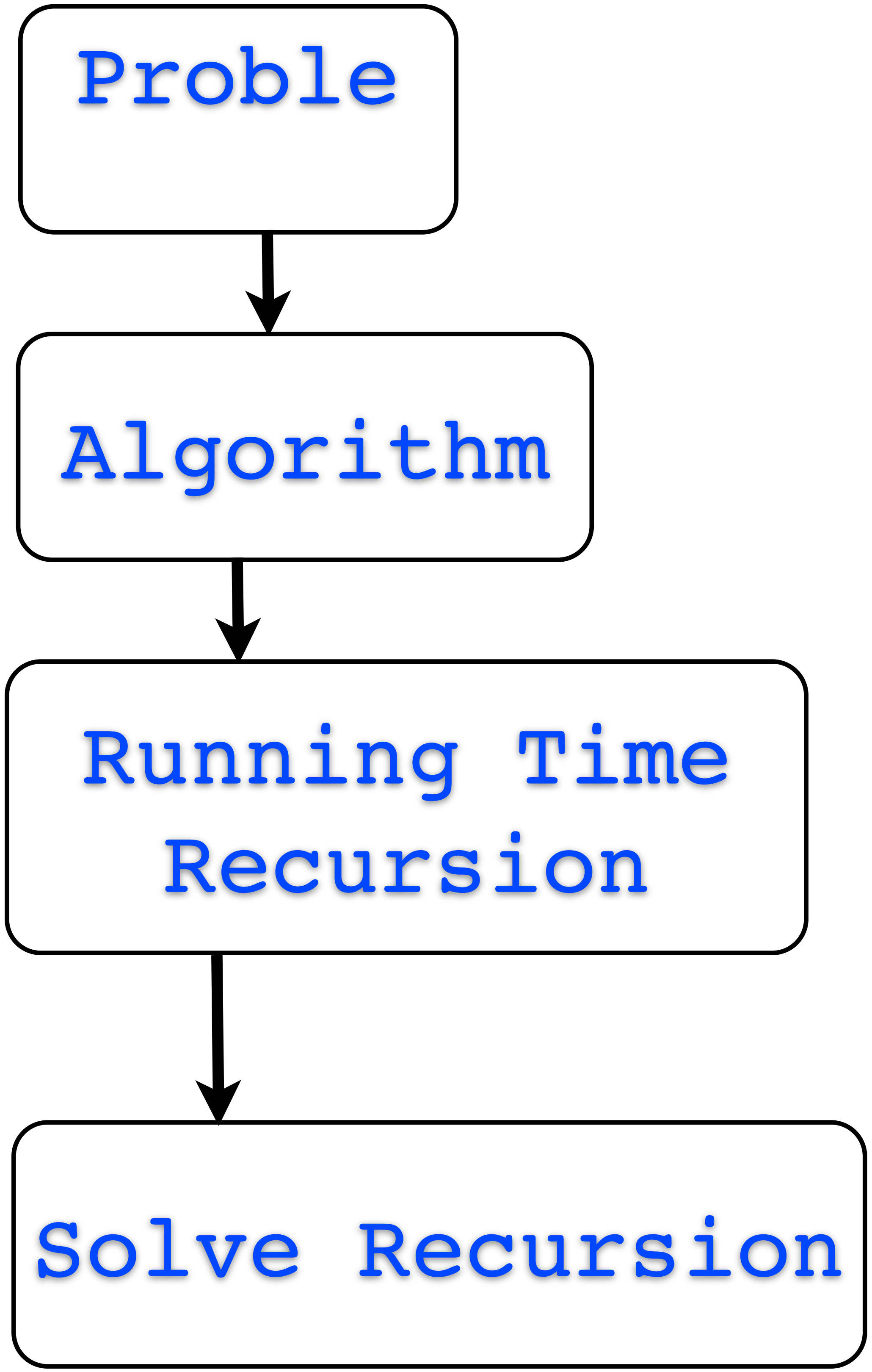
# Recurrences

# Objective

- running time as recursive function
- solve recurrence for order of growth
- method: substitution
- method: iteration/recursion tree
- method: MASTER method

- prerequisite:
  - mathematical induction, recursive definitions
  - arithmetic manipulations, series, products

