Week 3: Divide-and-Conquer, Recurrence

Agenda:

- Divide & Conquer: MergeSort (CLRS p30-37)
- Solving Recurrence Relations (iterated substitution, recurrence tree, guess and test) (CLRS p83-92)
- Solving Recurrences (Cont'd, maybe covered in the following Monday's lecture) Master Theorem (CLRS p.93-97)

Divide and Conquer and recursive programs

- A useful design technique for algorithms is *divide-and-conquer*
- These algorithms are often recursive and consist of the following three steps:
  - Divide:
    - If the input size is very small (base case) then solve the problem using a simple method.
    - else, divide the input into two or more disjoint (smaller) pieces & *recursively* solve the subproblems
  - Conquer: Leverage on the solutions for the subproblems to get a solution for the original problem.
- To analyze (the running time of) a recursive program we express their running time as a *recurrence*
- We then solve the recurrence to find a closed form for the running time of the algorithm.

Merge-Sort

1 17 23 3 | 9 19 28

1, 9, 17, 23, 28, 31

```
Merge(A; lo, mid, hi)
      **pre-condition:   lo ≤ mid ≤ hi
      **pre-condition:   A[lo, mid] and A[mid + 1, hi] sorted
      **post-condition:  A[lo, hi] sorted
```

Merge runs in $O(n)$ time; the details are not particularly relevant here.

```
Merge-Sort(A; lo, hi)
   if lo < hi then
      mid = ⌊(lo + hi)/2⌋
      Merge-Sort(A; lo, mid)
      Merge-Sort(A; mid + 1, hi)
      Merge(A; lo, mid, hi)
```

Merge sort, the big idea — divide-and-conquer:

- Divide the whole list into 2 sublists of equal size;
- Recursively merge sort the 2 sublists;
- Combine the 2 sorted sublists into a sorted list.
- An example:

	1	2	3	4	5	6	7	8	9	10	11	12	13
A	[31	23	01	17	19	28	09	03	13	15	22	08	29]

Merge sort — Example:

[31 23 01 17 19 28 09 03 13 15 22 08 29]

[31 23 01 17 19 28 09] [03 13 15 22 08 29]

[31 23 01 17] [19 28 09] [03 13 15] [22 08 29]

[31 23] [01 17] [19 28] [09] [03 13] [15] [22 08] [29]

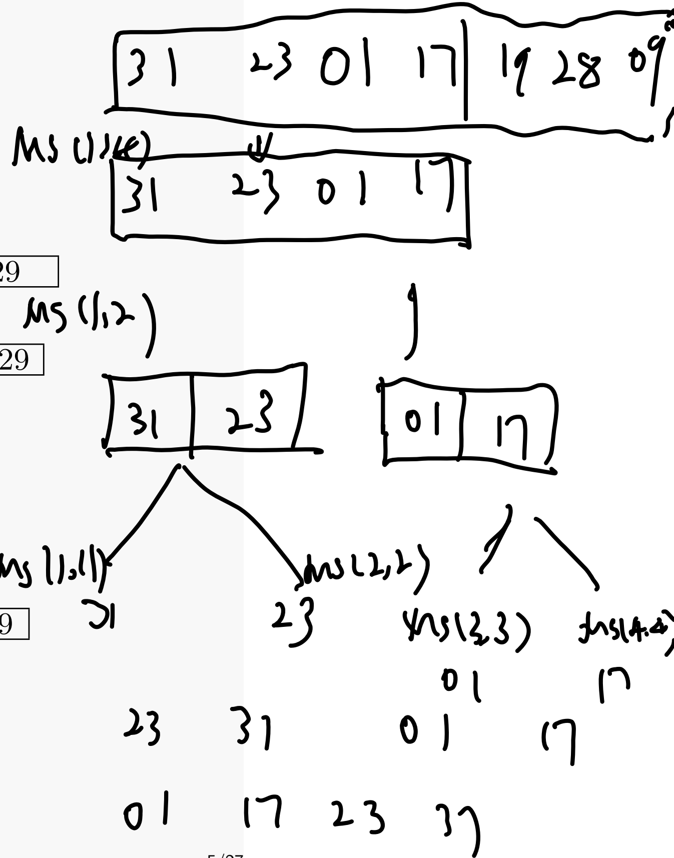
[31] [23] [01] [17] [19] [28] [03] [13] [22] [08]

