Chapter 1
Divide and Conquer

algorithm design principle

Algorithm Theory
WS 2014/15

Fabian Kuhn

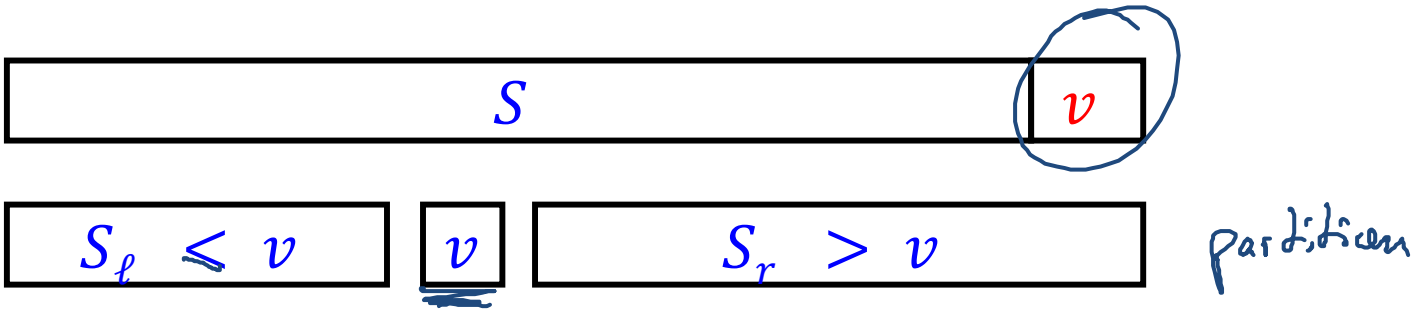
Divide-And-Conquer Principle

- Important algorithm design method

- Examples from Informatik 2:
  - Sorting: Mergesort, Quicksort
  - Binary search can be considered as a divide and conquer algorithm

- Further examples
  - Median, Selection
  - Compairing orders
  - Delaunay triangulation / Voronoi diagram
  - Closest pairs
  - Line intersections
  - Polynomial multiplication / FFT
  - ...

Example 1: Quicksort



function Quick (S: sequence): sequence;

{returns the sorted sequence S}

begin

if $\#S \leq 1$ then **return** S

else { choose pivot element v in S;