# Pipeline



# Running time

- will call it T(n) = number of computational steps required to run the algorithm/program for input of size n
- we are interested in order of growth, not exact values
  - for example $T(n) = \Theta(n^2)$ means quadratic running time
  - $T(n) = O(n \log n)$ means T(n) grows not faster than CONST*n*log(n)
- for simple problems, we know the answer right away
  - example: finding MAX of an array
  - solution: traverse the array, keep track of the max encountered
  - running time: one step for each array element, so n steps for

# Running time for complex problems

- complex problems involve solving subproblems, usually
  - init/prepare/preprocess, define subproblems
  - solve subproblems
  - put subproblems results together
- thus T(n) cannot be computed straight forward
  - instead, follow the subproblem decomposition

# Running time for complex problems

- often, subproblems are the same problem for a smaller input size:
  - for example max(array) can be solved as:
    - split array in array_Left, array_Right
    - solve max(array_Left), max (array_Right)
    - combine results to get global max

```
Max(A=[a1,a2,... ,an])
    if (n==1) return a1
    k= n/2
    max_left = Max([a1,a2,..,ak])
    max_right = Max([ak+1,ak+2,..,an])
    if(max_left>max_right) return max_left
    else return max_right
```

# Running time for complex problems

- many problems can be solved using a divide-and-conquer strategy
  - prepare, solve subproblems, combine results
- running time can be written recursively
  - T(n) = time(preparation) + time(subproblems) + time(combine)
  - for MAX recursive: $T(n) = 2*T(n/2) + O(1)$

# Running time for complex problems

- many problems can be solved using a divide-and-conquer strategy
  - prepare, solve subproblems, combine results
- running time can be written recursively
  - T(n) = time(preparation) + time(subproblems) + time(combine)
  - for MAX recursive: $T(n) = 2*T(n/2) + O(1)$

2 subproblems of size n/2
max(array_Left) ; max(array_Right)

# Running time for complex problems

- many problems can be solved using a divide-and-conquer strategy
  - prepare, solve subproblems, combine results
- running time can be written recursively
  - T(n) = time(preparation) + time(subproblems) + time(combine)
  - for MAX recursive: T(n) = 2*T(n/2) + O(1)

2 subproblems of size n/2
max(array_Left) ; max(array_Right)

constant time to check the maximum
out of the two max Left and Right

# Recurrence examples

- $T(n) = 2T(n/2) + O(1)$
- $T(n) = 2T(n/2) + O(n)$
  - 2 subproblems of size n/2 each, plus O(n) steps to combine results
- $T(n) = 4T(n/3) + n$
  - 4 subproblems of size n/3 each, plus n steps to combine results
- $T(n/4) + T(n/2) + n^2$
  - a subproblem of size n/3, another of size n/2; $n^2$ to combine
- want to solve such recurrences, to obtain the order of growth of function T

# Substitution method

- $T(n) = 4T(n/2) + n$
- STEP1 : guess solution, order of growth $T(n) = O(n^3)$
  - that means there is a constant C and a starting value $n_0$, such that $T(n) \leq Cn^3$, for any $n \geq n_0$
- STEP2: verify by induction
  - assume $T(k) \leq k^3$, for $k<n$
  - induction step: prove that $T(n) \leq Cn^3$, using $T(k) \leq Ck^3$, for $k<n$

$$
\begin{aligned}
T(n) &= 4T(\frac{n}{2}) + n && (1)\\
&\leq 4c\left(\frac{n}{2}\right)^3 + n && (2)\\
&= \frac{c}{2}n^3 + n && (3)\\
&= cn^3 - (\frac{c}{2}n^3 - n) && (4)\\
&\leq cn^3; \quad if\ \frac{c}{2}n^3 - n > 0, choose \quad c \geq 2 && (5)
\end{aligned}
$$

# Substitution method

- STEP 3 : identify constants, in our case c=2 works
- so we proved $T(n)=O(n^3)$
- thats correct, but the result is too weak
  - technically we say the bound $O(n^3)$ "cubic" is too lose
  - can prove better bounds like T(n) "quadratic" $T(n)=O(n^2)$
  - Our guess was wrong ! (too big)
- lets try again : STEP1: guess $T(n)=O(n^2)$
- STEP2: verify by induction
  - assume $T(k) \leq Ck^2$, for $k<n$
  - induction step: prove that $T(n) \leq Cn^2$, using $T(k) \leq Ck^2$, for $k<n$

