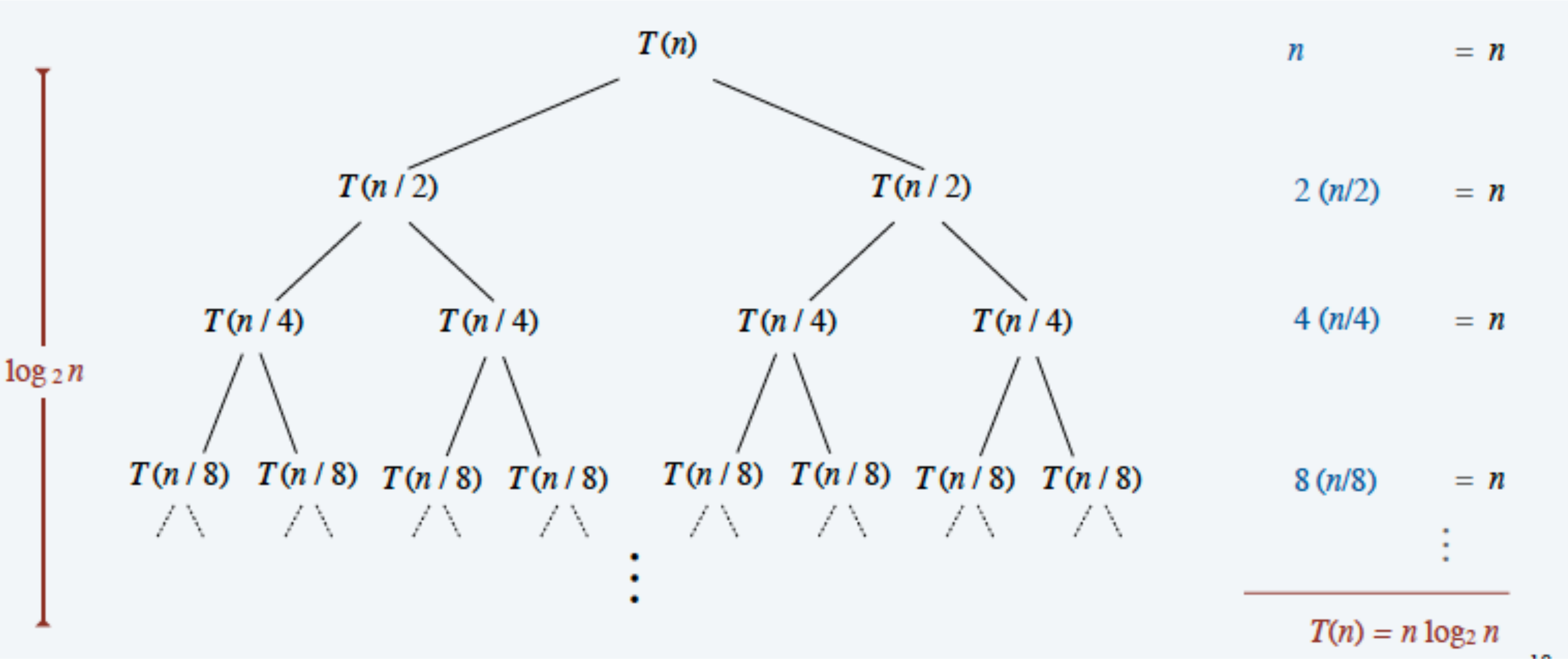
$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n & \text{if } n > 1 \end{cases}$$

Solving this recurrence: assume n is a power of 2 and replace $\leq$ with $=$ in the recurrence.

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{if } n > 1 \end{cases}$$

The Mergesort Algorithm

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{if } n > 1 \end{cases}$$



The Mergesort Algorithm

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) \; + \; n & \text{if } n > 1 \end{cases}$$

Proposition. If $T(n)$ satisfies the recurrence, then $T(n) = n\log_2 n$.

Proof. [by induction on n]

- ***Base case***: when $n = 1, T(1) = 0 = \; n\log_2 n$.
- ***Inductive hypothesis***: assume $T(n) = n\log_2 n$.
- ***Goal***: show that $T(2n) = 2n\log_2 2n$.

