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

Mads Richardt

1 General Methodology and Theory

Divide-and-Conquer Strategy

1. Divide problem into smaller subproblems (often of equal size).
2. Conquer each subproblem recursively.
3. Combine subproblem solutions into a full solution.

Recurrence Relations

General form for divide-and-conquer running times:

$$T(n) = q \cdot T\left(\frac{n}{b}\right) + f(n),$$

where

- q: number of subproblems,
- b: factor by which input size is reduced,
- $f(n)$: cost to divide and combine.

Solving Recurrences

Recursion Tree Method.

1. Expand recurrence level by level.
2. Compute cost per level.
3. Sum over all levels until base case.

Substitution Method.

1. Guess solution form $T(n) \leq k \cdot g(n)$.
2. Prove by induction:
    - Base case holds.
    - Inductive step: Plug hypothesis into recurrence.

Useful Mathematical Tools

- **Geometric series:** for $x \neq 1$,
$$\sum_{i=0}^{m} x^i = \frac{x^{m+1} - 1}{x - 1}.$$
For $|x| < 1$,
$$\sum_{i=0}^{\infty} x^i = \frac{1}{1 - x}.$$
- **Logarithm rules:**
$$\log_a b = \frac{\ln b}{\ln a}, \quad \log(ab) = \log a + \log b, \quad \log \frac{a}{b} = \log a - \log b.$$

2 Notes from Slides and Textbook

Mergesort Recurrence

$$T(n) \leq \begin{cases} 2T(n/2) + cn & n > 2, \\ c & n \leq 2. \end{cases}$$

Recursion tree analysis: Each level costs cn. There are $\log_2 n$ levels. Total:

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

Substitution: Guess $T(n) \leq kn \log n$. Show inductively:

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

Counting Inversions

- Inversion: pair (i, j) with $i < j$ and $a_i > a_j$.
- Divide-and-conquer algorithm: Sort-and-Count using merge.
- Running time: $O(n \log n)$.

3 Solutions to Problem Set

Exercise 1: Recurrences I

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

$$T(n) = 2T\Big(\frac{n}{4}\Big) + cn.$$

Recursion tree: At level i, 2^i subproblems of size $n/4^i$. Cost per subproblem $\approx c \cdot n/4^i$. Total per level:

$$2^i \cdot \frac{cn}{4^i} = \frac{cn}{2^i}.$$

Summing over $\log_4 n$ levels:

$$T(n) \leq cn \sum_{i=0}^{\log_4 n} \frac{1}{2^i} \leq 2cn = O(n).$$

(b) $T(n) \leq 2T(n/4)+c\sqrt{n}$**.** Level i has 2^i subproblems of size $n/4^i$. Cost per subproblem: $c\sqrt{n/4^i} = \frac{c}{2^i}\sqrt{n}$. Total per level:

$$2^i \cdot \frac{c}{2^i}\sqrt{n} = c\sqrt{n}.$$

Depth $\log_4 n$. Total:

$$T(n) = O(\sqrt{n}\log n).$$

Exercise 2: Significant Inversions (KT 4.2)

Use modification of Sort-and-Count:

```
Algorithm CountSignificantInversions(A):
  if length(A) = 1: return (0, A)
  split A into L and R
  (iL, L) := CountSignificantInversions(L)
  (iR, R) := CountSignificantInversions(R)
  (iM, M) := MergeAndCountSignificant(L, R)
  return (iL + iR + iM, M)
```

During merge, when comparing $a_i \in L$ with $a_j \in R$, if $a_i > 2a_j$, then all later elements in L (since sorted) also form significant inversions with a_j. Count efficiently in $O(n)$ per merge. Total complexity: $O(n \log n)$.

Exercise 3: Divide-and-Conquer on Trees (KT 4.6)

Problem: Find a local minimum in a complete binary tree with $n = 2^d - 1$ nodes using $O(\log n)$ probes.

Algorithm:

1. Probe the root.
2. Compare root with its children.
3. Recursively continue into the smaller child.

Because tree height is $\log n$, this uses $O(\log n)$ probes. Correctness: At each step, moving into the smaller neighbor guarantees a local minimum exists along that path.

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

Problem: Find a local minimum in an $n \times n$ grid with $O(n)$ probes.

Algorithm:

1. Check middle column for minimum entry x.
2. Compare x with its horizontal neighbors.
3. If x is smaller than both, x is a local minimum.
4. Otherwise recurse into the half-grid containing the smaller neighbor.

Each step reduces size by factor 2, cost $O(n)$ per level, $\log n$ levels. Total: $O(n)$ probes.