# Substitution method

- Fallacious argument

$$\begin{aligned} T(n) &= 4T(\frac{n}{2}) + n && (1) \\ &\le 4c\left(\frac{n}{2}\right)^2 + n && (2) \\ &= cn^2 + n && (3) \\ &= O(n^2) && (4) \\ &\le cn^2 && (5) \end{aligned}$$

  - cant prove $T(n)=O(n^2)$ this way: need same constant steps 3-4-5
  - maybe its not true? Guess $O(n^2)$ was too low?
  - or maybe we dont have the right proof idea
- common trick: if math doesnt work out, make a stronger assumption (subtract a lower degree term)
  - assume instead $T(k) \le C_1 k^2 - C_2 k$, for $k<n$

# Substitution method

$$
\begin{aligned}
T(n) &= 4T(\frac{n}{2}) + n \\
&\leq 4\left(c_1\left(\frac{n}{2}\right)^2 - c_2\frac{n}{2}\right) + n \\
&= c_1n^2 - 2c_2n + n \\
&= c_1n^2 - c_2n - (c_2n - n) \\
&\leq c_1n^2 - c_2n \quad for \quad c_2 > 1
\end{aligned}
$$

- So we can prove $T(n)=O(n^2)$, but is that asymptotically correct ?
  - maybe we can prove a lower upper bound, like O(nlogn)? NOPE
- to make sure its the asymptote, prove its also the lower bound
  - $T(n) = \Omega(n^2)$ or there is a different constant d s.t. $T(n) \geq dn^2$

# Substitution method: lower bound

- induction step

$$\begin{aligned} T(n) &= 4T(\frac{n}{2}) + n \\ &\geq 4d\left(\frac{n}{2}\right)^2 + n \\ &= dn^2 + n \geq dn^2 \end{aligned}$$

- now we know its asymptotically close, $T(n)=\Theta(n^2)$
- hard to make the initial guess $\Theta(n^2)$
  - need another method to educate our guess

# Iteration method

$$
\begin{aligned}
& T(n) = n + 4T(\frac{n}{2}) \\
= \; & n + 4(\frac{n}{2} + 4T(\frac{n}{4})) = n + 2n + 4^2 T(\frac{n}{2^2}) \\
= \; & n + 2n + 4^2(\frac{n}{2^2} + 4T(\frac{n}{2^3})) = n + 2n + 2^2 n + 4^3 T(\frac{n}{2^3}) \\
= \; & .... \\
= \; & n + 2n + 2^2 n + ... + 2^{k-1} n + 4^k T(\frac{n}{2^k}) \\
= \; & \sum_{i=0}^{k-1} 2^i n + 4^k T(\frac{n}{2^k});
\end{aligned}
$$

# Iteration method

$$
\begin{aligned}
& T(n) = n + 4T(\frac{n}{2}) \\
= \quad & n + 4(\frac{n}{2} + 4T(\frac{n}{4})) = n + 2n + 4^2T(\frac{n}{2^2}) \\
= \quad & n + 2n + 4^2(\frac{n}{2^2} + 4T(\frac{n}{2^3})) = n + 2n + 2^2n + 4^3T(\frac{n}{2^3}) \\
= \quad & .... \\
= \quad & n + 2n + 2^2n + ... + 2^{k-1}n + 4^kT(\frac{n}{2^k}) \\
= \quad & \sum_{i=0}^{k-1} 2^i n + 4^kT(\frac{n}{2^k});
\end{aligned}
$$

$$want\ \ k = \log(n) \Leftrightarrow \frac{n}{2^k} = 1$$

# Iteration method

$$
\begin{aligned}
& T(n) = n + 4T(\frac{n}{2}) \\
= \quad & n + 4(\frac{n}{2} + 4T(\frac{n}{4})) = n + 2n + 4^2T(\frac{n}{2^2}) \\
= \quad & n + 2n + 4^2(\frac{n}{2^2} + 4T(\frac{n}{2^3})) = n + 2n + 2^2n + 4^3T(\frac{n}{2^3}) \\
= \quad & .... \\
= \quad & n + 2n + 2^2n + ... + 2^{k-1}n + 4^kT(\frac{n}{2^k}) \\
= \quad & \sum_{i=0}^{k-1} 2^i n + 4^kT(\frac{n}{2^k});
\end{aligned}
$$