What if n is not a power of 2??

$$\begin{aligned} T(2n) &= 2T(n) + 2n \\ &= 2n\log_2 n + 2n \\ &= 2n[\log_2 2n - 1] + 2n \\ &= 2n\log_2 2n \end{aligned}$$

inductive hypothesis →

The Mergesort Algorithm

Proposition. If $T(n)$ satisfies the recurrence, then $T(n) \le n\lceil \log_2 n \rceil$.

$$T(n) \le \begin{cases} 0 & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n & \text{if } n > 1 \end{cases}$$

no longer assuming n is a power of 2

Proof*. [by induction on n]

- ***Base case***: $n = 1$
- Define $n_1 = \lfloor n/2 \rfloor$ and $n_2 = \lceil n/2 \rceil$ and note that $n = n_1 + n_2$.
- ***Induction step***: assume true for $1, 2, \ldots, n-1$.

$$\begin{aligned} T(n) &\le T(n_1) + T(n_2) + n \\ &\le n_1 \lceil \log_2 n_1 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\ &\le n_1 \lceil \log_2 n_2 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\ &= n\lceil \log_2 n_2 \rceil + n \le n(\lceil \log_2 n \rceil - 1) + n = n\lceil \log_2 n \rceil \end{aligned}$$

$$\begin{aligned} n_2 &= \lceil n/2 \rceil \\ &\le \left\lceil 2^{\lceil \log_2 n \rceil} / 2 \right\rceil \\ &= 2^{\lceil \log_2 n \rceil} / 2 \end{aligned}$$

$$\log_2 n_2 \le \lceil \log_2 n \rceil - 1$$

an integer

COUNTING INVERSIONS

Counting inversions

Similarity metric: number of inversions between two rankings.

- My rank: $1, 2, \ldots, n$.
- Your rank: $a_1, a_2, \cdots, a_n$.
- Items i and j are inverted if $i < j$, but $a_i > a_j$.

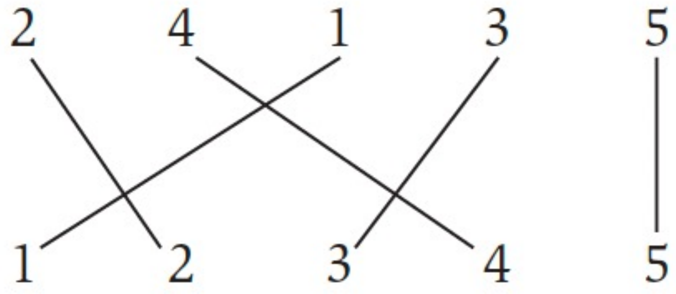
Brute force: check all $\Theta(n^2)$ pairs.

	A	B	C	D	E
me	1	2	3	4	5
you	1	3	4	2	5

2 inversions: 3-2 and 4-2

Divide-and-Conquer:

- Divide: separate list into two halves A and B.
- Conquer: recursively count inversions in each list.
- Combine: count inversions (a, b) with a ∈ A and b ∈ B.
- Return sum of three counts.



- Three inversions in this sequence: (2, 1), (4, 1), and (4, 3).
- Each crossing corresponds to one inversion.

Counting inversions

input

1	5	4	8	10	2	6	9	3	7

Divide-and-Conquer:

- Divide: separate list into two halves A and B.
- Conquer: recursively count inversions in each list.
- Combine: count inversions (a, b) with a ∈ A and b ∈ B. → How to execute this step?
- Return sum of three counts.

count inversions in left half A

1	5	4	8	10

5-4

count inversions in right half B

2	6	9	3	7

6-3 9-3 9-7

count inversions (a, b) with a ∈ A and b ∈ B

1	5	4	8	10	2	6	9	3	7

4-2 4-3 5-2 5-3 8-2 8-3 8-6 8-7 10-2 10-3 10-6 10-7 10-9

In total: 1 + 3 + 13 = 17

Counting inversions

How to execute this step?

Divide-and-Conquer:

- Divide: separate list into two halves A and B.
- Conquer: recursively count inversions in each list.
- Combine: count inversions (a, b) with a ∈ A and b ∈ B.
- Return sum of three counts.

list A

7	10	18	3	14

list B

20	23	2	11	16

Warmup algorithm.