Exercise 5: CSES Programming

Missing Number. Sort or use XOR sum trick. XOR all numbers $1, \ldots, n$. XOR with input list. Result is missing number. Complexity: $O(n)$.

Distinct Numbers. Insert all into a set. Output set size. Complexity: $O(n \log n)$.

Exercise 6: Recurrences II

(a) $T(n) \leq T(3n/4)+cn$**.** *Tree method:* Level i: subproblem size $(3/4)^i n$. Cost per level $\approx c\cdot(3/4)^i n$. Sum over $\log_{4/3} n$ levels:

$$T(n) \leq cn \sum_{i=0}^{\log_{4/3} n} \left(\frac{3}{4}\right)^i \leq 4cn = O(n).$$

(b) $T(n) \leq T(n/2) + T(n/3) + T(n/6) + cn$**.** All subproblem sizes add up to n, so each level cost $\approx cn$. Depth $O(\log n)$. Therefore $T(n) = O(n \log n)$.

4 Summary

- Divide-and-conquer recurrence template: $T(n) = qT(n/b) + f(n)$.
- Geometric sums: essential for solving recurrences.
- Mergesort: $O(n \log n)$.
- Counting inversions: Sort-and-Count in $O(n \log n)$.
- Significant inversions: modify merge, still $O(n \log n)$.
- Local minimum in binary tree: $O(\log n)$ probes.
- Local minimum in grid: $O(n)$ probes.
- Recurrence results:

$$\begin{aligned}
T(n) &= 2T(n/4) + cn \quad &\Rightarrow O(n),\\
T(n) &= 2T(n/4) + c\sqrt{n} \quad &\Rightarrow O(\sqrt{n}\log n),\\
T(n) &= T(3n/4) + cn \quad &\Rightarrow O(n),\\
T(n) &= T(n/2) + T(n/3) + T(n/6) + cn \quad &\Rightarrow O(n \log n).
\end{aligned}$$

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

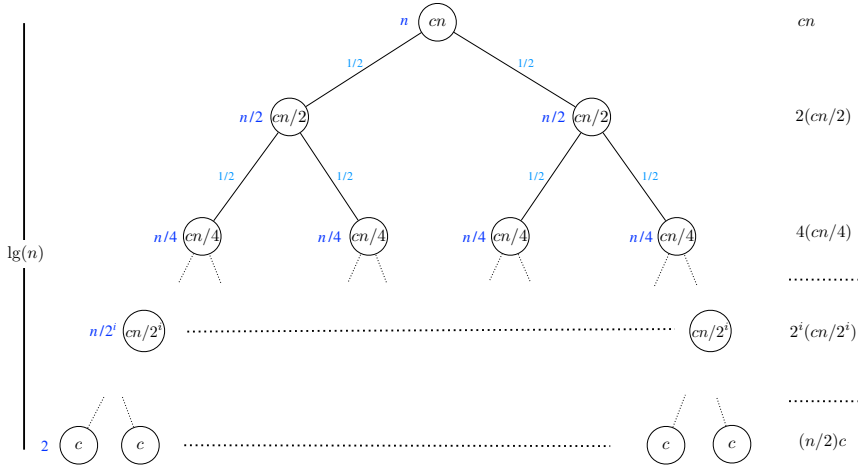
- T(n) = running time of mergesort on input of size n
- Mergesort recurrence:

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

- 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

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

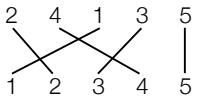
- Substitute $T(n)$ with $kn \lg n$ and use induction to prove $T(n) \le n \lg nk$.
- Base case ($n = 2$):
  - By definition $T(2) = c$.
  - Substitution: $k \cdot 2 \lg 2 = 2k \ge c = T(2)$ if $k \ge c/2$.
- Induction: Assume $T(m) \le km \lg m$ for $m < n$.

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

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

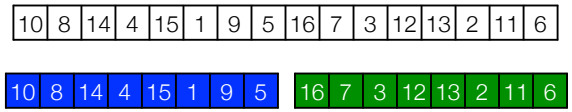
- Given sequence (permutation) $a_1, a_2, \ldots, a_n$ of the numbers from 1 to n.
- Inversion: a_i and a_j inverted if $i < j$ and $a_i > a_j$.

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

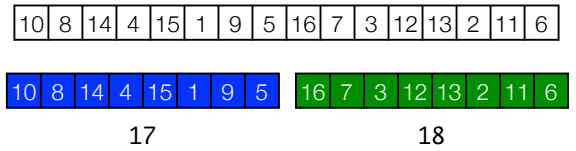
- Given sequence (permutation) $a_1, a_2, \ldots, a_n$ of the numbers from 1 to n.
- Inversion: a_i and a_j inverted if $i < j$ and $a_i > a_j$.