$$
want \;\; k = \log(n) \Leftrightarrow \frac{n}{2^k} = 1
$$

$$
\begin{aligned}
= \quad & n \sum_{i=0}^{\log(n)-1} 2^i + 4^{log(n)}T(1) \\
= \quad & n\frac{2^{log(n)} - 1}{2-1} + n^2T(1) \\
= \quad & n(n-1) + n^2T(1) = \Theta(n^2)
\end{aligned}
$$

# Iteration method

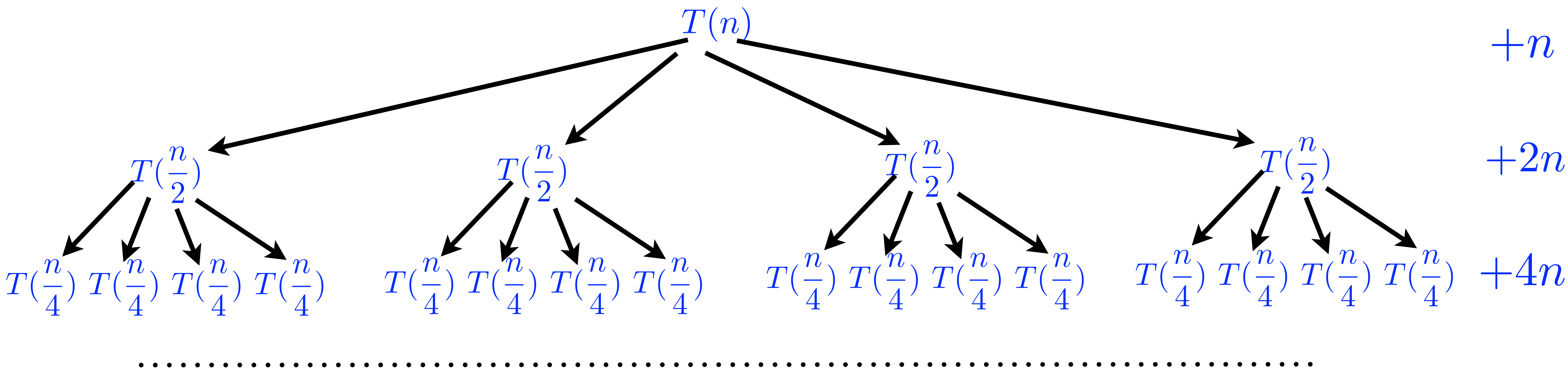
$$
\begin{aligned}
& T(n) = n + 4T(\frac{n}{2}) \\
= \; & n + 4(\frac{n}{2} + 4T(\frac{n}{4})) = n + 2n + 4^2 T(\frac{n}{2^2}) \\
= \; & n + 2n + 4^2(\frac{n}{2^2} + 4T(\frac{n}{2^3})) = n + 2n + 2^2 n + 4^3 T(\frac{n}{2^3}) \\
= \; & .... \\
= \; & n + 2n + 2^2 n + ... + 2^{k-1} n + 4^k T(\frac{n}{2^k}) \\
= \; & \sum_{i=0}^{k-1} 2^i n + 4^k T(\frac{n}{2^k});
\end{aligned}
$$