- Sort A and B.
- For each element b ∈ B, binary search in A to find how elements in A are greater than b.

sort A

3	7	10	14	18

sort B

2	11	16	20	23

binary search to count inversions (a, b) with a ∈ A and b ∈ B

3	7	10	14	18

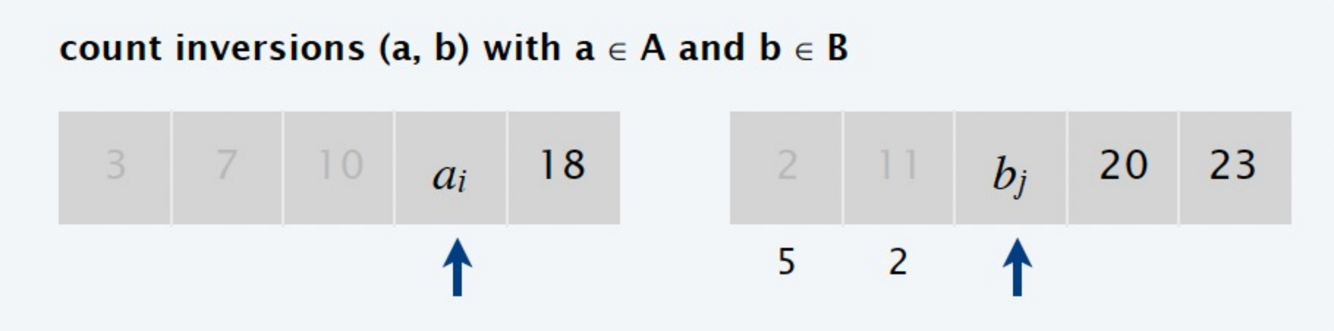
2	11	16	20	23
5	2	1	0	0

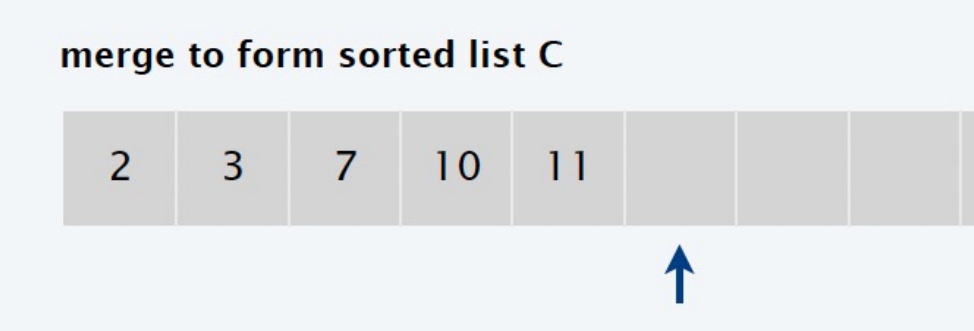
Can be better?

Merge-and-Count

Goal: Count inversions (a, b) with a ∈ A and b ∈ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
- If $a_i < b_j$, then a_i is not inverted with any element left in B.
- If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.





Counting inversions: divide-and-conquer algorithm

Input. List L.
Output. Number of inversions in L and L in sorted order.

SORT-AND-COUNT(L)

IF (list L has one element)

    RETURN $(0, L)$.

Divide the list into two halves A and B.

$(r_A,\ A) \leftarrow$ SORT-AND-COUNT(A). $\longleftarrow T(n / 2)$

$(r_B,\ B) \leftarrow$ SORT-AND-COUNT(B). $\longleftarrow T(n / 2)$

$(r_{AB},\ L) \leftarrow$ MERGE-AND-COUNT(A, B). $\longleftarrow \Theta(n)$

RETURN $(r_A + r_B + r_{AB},\ L)$.

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

Running time = $O(n \log n)$

INTEGER MULTIPLICATION

INTEGER ADDITION AND SUBTRACTION

- ***Addition***. Given two n-bit integers a and b, compute $a + b$.
- ***Subtraction***. Given two n-bit integers a and b, compute $a - b$.

Grade-school algorithm. $\Theta(n)$ bit operations. → "bit complexity" (instead of word time complexity)

1	1	1	1	1	1	0	1	
	1	1	0	1	0	1	0	1
+	0	1	1	1	1	1	0	1
1	0	1	0	1	0	0	1	0

Remark. Grade-school addition and subtraction algorithms are optimal.

INTEGER MULTIPLICATION

- ***Multiplication***. Given two n-bit integers a and b, compute $a \times b$.
- ***Grade-school algorithm*** (long multiplication). $\Theta(n^2)$ bit operations.

$$
\begin{array}{r}
12 \\
\times\, 13 \\
\hline
36 \\
12 \\
\hline
156
\end{array}
\qquad\qquad
\begin{array}{r}
1100 \\
\times\, 1101 \\
\hline
1100 \\
0000 \\
1100 \\
1100 \\
\hline
10011100
\end{array}
$$

Divide-and-Conquer?

