Algorithms and Data Structures 2
Exam Notes
Week 1: Divide-and-Conquer

Mads Richardt

Objective

This document is a self-contained set of notes and worked solutions for the **Week 1: Divide-and-Conquer** problem set. It contains:

- General methodology for solving recurrence relations.
- Relevant mathematical tools (geometric series, logarithm rules).
- Notes and derivations from the lecture slides.
- Complete solutions to the weekly problem set, with references back to the notes.

1 General Methodology

1.1 Recursion Tree Method

To solve a recurrence of the form

$$T(n) \leq aT\Big(\frac{n}{b}\Big) + f(n),$$

we:

1. Draw the recursion tree. The root is $T(n)$.
2. Expand each node into a children of size n/b.
3. Compute the work per level: sum of all $f(\cdot)$ terms.
4. Compute the number of levels: the depth is $\log_b n$ (until the subproblem size becomes constant).
5. Multiply out to find total work: sum of work over all levels.

Number of leaves: Each level multiplies the number of nodes by a. After $\log_b n$ levels, there are

$$a^{\log_b n} = n^{\log_b a}$$

leaves.

1.2 Substitution Method

To solve a recurrence, we:

1. Guess a bound $T(n) \leq cn \log n$ or similar.
2. Use induction to prove the bound holds.
3. Verify the base case.

1.3 Notation

- $O(g(n))$: upper bound.
- $\Omega(g(n))$: lower bound.
- $\Theta(g(n))$: tight bound.

2 Relevant Mathematical Tools

2.1 Geometric Series

For $x \neq 1$:

$$\sum_{i=0}^{m} x^i = \frac{x^{m+1} - 1}{x - 1}.$$

Special case $x = \frac{1}{2}$:

$$\sum_{i=0}^{\log n} \frac{1}{2^i} \leq 2.$$

2.2 Logarithm Rules

$$\log_b n = \frac{\log n}{\log b}, \quad a^{\log_b n} = n^{\log_b a}.$$

3 Notes from Slides

- Divide-and-conquer structure: divide, conquer, combine.
- Mergesort recurrence:

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

  Solution: $T(n) \in \Theta(n \log n)$.
- Counting inversions: use divide-and-conquer with merge step that counts cross-inversions.

4 Solutions to Problem Set

1.1 Recurrence

$$T(n) \leq 2T(n/4) + cn.$$

Recursion Tree: At level i, there are 2^i nodes, each of size $n/4^i$. Work per node: $c \cdot (n/4^i)$. Work per level: $2^i \cdot c(n/4^i) = c \cdot n \cdot (1/2)^i$.

$$\text{Total work} = cn \sum_{i=0}^{\log_4 n} \left(\frac{1}{2}\right)^i \leq 2cn.$$

$$\Rightarrow T(n) \in \Theta(n).$$

Substitution: Guess $T(n) \leq kn$. Induction step:

$$T(n) \leq 2k(n/4) + cn = \frac{k}{2}n + cn.$$

For $k \geq 2c$, $T(n) \leq kn$.

$$\Rightarrow T(n) \in O(n).$$

1.2 Recurrence

$$T(n) \leq 2T(n/4) + c\sqrt{n}.$$

Recursion Tree: Work per level: $2^i \cdot c\sqrt{n/4^i} = c\sqrt{n} \cdot (2^i/2^i) = c\sqrt{n}$. Number of levels: $\log_4 n$.

$$\text{Total work} = c\sqrt{n} \cdot \log_4 n.$$

$$\Rightarrow T(n) \in \Theta(\sqrt{n} \log n).$$

2. Counting Inversions (KT 4.2)

Divide-and-conquer:

1. Split array into halves.
2. Recursively count inversions in each half.
3. Count cross-inversions during merge: whenever $a[i] > b[j]$, add $(\text{mid} - i + 1)$ inversions.

$$T(n) \leq 2T(n/2) + cn \quad \Rightarrow \quad T(n) \in \Theta(n \log n).$$

3. Divide-and-Conquer on Trees (KT 4.6)

- Use recursion: pick the middle node, recursively process subtrees.
- Time: $O(n)$.

4. Divide-and-Conquer on Grid Graphs (KT 4.7)

- Divide the grid into quadrants.
- Recurse on subgrids, combine at boundaries.
- Recurrence depends on structure, usually $O(n \log n)$.