[23 31] [01 17] [19 28] [03 13] [08 22]

[01 17 23 31] [09 19 28] [03 13 15] [08 22 29]

[01 09 17 19 23 28 31] [03 08 13 15 22 29]

[01 03 08 09 13 15 17 19 22 23 28 29 31]



different α.

$$f(n) = \begin{cases} 1 & n=1 \\ n+f(n-1) & n\geq 2 \end{cases}$$

n	1	2	3	4	5 ...
$f(n)$	1	3	6	10	15 ......

Recurrence relations — merge sort analysis

- MergeSort:
  - Divide the whole list into 2 sublists of equal size; recursively sort each sublist;
  - Merge the 2 sorted sublists into a sorted list.
- Let $T(n)$ denote #KC (# of key comparisons) for a list of size n
- Assumptions:
  - n (number of keys in the whole list) is a power of 2;
    This makes the analysis easier (since each time we are dividing by 2)
- Deriving recurrence relation:
  - Merge sort on 2 sublists $2 \times T(\frac{n}{2})$
  - Assembling needs $n-1$ KC (in WC, the worst case) (it actually takes n KC; but the code can be easily revised so that $n-1$ KC would be sufficient)
  - $$T(n) = \begin{cases} 0 & , \quad \text{if } n = 1 \\ (n-1) + 2 \cdot T(\frac{n}{2}) & , \quad \text{otherwise} \end{cases}$$
- How to solve this?

MS WC KC

$$T(n) = \begin{cases} \phi & n=1 \\ T(n/2)+T(n/2)+n-1 & n\geq 2 \end{cases}$$

$$= \begin{cases} \phi & n=1 \\ n-1+2T(n/2) & n\geq 2 \end{cases}$$

Recurrence relations

- How to analyze Merge-Sort or any other recursive program?
- Write the running time as a recursive function of input size
- *Recurrence relation*: A relation defined recursively — in terms of itself, *e.g.*,

$$f(n) = \begin{cases} 1, & \text{if } n = 1 \\ n + f(n-1), & \text{if } n \geq 2 \end{cases}$$

- Must have *base case* and *general case*.
- They arise in the analysis of Divide-and-Conquer algorithms
- How are recurrences solved?
  1. iterated substitution/replacement method
  2. recurrence tree method
  3. Guess and Test method
  4. master theorem method

$f(n) = n + f(n-1)$

$= n + (n-1) + f(n-1-1)$

$= n + n-1 + n-2 + f(n-3)$

$= n + n-1 + n-2 + n-3 + f(n-4)$

$4+3+2+1$

$= n + n-1 + \cdots + 2 + 1$

$= \sum_{j=1}^{n} j$

1- Iterated Substitution

- Consider a simple example $f(n) = \begin{cases} 1, & \text{if } n = 1 \\ n + f(n-1), & \text{if } n \geq 2 \end{cases}$
- Particular cases:

n	1	2	3	4	5	6	7
$f(n)$	1	$2+1$	$3+3$	$4+6$	$5+10$	$6+15$	$7+21$
	1	3	6	10	15	21	28

- General case:

$$\begin{aligned} f(n) &= n + f(n-1) \\ &= n + (n-1) + f(n-2) \\ &= n + (n-1) + (n-2) + f(n-3) \\ &= n + (n-1) + (n-2) + (n-3) + f(n-4) \\ &= \ldots \\ &= n + (n-1) + (n-2) + (n-3) + \ldots + 2 + f(1) \\ &= \sum_{i=1}^{n} i \end{aligned}$$

Therefore, we *guess* that $f(n) = \sum_{i=1}^{n} i$

This is NOT a proof, yet. We prove our guess by induction.

claim $f(n) = \sum_{j=1}^{n} j$

BC: $f(1) = \sum_{j=1}^{1} j$

$f(1) = 1$

inductive:

$k \geq 1$, assume claim holds

when $n = k$, $f(n) = \sum_{j=1}^{k} j$

what to show $\sum_{j=1}^{k+1} j = f(k+1)$

$k+1+\sum_{j=1}^{k} j$

$=\sum_{j=1}^{k+1} j$

Iterated substitution: example (cont'd)

- Prove that $f(n) = \sum_{i=1}^{n} i$ by induction.
  - Base case: $n = 1$. $f(1) = 1$ by definition, and $\sum_{i=1}^{1} i = 1$.
  - Inductive step:

    Assume $f(k) = \sum_{i=1}^{k} i$, for some natural $k \geq 1$. Want to show

    $f(k+1) = \sum_{i=1}^{k+1} i$ by using the recurrence relation (only).