partition S into S_ℓ with elements $< v$,

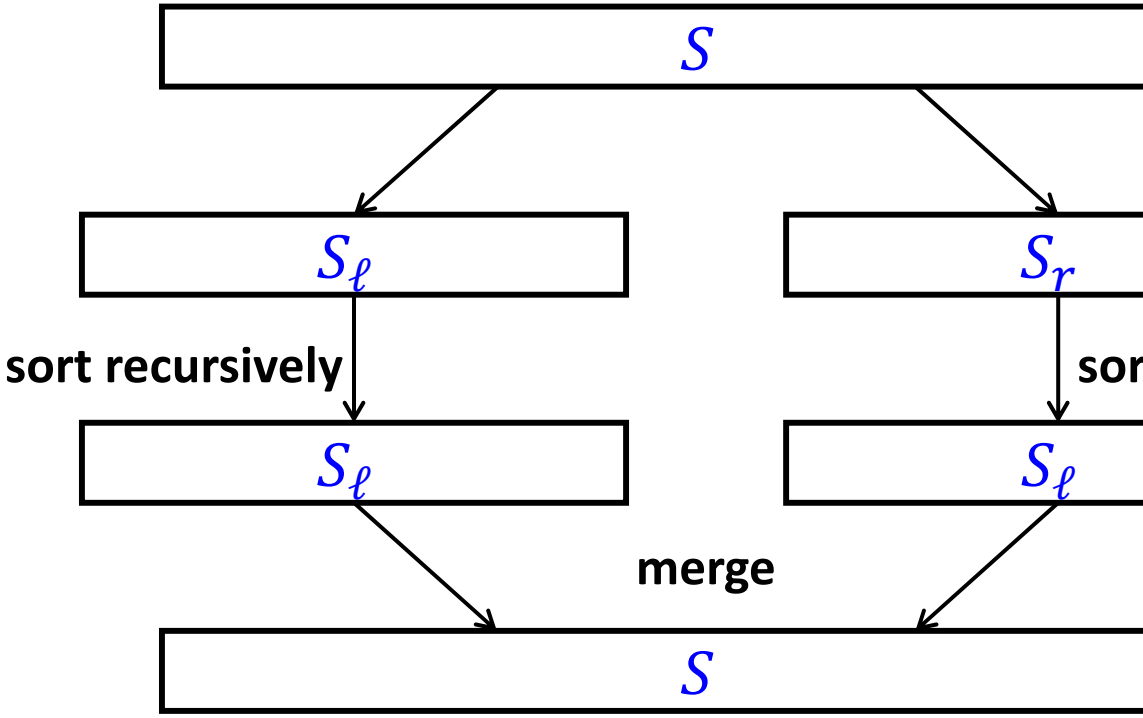
and S_r with elements $> v$ } partition

return Quick(S_ℓ) v Quick(S_r)

rec. rec.

end;

Example 2: Mergesort



Formulation of the D&C principle

Divide-and-conquer method for solving a problem instance of size n:

1. Divide

$n \leq c$: Solve the problem directly.

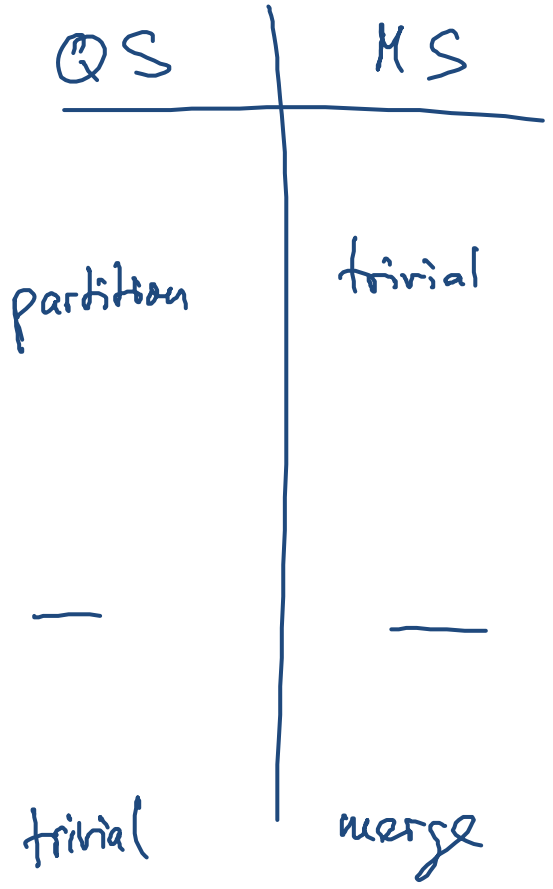
$n > c$: Divide the problem into k subproblems of sizes $n_1, \dots, n_k < n$ ($k \geq 2$).

2. Conquer

Solve the k subproblems in the same way (recursively).

3. Combine

Combine the partial solutions to generate a solution for the original instance.



Analysis

Recurrence relation:

- $T(n)$: max. number of steps necessary for solving an instance of size n
- $$T(n) = \begin{cases} a & \text{if } n \leq c \\ T(n_1) + \cdots + T(n_k) + \text{cost for divide and combine} & \text{if } n > c \end{cases}$$

Special case: $k = 2, n_1 = n_2 = n/2$

$n_1 = \lfloor n/2 \rfloor$, $n_2 = \lceil n/2 \rceil$

- cost for divide and combine: $\mathrm{DC}(n)$
- $T(1) = a$
- $T(n) = 2T(n/2) + \mathrm{DC}(n)$

mergesort : $T(n) = 2 \cdot T(n/2) + O(n)$

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

Analysis, Example

Recurrence relation: what is $T(n)$?

$$T(n) \leq 2 \cdot T(n/2) + cn^2, \qquad T(1) \leq a$$

Guess the solution by repeated substitution:

$$T(n) \leq 2T(n/2) + cn^2$$

$$\leq 2\left(2 \cdot T(n/4) + c \cdot \left(\tfrac{n}{2}\right)^2\right) + cn^2$$

$$= 4T(n/4) + c\left(1+\tfrac{1}{2}\right) \cdot n^2$$

$$\leq 4\left(2 \cdot T(n/8) + c\left(\tfrac{n}{4}\right)^2\right) + c\left(1+\tfrac{1}{2}\right) \cdot n^2$$

$$= 8T(n/8) + c\left(1+\tfrac{1}{2}+\tfrac{1}{4}\right) n^2$$

$\vdots$ ← guessing

$$\leq 2^i T(n/2^i) + c\left(1+\tfrac{1}{2}+\ldots+\tfrac{1}{2^{i-1}}\right) n^2 < n \cdot T(1) + 2cn^2$$