5. CSES Exercises

Missing Number (1083): Sum formula vs. array sum: $O(n)$.
Distinct Numbers (1621): Sort array, count distinct: $O(n \log n)$.

6.1 Recurrence

$$T(n) \leq T(3n/4) + cn.$$

Recursion tree: work per level decreases linearly, depth $\log_{4/3} n$.

$$\Rightarrow T(n) \in \Theta(n).$$

6.2 Recurrence

$$T(n) \leq T(n/2) + T(n/3) + T(n/6) + cn.$$

By expansion, the tree has total branching factor summing to 1.

$$\Rightarrow T(n) \in \Theta(n \log n).$$

7. Divide-and-Conquer on Trees II

If tree is unbalanced: use separator lemma. Always possible to find a node splitting tree into subtrees of size $\leq n/2$.

$$\Rightarrow O(\log n) \text{ probes.}$$

Summary

- Recurrences solved explicitly (with references to recursion trees and substitution).
- Inversions solved with divide-and-conquer.
- Trees and grid graphs analyzed.
- CSES tasks solved.

Reading Material

KT Chapter 4 (Divide and Conquer), section 4.1 (A First Recurrence: The Mergesort Algorithm), 4.2 (Further Recurrence Relations), and 4.3 (Counting Inversions).

If you have another edition of the book than the international one from 2014 Divide and Conquer might be in a different chapter (e.g. 5) , but the subsections are the same. In that case you should of course also solve the corresponding exercises (the numbers within the chapters are the same, e.g. if Divide and Conquer is chapter 5 in our book, you should solve exercise 5.1 instead of 4.1).

Exercises

Exercises marked with [w] are easy warmup exercises. They are just there to check that you understand the very basics. Exercises marked with [*] are extra difficult exercises. The puzzle of the week is just for fun.

1 Reccurrences I Use both the *recursion tree method* and the *(partial) substitution method* to solve each of the following recurrences.

1.1 $T(n) \leq \begin{cases} 2T(n/4) + cn & \text{for } n > 4 \\ c & \text{otherwise} \end{cases}$

1.2 $T(n) \leq \begin{cases} 2T(n/4) + c\sqrt{n} & \text{for } n > 4 \\ c & \text{otherwise} \end{cases}$

2 Significant Inversions Solve KT exercise 4.2.

3 Divide-and-conquer on trees Solve KT exercise 4.6.

4 Divide-and-conquer on grid graphs Solve KT exercise 4.7.

5 CSES exercises To get familiar with the CSES online judge system solve the following two exercises on CSES (you first need to make a CSES profile).

- Missing Number: https://cses.fi/problemset/task/1083
- Distinct Numbers: https://cses.fi/problemset/task/1621

We will use CSES for our programming exercises during the course.

6 Reccurrences II Use both the *recursion tree method* and the *(partial) substitution method* to solve each of the following recurrences.

6.1 $T(n) \leq \begin{cases} T(\frac{3n}{4}) + cn & \text{for } n > 4 \\ c & \text{otherwise} \end{cases}$

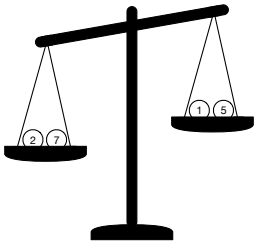
6.2 $T(n) \leq \begin{cases} T(n/2) + T(n/3) + T(n/6) + cn & \text{for } n > 6 \\ c & \text{otherwise} \end{cases}$

7 [$*$] **Divide-and-conquer on trees II** Consider the problem from exercise 2 (Divide-and-conquer on trees). What happens if the tree is not balanced? How many probes do we need in this case?

Hint: In a binary tree with n vertices there always exists a vertex such that if you remove it each of the remaining trees have size at most $n/2$. (This vertex is called a *separator*).

Finding such a separator does not count in the number of probes used in this exercise.

Puzzle of the week You are given twelve balls, eleven of which are identical and one of which is different, but it is not known whether it is heavier or lighter than the others. You have a traditional balance scale with two pans. To use such a scale, you can place a group of balls into each pan and the scale will determine which group is heavier.



The balance scale may be used three times to isolate the unique ball and determine wether it is heavier or lighter than the others. You can assume that the balls are numbered so that you can distinguish them from each other.