    $f(k+1) = (k+1) + f(k) = (k+1) + \sum_{i=1}^{k} i = \sum_{i=1}^{k+1} i$. ■
- So,

$$f(n) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}, n \geq 1.$$

  Proof by induction (exercise).
- $\frac{n(n+1)}{2}$ is the *closed form* for the recurrence.
- You NEED to get the closed forms, as simple as possible!

$T(n)=T(2^k)$

$=(2^k-1)+2T(2^{k-1})$

$=2^k-1+2(2^{k-1}+2T(2^{k-1}))$

$=2^k-1+2^k-2+4T(2^{k-2})$

$=2^k-1+2^k-2+4(2^{k-2}+2T(2^{k-3}))$

$=2^k-1+2^k-2+2^k-4+8T(2^{k-3})$

$=2^k-2^0+2^k-2^1+2^k-2^2+2^k-2^3+2^4(T(2^{k-4}))$

Recurrence relations — merge sort analysis

- Merge sort recall:
  - Divide the whole list into 2 sublists of equal size;
  - Recursively merge sort the 2 sublists;
  - Combine the 2 sorted sublists into a sorted list.
- Assumptions:
  - n is a power of 2, which makes the analysis easier (since each time we are dividing by 2)
  - Let $T(n)$ denote #KC for a list of size n
- Deriving recurrence relation:
  - Merge sort on 2 sublists $2 \times T(\frac{n}{2})$
  - Assembling needs $n-1$ KC (in the WC)
  - $$T(n) = \begin{cases} 0 & , \quad \text{if} n = 1 \\ (n-1) + 2 \cdot T(\frac{n}{2}) & , \quad \text{otherwise} \end{cases}$$
- Solving recurrence relation:

Merge sort analysis — solving the recurrence relation

- Particular case:
  $T(1) = 0$,
  $T(2) = 1$,
  $\ldots$
- General case:

$$
\begin{aligned}
T(n) &= (n-1) + 2 \times T(\tfrac{n}{2}) \\
&= (n-1) + 2 \times \left((\tfrac{n}{2} - 1) + 2 \times T(\tfrac{n}{4})\right) \\
&= \ldots
\end{aligned}
$$

$$
\begin{aligned}
T(2^k) &= 2^k - 2^0 \\
&+ 2^k - 2^1 \\
&+ 2^k - 2^2 \\
&+ \cdots \\
&+ 2^k - 2^{k-1} \\
&= 2^k - T(2^0) \\
&= (k+1)2^k \\
&= k \cdot 2^k - \left(\sum^{k-1} 2^i\right) + 2^k \cancel{0}
\end{aligned}
$$

$j=0$

$= k2^k - (2^k - 1)$

Solving Merge Sort (Cont'd)

- We assume $n = 2^k$ so:

$$
\begin{aligned}
T(2^k) &= (2^k - 1) + 2 \times T(2^{k-1}) \\
&= (2^k - 1) + 2 \times \left((2^{k-1} - 1) + 2 \times T(2^{k-2})\right) \\
&= (2^k - 1) + (2^k - 2) + 2^2 \times T(2^{k-2}) \\
&= (2^k - 1) + (2^k - 2) + 2^2 \times \left((2^{k-2} - 1) + 2 \times T(2^{k-3})\right) \\
&= (2^k - 1) + (2^k - 2) + (2^k - 2^2) + 2^3 \times T(2^{k-3}) \\
&= (2^k - 2^0) + (2^k - 2^1) + (2^k - 2^2) + 2^3 \times T(2^{k-3}) \\
&= (2^k - 2^0) + (2^k - 2^1) + (2^k - 2^2) + (2^k - 2^3) + 2^4 \times T(2^{k-4}) \\
&= \ldots \\
&= (2^k - 2^0) + (2^k - 2^1) + (2^k - 2^2) + \ldots + (2^k - 2^{k-1}) + 2^k \times T(2^{k-k}) \\
&= (2^k - 2^0) + (2^k - 2^1) + (2^k - 2^2) + \ldots + (2^k - 2^{k-1}) \\
&= k \times 2^k - \textstyle\sum_{i=0}^{k-1} 2^i \\
&= (k-1)2^k + 1 \qquad \text{by applying } \textstyle\sum_{i=0}^{k-1} 2^i = 2^k - 1
\end{aligned}
$$