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

Counting Inversions

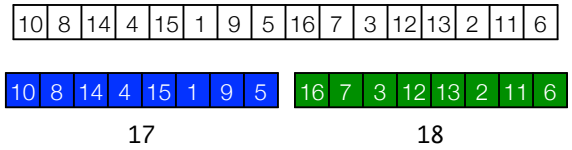
- Given sequence (permutation) $a_1, a_2, \ldots, a_n$ of the numbers from 1 to n.
- Inversion: a_i and a_j inverted if $i < j$ and $a_i > a_j$.



- 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 $a_i > a_j$.

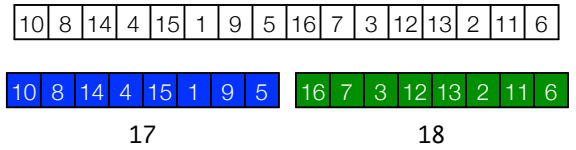


- Divide-and-Conquer:
  - Divide: Split list in two.
  - Conquer: recursively count inversions in each half.
  - Combine:
    - count inversions where a_i and a_j are in different halves
    - return sum.

```
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)

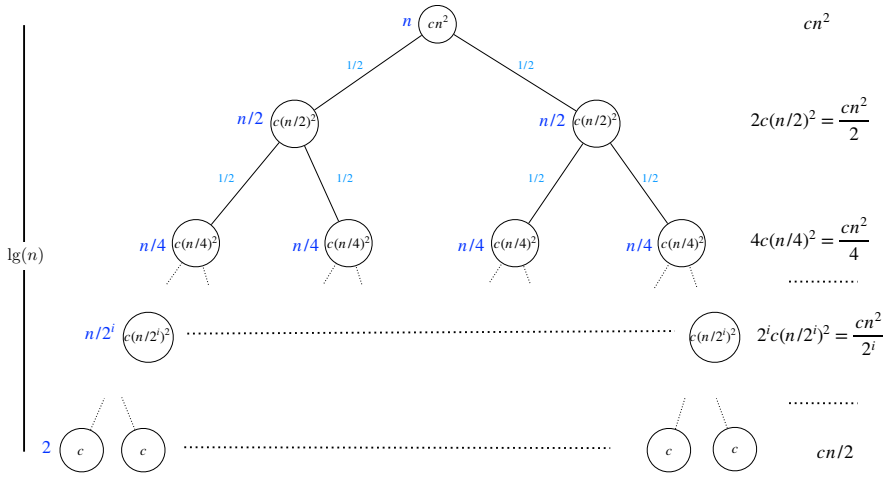
Conquer: 2T(n/2)

Combine: ???

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

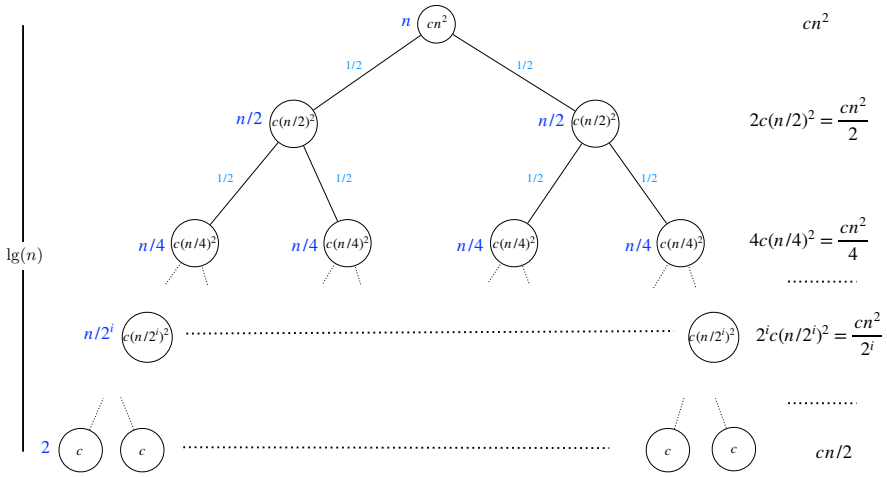
Another recurrence

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



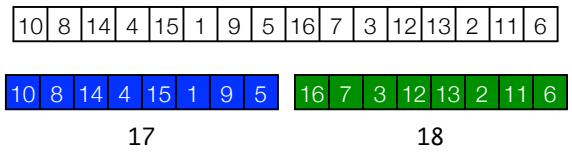
More recurrences

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



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

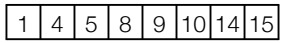
- Combine: count inversions where a_i and a_j are in different halves.
  - Assume each half sorted.

Counting Inversions: Combine

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

Counting Inversions: Combine

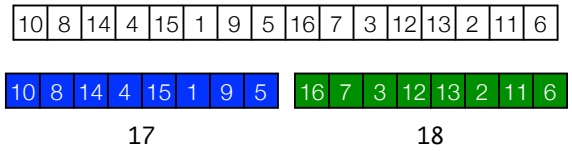
- Combine: count inversions where a_i and a_j are in different halves.
  - Assume each half sorted.
  - Merge sorted halves into sorted whole while counting inversions.



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

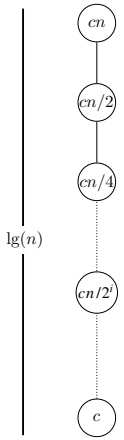
Counting Inversions: Implementation