Divide-and-Conquer

Inge Li Gørtz

Thank you to Kevin Wayne for inspiration to slides

Mergesort

Divide-and-Conquer

- Divide -and-Conquer.
  - Break up problem into several parts.
  - Solve each part recursively.
  - Combine solutions to subproblems into overall solution.

- Today
  - Mergesort (recap)
  - Recurrence relations
  - Integer multiplication

Recurrence relations

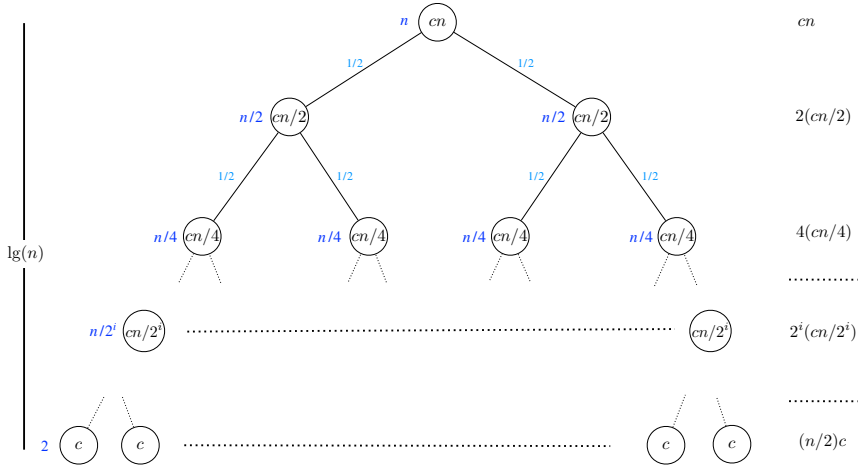
- T(n) = running time of mergesort on input of size n
- Mergesort recurrence:

$$T(n) \leq \begin{cases} 2T(n/2) + cn & \text{if } n > 2 \\ c & \text{otherwise} \end{cases}$$

- Solving the recurrence:
  - Recursion tree
  - Substitution

Mergesort recurrence: recursion tree

$$T(n) \le \begin{cases} 2T(n/2) + cn & \text{if } n > 2 \\ c & \text{otherwise} \end{cases}$$



Mergesort recurrence: substitution

$$T(n) \le \begin{cases} 2T(n/2) + cn & \text{if } n > 2 \\ c & \text{otherwise} \end{cases}$$

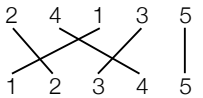
- Substitute $T(n)$ with $kn \lg n$ and use induction to prove $T(n) \le n \lg nk$.
- Base case ($n = 2$):
  - By definition $T(2) = c$.
  - Substitution: $k \cdot 2 \lg 2 = 2k \ge c = T(2)$ if $k \ge c/2$.
- Induction: Assume $T(m) \le km \lg m$ for $m < n$.

$$\begin{aligned} T(n) &\le 2T(n/2) + cn \\ &\le 2k(n/2)\lg(n/2) + cn \\ &= kn(\lg n - 1) + cn \\ &= kn \lg n - kn + cn \\ &\le kn \lg n \quad \text{if} \quad k \ge c. \end{aligned}$$

Counting Inversions

Counting Inversions

- Given sequence (permutation) $a_1, a_2, \ldots, a_n$ of the numbers from 1 to n.
- Inversion: a_i and a_j inverted if $i < j$ and $a_i > a_j$.



- Applications:
  - Comparing preferences (e.g. on a music site).
  - Voting theory
  - Collaborative filtering
  - Measuring the “sortedness” of an array.
  - Sensitivity of Google’s ranking function.

Counting Inversions

- Given sequence (permutation) $a_1, a_2, \ldots, a_n$ of the numbers from 1 to n.
- Inversion: a_i and a_j inverted if $i < j$ and $a_i > a_j$.

10	8	14	4	15	1	9	5	16	7	3	12	13	2	11	6

- Brute-force:
  - Compare each a_i with each a_j, where $i < j$.

Counting Inversions

- Given sequence (permutation) $a_1, a_2, \ldots, a_n$ of the numbers from 1 to n.
- Inversion: a_i and a_j inverted if $i < j$ and $a_i > a_j$.

10	8	14	4	15	1	9	5	16	7	3	12	13	2	11	6

- Brute-force:
  - Compare each a_i with each a_j, where $i < j$.