($1+\frac{1}{2}+\ldots+\frac{1}{2^{i-1}} < 2$; stop when $= 1$)

$$\leq an + 2cn^2$$

Analysis, Example

guess: $T(n) < a \cdot n + 2cn^2$ (*)

Recurrence relation:

$$T(n) \leq 2 \cdot T(n/2) + cn^2, \qquad T(1) \leq a$$

Verify by induction:

Ind. Base: $n = 1$ $\quad T(1) \leq a$, $\quad T(1) < a \cdot 1 + 2c \cdot 1$ ✓

Ind. Step:

Assume (*) holds for problems of size $< n$

$$T(n) \leq 2 \cdot T(n/2) + cn^2$$

(I.H.: formula holds for $n/2$ — induction hypothesis)

$$\overset{\text{(I.H.)}}{\leq} 2\left(a \cdot \frac{n}{2} + 2c\left(\frac{n}{2}\right)^2\right) + cn^2$$

$$= a \cdot n + c \cdot n^2 + c \cdot n^2 = a \cdot n + 2cn^2$$

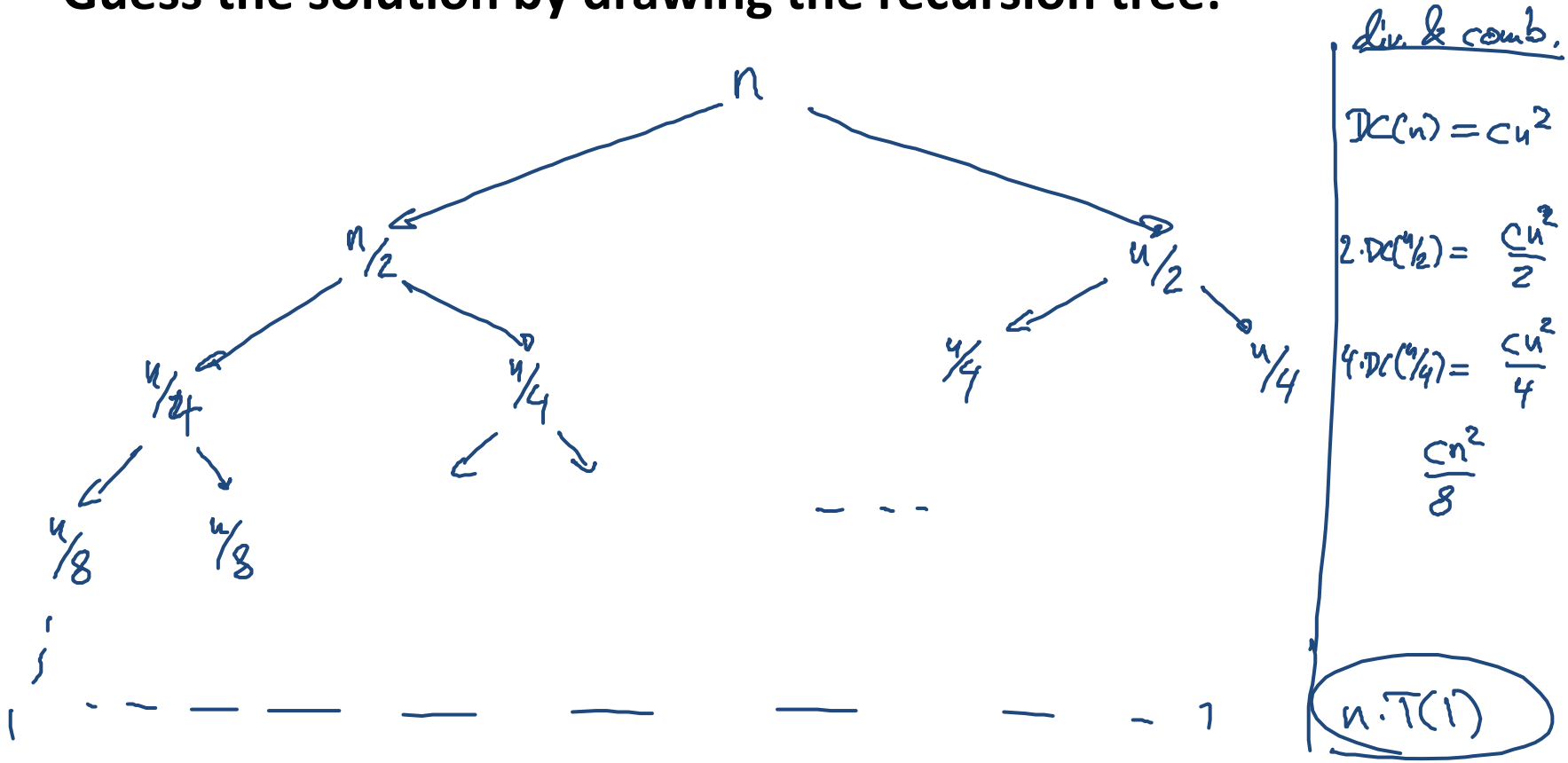
□

Analysis, Example

Recurrence relation:

$$T(n) \leq 2 \cdot T(n/2) + cn^2, \qquad T(1) \leq a$$

Guess the solution by drawing the recursion tree:



Comparing Orders

- Many web systems maintain user preferences / rankings on things like books, movies, restaurants, …