Since $n = 2^k$, we have $k = \lg n$. So, $T(n) = n(\lg n - 1) + 1$.

1. Variable substitution makes guessing easy ...
2. In recurrence solving always assume n being some power whenever necessary (ignore floor and ceiling).
3. Prove by induction.
4. Need to transform back to original variable.

Closed form proof by induction:

- Recurrence: $T(n) = \begin{cases} 0 & \text{if } n = 1 \\ (n-1) + 2 \times T(\frac{n}{2}) & \text{if } n \geq 2 \end{cases}$
  Guessed closed form: $T(n) = n(\lg n - 1) + 1, n \geq 1$
- Assuming $n = 2^k, k \geq 0$
- Base case: $T(1) = 0$ and indeed $1(\lg(1) - 1) + 1 = 0$.
- Inductive step: Assuming that $T(2^k) = 2^k(k-1) + 1$, $k \geq 0$, want to show $T(2^{k+1}) = 2^{k+1}k + 1$.
  By recurrence relation,

$$\begin{aligned} T(2^{k+1}) &= (2^{k+1} - 1) + 2 \times T(2^k) \\ &= (2^{k+1} - 1) + 2^{k+1}(k-1) + 2 \\ &= k2^{k+1} + 1. \quad \blacksquare \end{aligned}$$

- Extending to n which isn't a power of 2 is just tedious: if n is not a power of 2, $\exists$ integer k s.t. $n \leq 2^k < 2n$. So: $T(n) \leq T(2^k) = 2^k(k-1)+1 \leq (2n) \cdot (\lg(2n) - 1) + 1 = 2n\lg(n) + 1 = O(n\log(n))$.

$T(n) = 2^k(k-1)+1$
$= n(\lg n - 1) + 1$
$= O(n \log n)$

Running Time Analysis:

- We now wish to deduce that WC running time is $\Theta(n \log n)$
- Which direction is obvious?
  - Lower bound: if each KC takes one "unit of time" then our running time is $n(\lg(n) - 1) + 1 \geq \frac{1}{2} n \lg(n) \in \Omega(n \lg(n))$
- Upper bound: `Merge` takes $O(n)$ times. So $\exists c_1$ such that its running time $\leq c_1 n$.
- `Merge-Sort` takes $O(1)$ time in the base case and $O(1)$ for the two recursive calls to make.
- Hence, if $R(n)$ denotes the running time of `Merge-Sort` on n-size input, we have
  $$R(n) = \begin{cases} c_3, & \text{if } n = 1 \\ c_1 \cdot n + c_2 + 2R(\frac{n}{2}), & \text{if } n \geq 2 \end{cases}$$
- Set C to be any number $\geq c_1 + c_2 + c_3$ and we get
  $R(n) \leq \begin{cases} C, & \text{if } n = 1 \\ Cn + 2R(\frac{n}{2}), & \text{if } n \geq 2 \end{cases}$ and this recursion solves to $C \cdot n(\lg(n))$.
- Conclusion: merge sort WC running time is $\Theta(n \log n)$.