Counting Inversions

- Given sequence (permutation) $a_1, a_2, \ldots, a_n$ of the numbers from 1 to n.
- Inversion: a_i and a_j inverted if $i < j$ and $a_i > a_j$.

10	***8***	14	***4***	15	***1***	***9***	***5***	16	***7***	***3***	12	13	***2***	11	***6***

- Brute-force:
  - Compare each a_i with each a_j, where $i < j$.

Counting Inversions

- Given sequence (permutation) $a_1, a_2, \ldots, a_n$ of the numbers from 1 to n.
- Inversion: a_i and a_j inverted if $i < j$ and $a_i > a_j$.

10	8	14	4	15	1	9	5	16	7	3	12	13	2	11	6

- Brute-force:
  - Compare each a_i with each a_j, where $i < j$.

Counting Inversions

- Given sequence (permutation) $a_1, a_2, \ldots, a_n$ of the numbers from 1 to n.
- Inversion: a_i and a_j inverted if $i < j$ and $a_i > a_j$.

10	8	14	***4***	15	***1***	9	***5***	16	***7***	***3***	12	13	***2***	11	***6***

- Brute-force:
  - Compare each a_i with each a_j, where $i < j$.

Counting Inversions

- Given sequence (permutation) $a_1, a_2, \ldots, a_n$ of the numbers from 1 to n.
- Inversion: a_i and a_j inverted if $i < j$ and $a_i > a_j$.

10	8	14	4	15	1	9	5	16	7	3	12	13	2	11	6

- Brute-force:
  - Compare each a_i with each a_j, where $i < j$.

Counting Inversions

- Given sequence (permutation) $a_1, a_2, \ldots, a_n$ of the numbers from 1 to n.
- Inversion: a_i and a_j inverted if $i < j$ and $a_i > a_j$.

10	8	14	***4***	15	***1***	***9***	***5***	16	***7***	***3***	12	13	***2***	11	***6***

- Brute-force:
  - Compare each a_i with each a_j, where $i < j$.
  - Time: $O(n^2)$

Counting Inversions

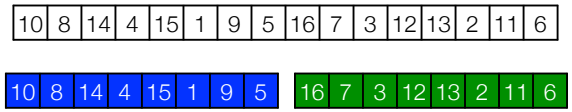
- Given sequence (permutation) $a_1, a_2, \ldots, a_n$ of the numbers from 1 to n.
- Inversion: a_i and a_j inverted if $i < j$ and $a_i > a_j$.

10	8	14	4	15	1	9	5	16	7	3	12	13	2	11	6

- Divide-and-Conquer:
  - Divide: Split list in two.

Counting Inversions

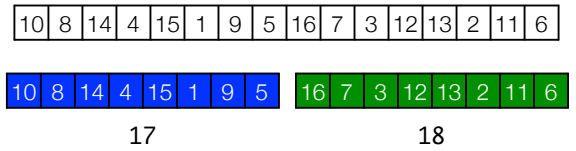
- Given sequence (permutation) $a_1, a_2, \ldots, a_n$ of the numbers from 1 to n.
- Inversion: a_i and a_j inverted if $i < j$ and $a_i > a_j$.



- Divide-and-Conquer:
  - Divide: Split list in two.
  - Conquer: recursively count inversions in each half.

Counting Inversions

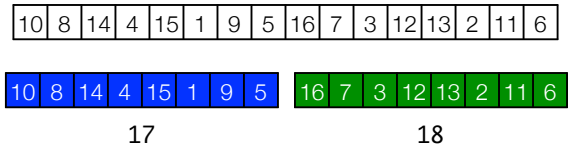
- Given sequence (permutation) $a_1, a_2, \ldots, a_n$ of the numbers from 1 to n.
- Inversion: a_i and a_j inverted if $i < j$ and $a_i > a_j$.



- Divide-and-Conquer:
  - Divide: Split list in two.
  - Conquer: recursively count inversions in each half.

Counting Inversions

- Given sequence (permutation) $a_1, a_2, \ldots, a_n$ of the numbers from 1 to n.
- Inversion: a_i and a_j inverted if $i < j$ and $a_i > a_j$.