- Collaborative filtering:
  - Predict user taste by comparing rankings of different users.
  - If the system finds users with similar tastes, it can make recommendations (e.g., Amazon)

- Core issue: Compare two rankings
  - Intuitively, two rankings (of movies) are more similar, the more pairs are ordered in the same way
  - Label the first user's movies from 1 to n according to ranking
  - Order labels according to second user's ranking
  - How far is this from the ascending order (of the first user)?

Number of Inversions

Formal problem:

- **Given**: array $A = [a_1, a_2, a_3, \dots, a_n]$ of distinct elements

- **Objective**: Compute number of inversions I

$$I := \left|\{0 \leq i < j \leq n \mid a_i > a_j)\}\right|$$

- **Example**: $A = [\ 4\ ,\ 1\ ,\ 5\ ,\ 2\ ,\ 7\ ,\ 10\ ,\ 6\ ]$

5 inversions

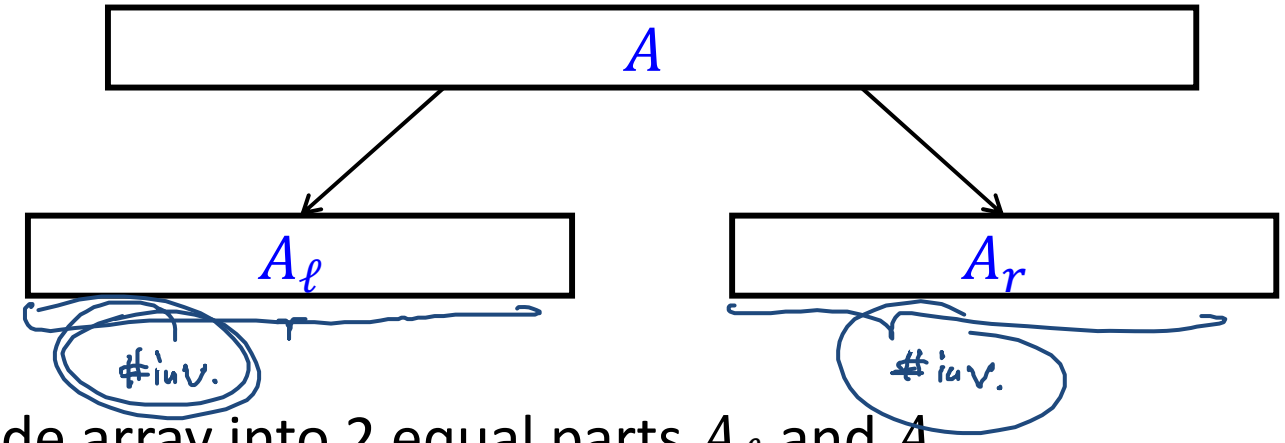
compare this to 1, 2, 4, 5, 6, 7, 10

pairs ordered in the wrong way: (4,1), (4,2), (5,2), (10,6), (7,6)