$3n^3+2n^2+5n \in O(n^3)$

$\leq 6n^3$

$0.1n^3 \leq \Omega(n^3)$

$\geq 0.1n^3$

$\log(n!) \in \Theta(n\log n)$

$\underbrace{\log(n)+\log(n-1)+\dots+1}_{n \text{ many}}$

$\in O(n\log n)$

Conclusions

- Divide-and-conquer algorithm often recursive
- Analysis of recursive algorithm $\Longrightarrow$ solving recurrence

An exercise:

- Examine the running time of $\texttt{QZ}(n)$

  Proc $\texttt{QZ}(n)$

  if $n > 1$ then
      $a = n \times n + 37$
      $b = a \times \texttt{QZ}(\frac{n}{2})$
      return $\texttt{QZ}(\frac{n}{2}) \times \texttt{QZ}(\frac{n}{2}) + n$
  else
      return $n \times n$

- If we only consider arithmetic operations then:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 3\texttt{QZ}(\frac{n}{2}) + 5 & \text{if } n \geq 2 \end{cases}$$

- Again, we use Iterated Substitution to obtain a proper guess
- Then we prove our guess by induction

$\log n + \log(n-1) + \dots + \log(\frac{n}{2}) + \dots$

$\geq \frac{n}{2}\log(\frac{n}{2}) = \frac{n}{2}(\log n - 1)$

$= \frac{n}{4}\log n \in \Omega(n\log n)$

Exercise (Cont'd):

- For simplicity, assume n is a power of 2, say $n = 2^k$:
- $$\begin{aligned} T(2^k) &= 3 \times T(2^{k-1}) + 5 \\ &= 3 \times \left(3 \times T(2^{k-2}) + 5\right) + 5 \\ &= 3^2 \times T(2^{k-2}) + 3 \times 5 + 5 \\ &= 3^2 \times \left(3 \times T(2^{k-3}) + 5\right) + 3 \times 5 + 5 \\ &= 3^3 \times T(2^{k-3}) + 3^2 \times 5 + 3 \times 5 + 5 \\ \\ &= \ldots \\ &= 3^k \times T(2^{k-k}) + 3^{k-1} \times 5 + 3^{k-2} \times 5 + \ldots + 3 \times 5 + 5 \\ &= 3^k + 5 \times \left(\sum_{i=0}^{k-1} 3^i\right) \\ &= 3^k + 5 \times \left(\frac{3^k - 1}{2}\right) \\ &= 3.5 \times 3^k - 2.5 \end{aligned}$$
- So, our guess is: $T(n) = 3.5 \times 3^{\log n} - 2.5 = 3.5 \times n^{\log 3} - 2.5$.

Exercise (Cont'd):

- Next, prove $T(2^k) = 3.5 \times 3^k - 2.5$, for $k \geq 0$, by induction
- Base step: $k = 0$ and $T(2^0) = 1 = 3.5 - 2.5$.
- Inductive step: Assume that $T(2^{k-1}) = 3.5 \times 3^{k-1} - 2.5$.
  By recurrence relation

  $$T(2^k) = 3 \times T(\frac{2^k}{2}) + 5 = 3 \times T(2^{k-1}) + 5,$$

  so

  $$T(2^k) = 3 \times \left(3.5 \times 3^{k-1} - 2.5\right) + 5 = 3.5 \times 3^k - 2.5.$$

  Thus, it holds for inductive step too.
- Therefore, $T(2^k) = 3.5 \times 3^k - 2.5$ holds for any $k \geq 0$.

n

$\frac{1}{2}n$ $\frac{1}{2}n$

$\frac{1}{4}n$ $\frac{1}{4}n$ ...

...

2- Recurrence Tree

- Another method to solve a recurrence (and analyze the running time of a recursive program)
- Not as formal as iterated substitution; more visual
- Consider the Merge-Sort and the tree for the recursive calls of it:

$$(1, n)$$

$$(1, \tfrac{n}{2}) \qquad (\tfrac{n}{2}+1, n)$$

$$(1, \tfrac{n}{4}) \quad (\tfrac{n}{4}+1, \tfrac{n}{2}) \quad (\tfrac{n}{2}+1, \tfrac{3n}{4}) \quad (\tfrac{3n}{4}+1, n)$$