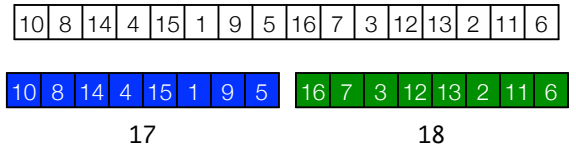


- Divide-and-Conquer:
  - Divide: Split list in two.
  - Conquer: recursively count inversions in each half.
  - Combine:
    - count inversions where a_i and a_j are in different halves
    - return sum.

```
17 + 18 + 30 = 65
```

Counting Inversions

- Given sequence (permutation) $a_1, a_2, \ldots, a_n$ of the numbers from 1 to n.
- Inversion: a_i and a_j inverted if $i < j$ and $a_i > a_j$.



- Divide-and-Conquer:
  - Divide: Split list in two.
  - Conquer: recursively count inversions in each half.
  - Combine:
    - count inversions where a_i and a_j are in different halves
    - return sum.

Divide: O(1)

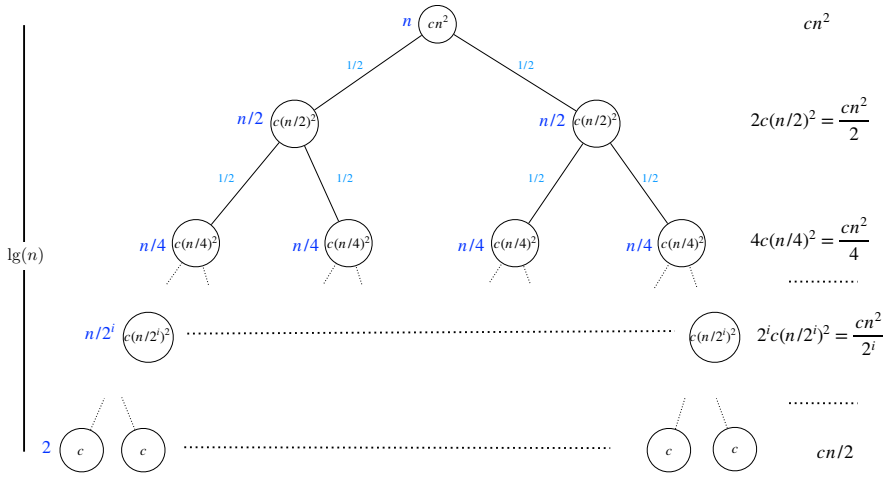
Conquer: 2T(n/2)

Combine: ???

```
17 + 18 + 30 = 65
```

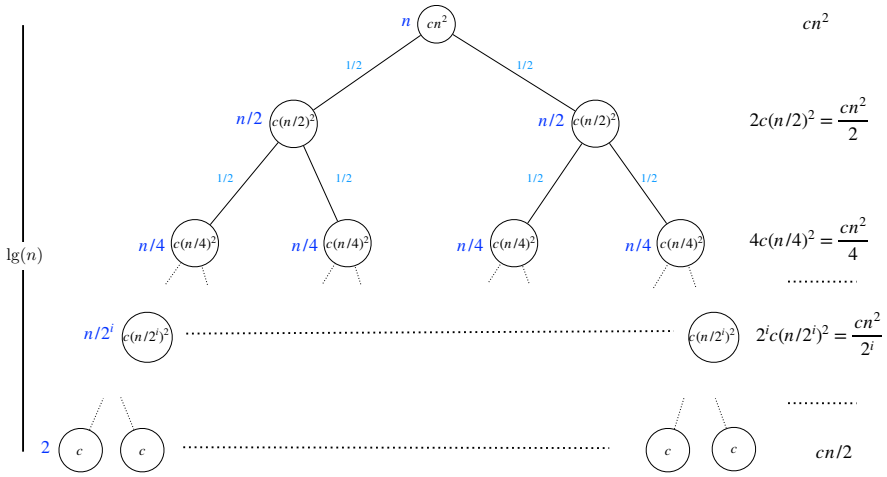
Another recurrence

$$T(n) \le \begin{cases} 2T(n/2) + cn^2 & \texttt{if } n > 2 \\ c & \texttt{otherwise} \end{cases}$$



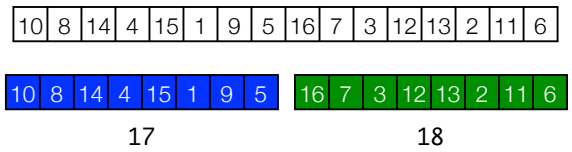
More recurrences

$$T(n) \le \begin{cases} 2T(n/2) + cn^2 & \texttt{if } n > 2 \\ c & \texttt{otherwise} \end{cases} \qquad T(n) \le \sum_{i=0}^{\lg n} \frac{cn^2}{2^i} \le cn^2 \sum_{i=0}^{\lg n} \frac{1}{2^i} \le 2cn^2$$



Counting Inversions

- Given sequence (permutation) $a_1, a_2, \ldots, a_n$ of the numbers from 1 to n.
- Inversion: a_i and a_j inverted if $i < j$ and $a_i > a_j$.