- **Naive solution**: go through all pairs

running time: $\Theta(n^2)$

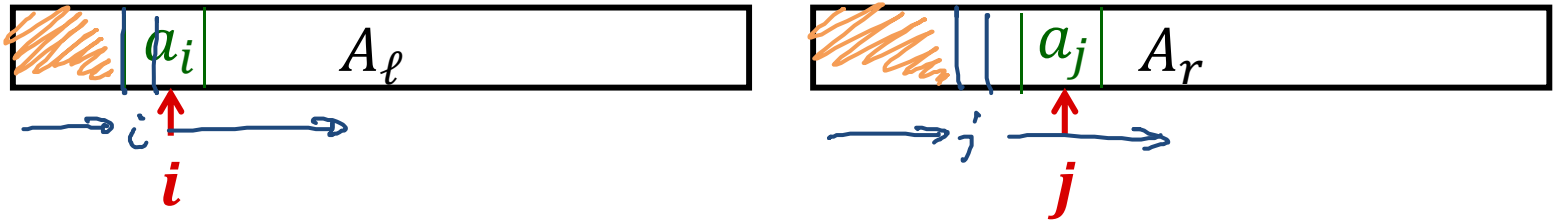
Divide and conquer



1. Divide array into 2 equal parts A_ℓ and A_r
2. Recursively compute #inversions in A_ℓ and A_r
3. Combine: add #pairs $a_i \in A_\ell$, $a_j \in A_r$ such that $a_i > a_j$

Combine Step

Assume A_ℓ and A_r are sorted



Idea:

- Maintain pointers i and j to go through the sorted parts
- While going through the sorted parts, we merge the two parts into one sorted part (like in MergeSort)

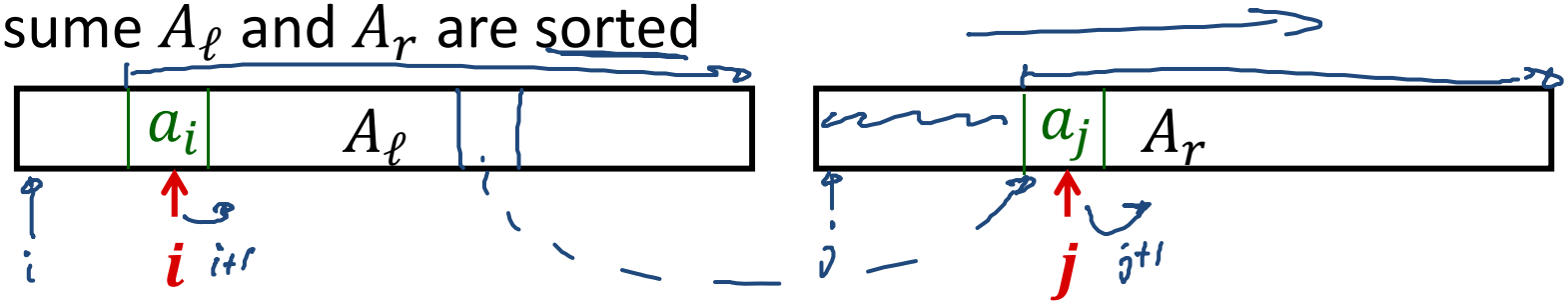
  and we count the number of inversions between the parts

Invariant:

- At each point in time, all inversions involving some element left of i (in A_ℓ) or left of j (in A_r) are counted
  - and all others still have to be counted...

Combine Step

Assume A_ℓ and A_r are sorted



- Pointers i and j, initially pointing to first elements of A_ℓ and A_r
- If $a_i \leqslant a_j$:
  - a_i is smallest among the remaining elements
  - No inversion of a_i and one of the remaining elements
  - Do not change count
- If $a_i > a_j$:
  - a_j is smallest among the remaining elements
  - a_j is smaller than all remaining elements in A_ℓ
  - Add number of remaining elements in A_ℓ to count
- Increment point, pointing to smaller element

Combine Step

- Need sub-sequences in sorted order
- Then, combine step is like merging in merge sort