$$\cdots\cdots\cdots$$

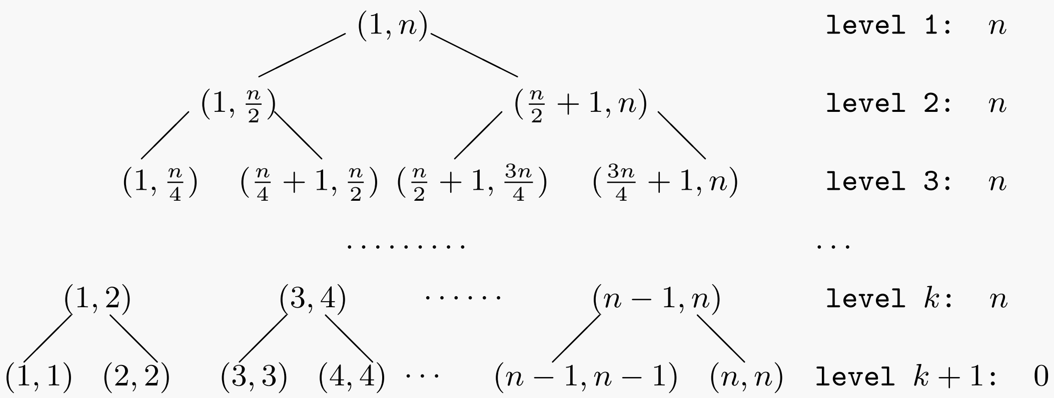
$$(1,2) \qquad (3,4) \qquad \cdots\cdots \qquad (n-1, n)$$

$$(1,1) \quad (2,2) \quad (3,3) \quad (4,4) \cdots \quad (n-1, n-1) \quad (n, n)$$

- Question: the number of KC per cell?

Merge sort recursion tree (KC per cell):

- Assuming `merge`(n) takes $\sim n$ KC:



Where $k = \log n$.

total: kn

- Therefore, the running time of Merge-Sort is (as found before): $\Theta(n \log n)$.
- Note: the recurrence tree method is not as applicable nor as formal as the iterated substitution.

3- Guess and Test method:

- First make a guess for the closed form of the recurrence
- Guess can come from the iterated substitution, recurrence tree, or previous experiences
  - **But regardless of the method, your guess must be verified!**
- Prove the guess by induction
- May have to change the guess if the inductive proof fails
- **Example:** Find a closed form for

  $$T(n) = \begin{cases} T(\frac{n}{2}) + 2T(\frac{n}{4}) + 2n & \text{if } n \geq 4 \\ n & \text{if } 1 \leq n \leq 3 \end{cases}$$

- **Solution:** We guess that $T(n) \in \Theta(n \log n)$
- Need to show that there are constants $c, d > 0$ and naturals n_0, n_1 such that:
  (i) $T(n) \leq cn \log n$ for any $n \geq n_0$, and
  (ii) $T(n) \geq dn \log n$ for any $n \geq n_1$.

- (i) Base case: $T(4) = T(2) + T(1) + 8 = 2 + 1 + 8 = 11$ so $T(4) \leq c \cdot 4 \cdot \lg(4) = 8c$ for $c \geq 11/8$.
- Assume $T(i) \leq ci \log i$ for all values of $i < n$, with $i \geq 4$. (Note the use of full induction!)

$$
\begin{aligned}
T(n) &= T(\tfrac{n}{2}) + 2T(\tfrac{n}{4}) + 2n \\
&\leq c\tfrac{n}{2} \log \tfrac{n}{2} + 2c\tfrac{n}{4} \log \tfrac{n}{4} + 2n \\
&\leq c\tfrac{n}{2}(\log n - 1) + c\tfrac{n}{2}(\log n - 2) + 2n \\
&= cn \log n - \tfrac{3cn}{2} + 2n \leq cn \log n,
\end{aligned}
$$

if we take $c \geq \frac{4}{3}$. Now we prove by induction that for $c = \frac{11}{8} = \max\{\frac{4}{3}, \frac{11}{8}\}$ and $n \geq 4$: $T(n) \leq \frac{11}{8} \cdot n \log n$ (left as an exercise).

- Note that we could have started with a guess of $c = 100$ and the induction would follow through too...