- Divide-and-Conquer:
  - Divide: Split list in two. *Divide: O(1)*
  - Conquer: recursively count inversions in each half. *Conquer: 2T(n/2)*
  - Combine: *Combine: ???*
    - count inversions where a_i and a_j are in different halves
    - return sum.

`17 + 18 + 30 = 65`

Counting Inversions: Combine

- Combine: count inversions where a_i and a_j are in different halves.

Counting Inversions: Combine

- Combine: count inversions where a_i and a_j are in different halves.
  - Assume each half sorted.

Counting Inversions: Combine

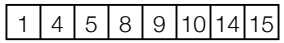
- Combine: count inversions where a_i and a_j are in different halves.
  - Assume each half sorted.

Counting Inversions: Combine

- Combine: count inversions where a_i and a_j are in different halves.
  - Assume each half sorted.

Counting Inversions: Combine

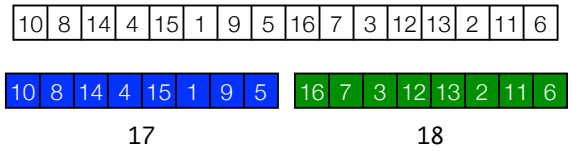
- Combine: count inversions where a_i and a_j are in different halves.
  - Assume each half sorted.
  - Merge sorted halves into sorted whole while counting inversions.



Inversions: 4 + 3 + 2 + 0 = 9

Counting Inversions

- Given sequence (permutation) $a_1, a_2, \ldots, a_n$ of the numbers from 1 to n.
- Inversion: a_i and a_j inverted if $i < j$ and $a_i > a_j$.



- Divide-and-Conquer:
  - Divide: Split list in two. *Divide: O(1)*
  - Conquer: recursively count inversions in each half. *Conquer: 2T(n/2)*
  - Combine: *Combine: O(n)*
    - Merge-and-Count.

More Recurrence Relations

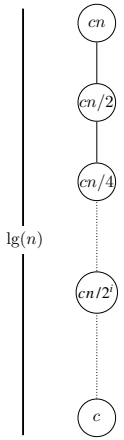
Counting Inversions: Implementation

```
Sort-and-Count(L):

  if list L has one element:
     return (0, L)

  divide the list L into two halves A and B
  (iA, A) = Sort-and-Count(A)
  (iB, B) = Sort-and-Count(B)
  (iL, L) = Merge-and-Count(A, B)

  i = iA + iB + iL

  return (i, L)
```

- Pre-condition (Merge-and-Count): A and B are sorted.
- Post-condition (Sort-and-Count, Merge-and-Count): L is sorted.

More recurrence relations: 1 subproblem

$$T(n) \leq \begin{cases} T(n/2) + cn & \text{if } n > 2 \\ c & \text{otherwise} \end{cases}$$



- Summing over all levels:

$$T(n) \leq \sum_{i=0}^{\lg n - 1} \frac{cn}{2^i} = cn \sum_{i=0}^{\lg n - 1} \frac{1}{2^i} \leq 2cn = O(n)$$

- Substitution: Guess $T(n) \leq kn$
  - Base case:

$$k \cdot 2 \geq c = T(2) \quad \text{if} \quad k \geq c/2.$$

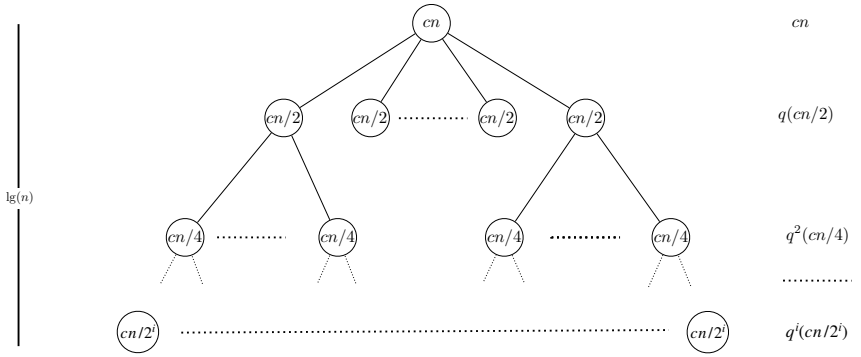
  - Assume $T(m) \leq km$ for $m < n$.