```
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:

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

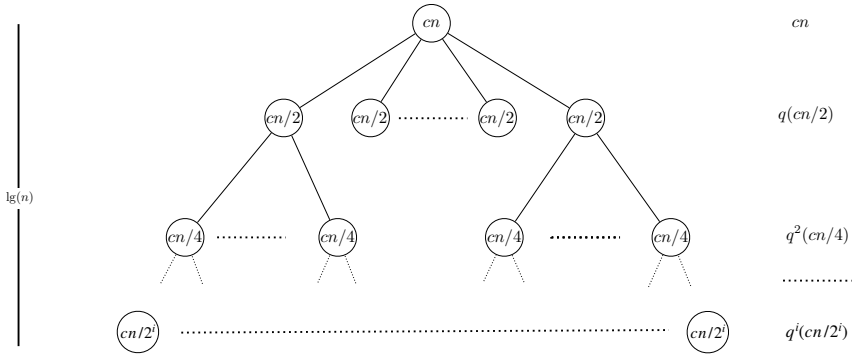
  - Assume $T(m) \leq km$ for $m < n$.

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

More than 2 subproblems

- q subproblems of size n/2.

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



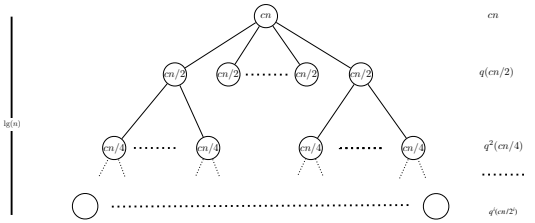
More than 2 subproblems

- 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

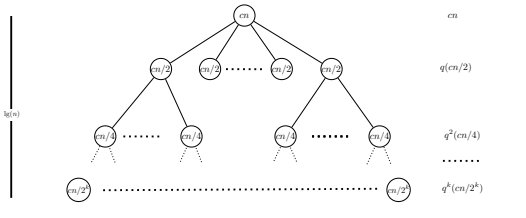
More than 2 subproblems

- 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:

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



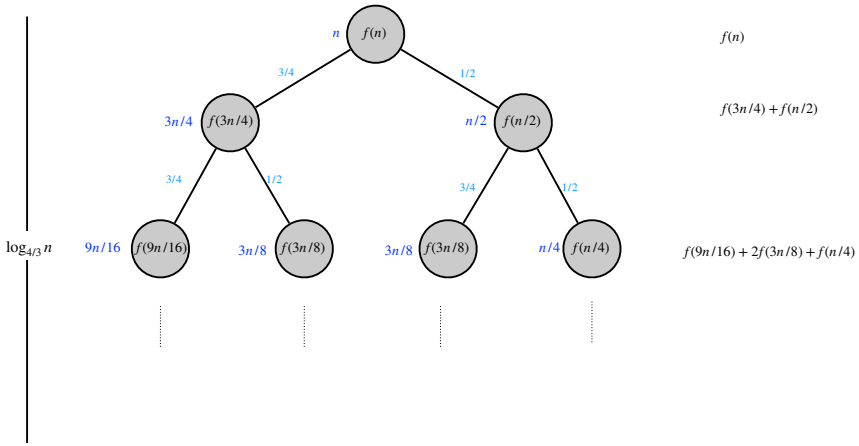
Geometric series.

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

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

Subproblems of different sizes

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



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

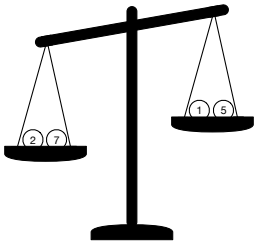
6.2 $T(n) \leq \begin{cases} T(n/2) + T(n/3) + T(n/6) + cn & \text{for } n > 6 \\ c & \text{otherwise} \end{cases}$

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.