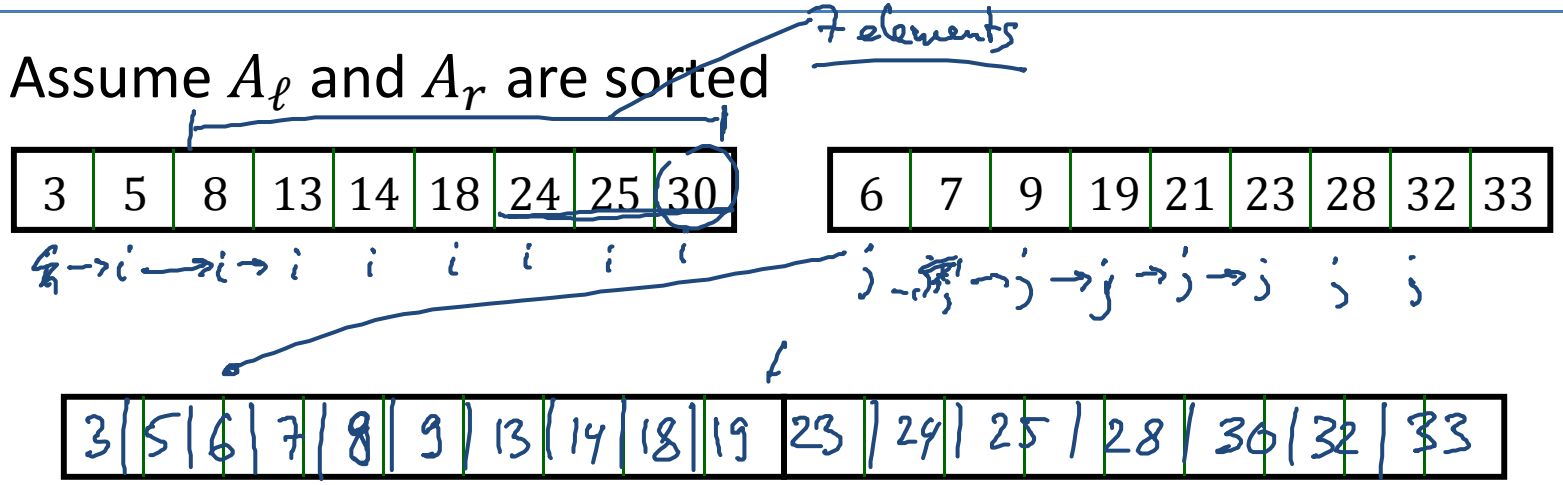
- **Idea**: Solve sorting and #inversions at the same time!
  1. Partition A into two equal parts A_ℓ and A_r
  2. Recursively compute #inversions and sort A_ℓ and A_r

     $2T(n/2)$
  3. Merge A_ℓ and A_r to sorted sequence, at the same time, compute number of inversions between elements a_i in A_ℓ and a_j in A_r

     all of this in linear time

Combine Step: Example

- Assume A_ℓ and A_r are sorted



count: $0+7+7+6+3+3+1=27$

$(+3? = 30)$

to account for "21"

Analysis, Guessing

Recurrence relation:

$$T(n) \leq 2 \cdot T(n/2) + c \cdot n, \qquad T(1) \leq c$$

Repeated substitution:

$$\begin{aligned} T(n) &\leq 2T(n/2) + cn \\ &\leq 4 \cdot T(n/4) + 2cn \\ &\leq 8T(n/8) + 3cn \\ &\vdots \\ &\leq 2^i T(n/2^i) + icn \\ &\leq n \cdot T(1) + \underbrace{c \cdot n \log_2 n}_{\leq c} \leq \underline{cn(1 + \log_2 n)} \end{aligned}$$

Analysis, Induction

guess: $T(n) \leq cn(1+\log_2 n)$

Recurrence relation:

$$T(n) \leq 2 \cdot T(n/2) + c \cdot n, \qquad T(1) \leq c$$

Verify by induction:

base: $n=1$ $\quad T(n) \leq c$ ✓

step: $T(n) \leq 2T(n/2) + c \cdot n$

$$\overset{(I+1)}{\leq} 2\left(c\frac{n}{2} \cdot \underbrace{(1+\log_2 \tfrac{n}{2})}_{\log_2 n}\right) + cn$$

$$= c \cdot n \cdot \log_2 n + cn = cn(1+\log_2 n)$$ ✓

□