$$T(n) \leq T(n/2) + cn \leq k(n/2) + cn = (k/2)n + cn$$
$$\leq kn \quad \text{if} \quad c \leq k/2.$$

More than 2 subproblems

- q subproblems of size n/2.

$$T(n) \le \begin{cases} qT(n/2) + cn & \text{if } n > 2 \\ c & \text{otherwise} \end{cases}$$



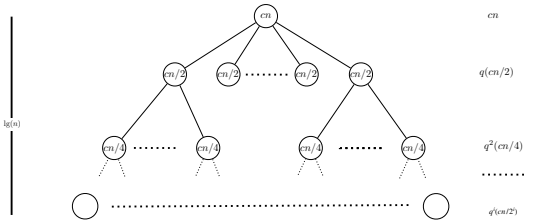
More than 2 subproblems

- q subproblems of size n/2.

$$T(n) \le \begin{cases} qT(n/2) + cn & \text{if } n > 2 \\ c & \text{otherwise} \end{cases}$$

- Summing over all levels:

$$T(n) \le \sum_{j=0}^{\lg n-1} \left(\frac{q}{2}\right)^j cn = cn \sum_{j=0}^{\lg n-1} \left(\frac{q}{2}\right)^j$$



Geometric series.

$$\text{for } x \neq 1: \sum_{i=0}^{m} x^i = \frac{x^{m+1}-1}{x-1}$$

$$\text{for } x < 1: \sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$$

More than 2 subproblems

Proof of $cn \sum_{j=0}^{\lg n-1} \left(\frac{q}{2}\right)^j = O(n^{\lg q})$

Use geometric series: $cn \sum_{j=0}^{\lg n-1} \left(\frac{q}{2}\right)^j = cn \frac{\left(\frac{q}{2}\right)^{\lg n} - 1}{\frac{q}{2} - 1}$

Geometric series.

$$\text{for } x \neq 1: \sum_{i=0}^{m} x^i = \frac{x^{m+1}-1}{x-1}$$

$$\text{for } x < 1: \sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$$

Reduce $\left(\frac{q}{2}\right)^{\lg n} = \frac{q^{\lg n}}{2^{\lg n}} = \frac{q^{\lg n}}{n}$

Now:

$$cn \frac{\left(\frac{q}{2}\right)^{\lg n} - 1}{\frac{q}{2} - 1} = cn \frac{\frac{q^{\lg n}}{n} - 1}{\frac{q-2}{2}} = \frac{2c}{q-2} n \left(\frac{q^{\lg n}}{n} - 1\right) = \frac{2c}{q-2}\left(q^{\lg n} - n\right) = O(q^{\lg n})$$

constant

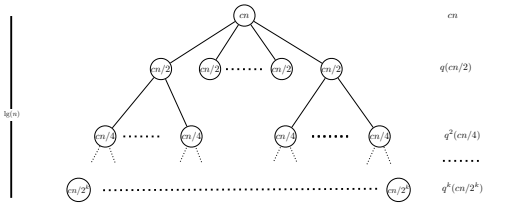
More than 2 subproblems

- q subproblems of size n/2.

$$T(n) \le \begin{cases} qT(n/2) + cn & \text{if } n > 2 \\ c & \text{otherwise} \end{cases}$$

- Summing over all levels:

$$T(n) \le \sum_{j=0}^{\lg n-1} \left(\frac{q}{2}\right)^j cn = cn \sum_{j=0}^{\lg n-1} \left(\frac{q}{2}\right)^j = O(n^{\lg q})$$



Geometric series.

$$\text{for } x \neq 1: \sum_{i=0}^{m} x^i = \frac{x^{m+1}-1}{x-1}$$

$$\text{for } x < 1: \sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$$

Subproblems of different sizes

$$T(n) = \begin{cases} T(3n/4) + T(n/2) + f(n) & \text{if } n > 4 \\ c & \text{otherwise} \end{cases}$$