- (ii) $T(n) \geq \frac{1}{100} n \log n$ for any $n \geq 4$.
- Base case: $T(4) = 11 \geq \frac{1}{100} \cdot 4 \lg(4)$.
- Induction step: Assume $T(i) \geq \frac{1}{100} i \log i$ for all values of $i < n$, with $i \geq 4$.

$$
\begin{aligned}
T(n) &= T(\tfrac{n}{2}) + 2T(\tfrac{n}{4}) + 2n \\
&\geq \tfrac{1}{100} \cdot \tfrac{n}{2} \log \tfrac{n}{2} + 2 \cdot \tfrac{1}{100} \cdot \tfrac{n}{4} \log \tfrac{n}{4} + 2n \\
&\geq \tfrac{n}{200} (\log n - 1 + \log n - 2) + 2n \\
&\geq \tfrac{2n \log(n)}{200} + \left(2 - \tfrac{3}{100}\right) n \\
&\geq \tfrac{1}{100} n \log n
\end{aligned}
$$

- Combining (i) + (ii) we get: $T(n) \in \Theta(n \log n)$.

- Note: Sometimes we need to revise our guess
- The correct guess is not always obvious; the method requires practice

$n^{\log_b a}$ vs $f(n)$

merge sort:

$n^{\log_b a} = n$

$f(n) \in \Theta(n)$

$\Rightarrow \Theta(n \log n)$

$n^{\log_2 4} = n^2$ $f(n) \in \Theta(n)$

$\Rightarrow \Theta(n^{\log_b a}) = \Theta(n^2)$

$n^{\log_2 4} = n^2$ $f(n) \in \Theta(1)$

4- Master Theorem Method

Some recurrences can be solved conveniently by the master theorem method, which depends on the Master Theorem.

- **Master Theorem:**
  Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function.
  Let $T(n)$ be defined on nonnegative integers by the recurrence

$$T(n) = aT(\tfrac{n}{b}) + f(n).$$

Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) \in O(n^{\log_b a - \epsilon})$, for some $\epsilon > 0$ then $T(n) \in \Theta(n^{\log_b a})$.
2. If $f(n) \in \Theta(n^{\log_b a} \log^k n)$ for some $k \geq 0$ then $T(n) \in \Theta(n^{\log_b a} \log^{k+1} n)$,
3. If $f(n) \in \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(\frac{n}{b}) \leq \delta f(n)$ for some constant $\delta < 1$ and all sufficiently large n, then $T(n) \in \Theta(f(n))$.

$T(n) = 7T(\frac{n}{4}) + n^2$

$n^{\log_4 7} = n^{1.xx}$

$7(\frac{n}{4})^2$

Some examples:

1. $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 7T(\frac{n}{2}) + n^2 & \text{if } n \geq 2 \end{cases}$

   $a = 7,\ b = 2,\ f(n) = n^2 \Rightarrow \log_b a = \log_2 7 > 2.8$, so $f(n) \in O(n^{\log 7 - 0.1})$ and $T(n) \in \Theta(n^{\log 7})$

2. $T(n) = \begin{cases} 1 & \text{if } n \leq 2 \\ 14T(\frac{n}{3}) + n^3 & \text{if } n \geq 3 \end{cases}$

   $a = 14, b = 3, f(n) = n^3 \Rightarrow \log_b a = \log_3(14) \in (2, 3)$, since $f(n) \in \Omega(n^{\log_3(14) + \epsilon})$ for $\epsilon = \frac{3 - \log_3(14)}{2}$, and $14\left(\frac{n}{3}\right)^3 \leq \frac{14}{27} n^3$ (i.e. with $\delta = 2/3$ in case 3) $T(n) \in \Theta(n^3)$

   $a = 14 \quad b = 3 \quad d = 3$

   $b^d = \qquad > 0$

3. $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(\frac{n}{2}) + n & \text{if } n \geq 2 \end{cases}$

   $a = 2,\ b = 2,\ f(n) = n$, since $n^{\log_b a} = n = f(n)$, we have $f(n) \in \Theta(n^{\log_2 2})$ and so (by case 2) $T(n) \in \Theta(n \log n)$.