DIVIDE-AND-CONQUER MULTIPLICATION

Let's assume we're in base-2.

- Write x as $x = x_1 \cdot 2^{n/2} + x_0$. In other words, x_1 corresponds to the "high-order" $n/2$ bits, and x_0 corresponds to the "low-order" $n/2$ bits.
- Similarly, write $y = y_1 \cdot 2^{n/2} + y_0$.

Ex. $x = \underbrace{1\,0\,0\,0}_{x_1}\,\underbrace{1\,1\,0\,1}_{x_0} \qquad y = \underbrace{1\,1\,1\,0}_{y_1}\,\underbrace{0\,0\,0\,1}_{y_0}$

Thus, we have

$$\begin{aligned} xy &= (x_1 \cdot 2^{n/2} + x_0)(y_1 \cdot 2^{n/2} + y_0) \\ &= x_1 y_1 \cdot 2^n + (x_1 y_0 + x_0 y_1) \cdot 2^{n/2} + x_0 y_0. \end{aligned}$$

$$(x_1 + x_0)(y_1 + y_0) = \underbrace{x_1 y_1} + \underbrace{x_1 y_0 + x_0 y_1} + \underbrace{x_0 y_0}$$

Warm-up Algorithm

Reduce the problem of solving a single n-bit instance to the problem of solving four $n/2$-bit instances:

- $x_1 y_1, x_1 y_0, x_0 y_1, x_0 y_0$.
- Combining of the solution requires a constant number of additions of $O(n)$-bit numbers.
- $T(n) \le 4T(n/2) + cn$. $\quad O(n^2)$

$$T(n) \le 3T\left(\frac{n}{2}\right) + cn = O(n^{1.59})$$

Master Theorem

DIVIDE-AND-CONQUER MULTIPLICATION

```
Recursive-Multiply(x,y):
```

Write $x = x_1 \cdot 2^{n/2} + x_0$

$y = y_1 \cdot 2^{n/2} + y_0$

Compute $x_1 + x_0$ and $y_1 + y_0$ ← $O(n)$

p = Recursive-Multiply($x_1 + x_0,\ y_1 + y_0$)

x_1y_1 = Recursive-Multiply(x_1, y_1)

x_0y_0 = Recursive-Multiply(x_0, y_0)

Return $x_1y_1 \cdot 2^n + (p - x_1y_1 - x_0y_0) \cdot 2^{n/2} + x_0y_0$ ← $O(n)$

Theorem. The running time of Recursive-Multiply on two n-bit factors is $O(n^{\log_2 3}) = O(n^{1.59})$.

MASTER THEOREM

MASTER THEOREM

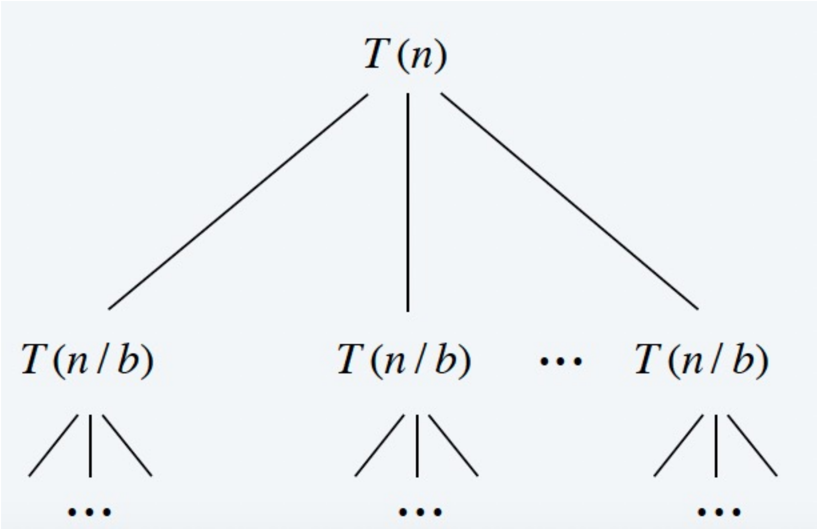
Goal. To solve divide-and-conquer recurrences:

$$T(n) = a \cdot T(\frac{n}{b}) + f(n)$$

with $T(0) = 0$ and $T(1) = \Theta(1)$.

Terms.

- $a \geq 1$ is the number of subproblems.
- $b \geq 2$ is the factor by which the subproblem size decreases.
- $f(n) \geq 0$ is the work to divide and combine subproblems.



MASTER THEOREM

Proof is left for exercise.