4. $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 5T(\frac{n}{2}) + n^2 \log n & \text{if } n \geq 2 \end{cases}$

   $a = 5,\ b = 2,\ f(n) = n^2 \log n$. So $\log_b a = \log 5 > 2.3$, so $f(n) \in O(n^{\log 5 - 0.1})$ and $T(n) \in \Theta(n^{\log 5})$

if $T(n) = aT(\frac{n}{b}) + \Theta(n^d)$
where $a > 0$, $b > 1$ and $d \geq 0$

$T(n) \in \Theta(n^d)$ if $a < b^d$

$T(n) \in \Theta(n^d \log n)$ if $a = b^d$

$T(n) \in \Theta(n^{\log_b a})$ if $a > b^d$

Master Theorem doesn't always apply:

$\frac{n^2}{\log n}$ vs $\frac{n^2}{n^\epsilon}$

$$T(n) = \begin{cases} 4T(\frac{n}{2}) + \frac{n^2}{\log n} & \text{if } n \geq 2 \\ 1 & \text{if } n = 1 \end{cases}$$

$a = 4, b = 2, \log_b a = 2$

$f(n) = \frac{n^2}{\log n} \notin \Theta(n^2)$;

$f(n) = \frac{n^2}{\log n} \in O(n^2)$ but $f(n) = \frac{n^2}{\log n} \notin O(n^{2-\epsilon})$ for any $\epsilon > 0$.

What can we do to get the closed form?

— iterated substitution!

$$
\begin{aligned}
& T(2^k) \\
= \quad & 4 \times T(2^{k-1}) + \tfrac{2^{2k}}{k} \\
= \quad & 4^2 \times T(2^{k-2}) + 4 \times \tfrac{2^{2(k-1)}}{k-1} + \tfrac{2^{2k}}{k} \\
= \quad & 4^2 \times T(2^{k-2}) + \tfrac{2^{2k}}{k-1} + \tfrac{2^{2k}}{k} \\
\\
= \quad & 4^3 \times T(2^{k-3}) + 4^2 \times \tfrac{2^{2(k-2)}}{k-2} + \tfrac{2^{2k}}{k-1} + \tfrac{2^{2k}}{k} \\
= \quad & 4^3 \times T(2^{k-3}) + \tfrac{2^{2k}}{k-2} + \tfrac{2^{2k}}{k-1} + \tfrac{2^{2k}}{k} \\
\\
= \quad & 4^k \times T(1) + \tfrac{2^{2k}}{k-(k-1)} + \ldots + \tfrac{2^{2k}}{k-1} + \tfrac{2^{2k}}{k} \\
= \quad & 4^k \times T(1) + 2^{2k} \left(\tfrac{1}{1} + \tfrac{1}{2} + \tfrac{1}{3} + \ldots + \tfrac{1}{k}\right) \\
= \quad & 4^k \times T(1) + 4^k \times H(k)
\end{aligned}
$$

(Harmonic series: $H(n) = \left(1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n}\right) = \ln n + O(1) = \Theta(\log n)$, CLRS p.1147).

Therefore, $T(n) = n^2 \times T(1) + n^2 \times H(\log n) \in \Theta(n^2 H(\log n))$.
Further we have $H(k) \in \Theta(\log k)$ (in fact $H(k) = \ln k + \Theta(1)$), thus $T(n) \in \Theta(n^2 H(\log n)) = \Theta(n^2 \log\log n)$.

An exercise — dealing with floor & ceiling:

Prove that $T(n)$ defined by the following recurrence is in $O(\log n)$:

$$T(n) = \begin{cases} 1, & \text{if } n = 1, \\ T(\lceil \frac{n}{2} \rceil) + 1, & \text{if } n \geq 2 \end{cases}$$

- Examine some small cases:
  $T(1) = 1$
  $T(2) = 2$
  $T(3) = T(4) = 3$
  $T(5) = T(6) = T(7) = T(8) = 4$
  ...
  Guess: $T(n) = k + 1$, for any $2^{k-1} < n \leq 2^k$
- Prove the above guessed (by induction).
- Now you only need to get the closed form for n being a power of 2 ...
- By iterated substitution, $T(2^k) = k + 1$ (again, prove by induction)
  So, $T(n) = \log n + 1$ for any n which is a power of 2.
- Now, prove by induction on k that for any n satisfying $2^{k-1} < n \leq 2^k$ we have $T(n) = k + 1$.
- Conclusion: since $T(n) = \lceil \log n \rceil + 1 \leq \log n + 2 \leq 2 \log n$, for $n \geq 4$, $T(n) \in O(\log n)$