Master theorem. Let $a \geq 1$, $b \geq 2$, and $c \geq 0$ and suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) \;=\; a\,T\left(\frac{n}{b}\right) \;+\; \Theta(n^c)$$

with $T(0) = 0$ and $T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

Case 1. If $c > \log_b a$, then $T(n) = \Theta(n^c)$.

Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.

Case 3. If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

Mergesort: $T(n) \leq 2T(\frac{n}{2}) + \Theta(n)$ ⟶ $c = 1 = \log_b a = \log_2 2 = 1 \Rightarrow T(n) = \Theta(n \log n)$

Integer Multiplication I: $T(n) \leq 4T(\frac{n}{2}) + \Theta(n)$ ⟶ $c = 1 < \log_b a = \log_2 4 = 2 \Rightarrow T(n) = \Theta(n^2)$

Integer Multiplication II: $T(n) \leq 3T\left(\frac{n}{2}\right) + \Theta(n)$ ⟶ $c = 1 < \log_b a = \log_2 3 = 1.59 \Rightarrow T(n) = \Theta(n^{1.59})$

MATRIX MULTIPLICATION*

MATRIX MULTIPLICATION

Can we improve this?

Matrix multiplication. Given two n-by-n matrices A and B, compute $C = AB$.

Grade-school. $\Theta(n^3)$ arithmetic operations.

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

$$\begin{bmatrix} .59 & .32 & .41 \\ .31 & .36 & .25 \\ .45 & .31 & .42 \end{bmatrix} = \begin{bmatrix} .70 & .20 & .10 \\ .30 & .60 & .10 \\ .50 & .10 & .40 \end{bmatrix} \times \begin{bmatrix} .80 & .30 & .50 \\ .10 & .40 & .10 \\ .10 & .30 & .40 \end{bmatrix}$$

MATRIX MULTIPLICATION

C_{11} A_{11} A_{12} B_{11}

$$\begin{bmatrix} 152 & 158 & 164 & 170 \\ 504 & 526 & 548 & 570 \\ 856 & 894 & 932 & 970 \\ 1208 & 1262 & 1316 & 1370 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix} \times \begin{bmatrix} 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 \\ 28 & 29 & 30 & 31 \end{bmatrix}$$

B_{21}

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 16 & 17 \\ 20 & 21 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 24 & 25 \\ 28 & 29 \end{bmatrix} = \begin{bmatrix} 152 & 158 \\ 504 & 526 \end{bmatrix}$$

"$(x_1 + x_0)(y_1 + y_0) = x_1 y_1 + x_1 y_0 + x_0 y_1 + x_0 y_0$"

MATRIX MULTIPLICATION

To multiply two n-by-n matrices A and B:

- ***Divide***: partition A and B into ½n-by-½n blocks.
- ***Conquer***: multiply 8 pairs of ½n-by-½n matrices.
- ***Combine***: 4 matrix additions

n-by-n matrices

$$C = A \times B$$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

½n-by-½n matrices

Running time.

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$$

$$c = 2 < \log_b a = \log_2 8 = 3 \Rightarrow T(n) = \Theta(n^3)$$

8 matrix multiplications
(of ½n-by-½n matrices)

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

4 matrix additions
(of ½n-by-½n matrices)

MATRIX MULTIPLICATION

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$C_{11} = P_5 + P_4 - P_2 + P_6$$

$$C_{12} = P_1 + P_2$$

$$C_{21} = P_3 + P_4$$

$$C_{22} = P_1 + P_5 - P_3 - P_7$$

11 additions and 7 subtractions

$$P_1 \leftarrow A_{11} \times (B_{12} - B_{22})$$

$$P_2 \leftarrow (A_{11} + A_{12}) \times B_{22}$$

$$P_3 \leftarrow (A_{21} + A_{22}) \times B_{11}$$

$$P_4 \leftarrow A_{22} \times (B_{21} - B_{11})$$

$$P_5 \leftarrow (A_{11} + A_{22}) \times (B_{11} + B_{22})$$

$$P_6 \leftarrow (A_{12} - A_{22}) \times (B_{21} + B_{22})$$

$$P_7 \leftarrow (A_{11} - A_{21}) \times (B_{11} + B_{12})$$

7 multiplications
of ½n-by-½n matrices

Running time.

$$T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2)$$

$$c = 2 < \log_b a = \log_2 7 = 2.81 \Rightarrow T(n) = \Theta(n^{2.81})$$

Thank You!