$$want \;\; k = \log(n) \Leftrightarrow \frac{n}{2^k} = 1$$

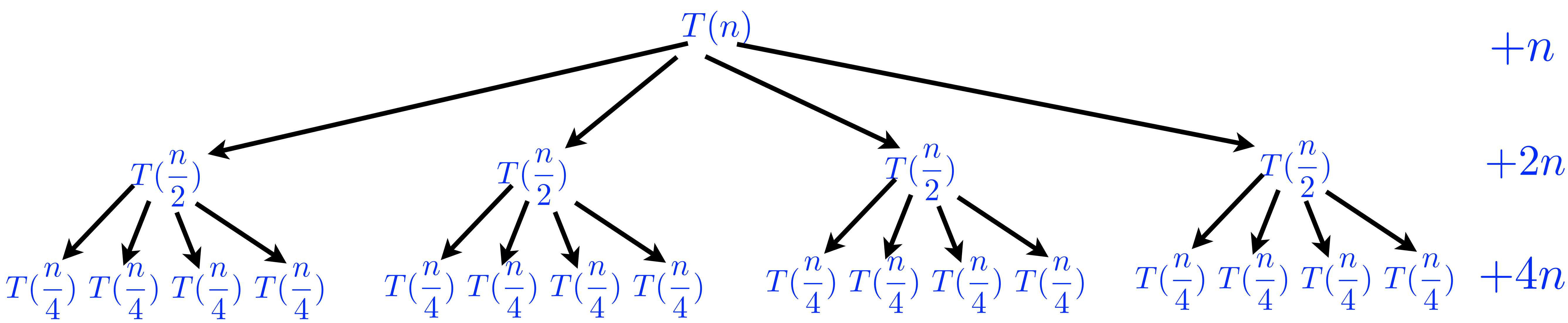
$$
\begin{aligned}
= \; & n \sum_{i=0}^{\log(n)-1} 2^i + 4^{log(n)} T(1) \\
= \; & n \frac{2^{log(n)} - 1}{2 - 1} + n^2 T(1) \\
= \; & n(n-1) + n^2 T(1) = \Theta(n^2)
\end{aligned}
$$

- math can be messy
  - recap sum, product, series, logarithms
  - iteration method good for guess, but usually unreliable for an exact result
  - use iteration for guess, and substitution for proofs
- stopping condition
  - T(...) = T(1), solve for k

# Iteration method: visual tree



# Iteration method: visual tree




- compute the tree depth: how many levels till nodes become leaves T(1)? log(n)
- compute the total number of leaves T(1) in the tree (last level): $4^{\log(n)}$
- compute the total additional work (right side) n+2n+4n+... = n(n-1)
- add the work $4^{\log(n)} + n(n-1) = \Theta(n^2)$

# Iteration Method : derivation

- $T(n) = n^2 + T(n/2) + T(n/4)$

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

$$= n^2 + (\frac{n}{2})^2 + T(\frac{n}{4}) + T(\frac{n}{8}) + (\frac{n}{4})^2 + T(\frac{n}{8}) + T(\frac{n}{16})$$

$$= n^2 + \frac{5}{16}n^2 + T(\frac{n}{4}) + 2T(\frac{n}{8}) + T(\frac{n}{16})$$

$$= n^2 + \frac{5}{16}n^2 + (\frac{n}{4})^2 + T(\frac{n}{8}) + T(\frac{n}{16}) + 2(\frac{n}{8})^2 + 2T(\frac{n}{16}) + 2T(\frac{n}{32}) + (\frac{n}{16})^2 + T(\frac{n}{32}) + T(\frac{n}{64})$$

$$= n^2 + \frac{5}{16}n^2 + (\frac{5}{16})^2 n^2 + T(\frac{n}{8}) + 3T(\frac{n}{16}) + 3T(\frac{n}{32}) + T(\frac{n}{64})$$

# Iteration Method : derivation

- $T(n) = n^2 + T(n/2) + T(n/4)$

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

$$= n^2 + (\frac{n}{2})^2 + T(\frac{n}{4}) + T(\frac{n}{8}) + (\frac{n}{4})^2 + T(\frac{n}{8}) + T(\frac{n}{16})$$

$$= n^2 + \frac{5}{16}n^2 + T(\frac{n}{4}) + 2T(\frac{n}{8}) + T(\frac{n}{16})$$

$$= n^2 + \frac{5}{16}n^2 + (\frac{n}{4})^2 + T(\frac{n}{8}) + T(\frac{n}{16}) + 2(\frac{n}{8})^2 + 2T(\frac{n}{16}) + 2T(\frac{n}{32}) + (\frac{n}{16})^2 + T(\frac{n}{32}) + T(\frac{n}{64})$$

$$= n^2 + \frac{5}{16}n^2 + (\frac{5}{16})^2 n^2 + T(\frac{n}{8}) + 3T(\frac{n}{16}) + 3T(\frac{n}{32}) + T(\frac{n}{64})$$

# Iteration Method : derivation

- $T(n) = n^2 + T(n/2) + T(n/4)$

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

$$= n^2 + (\frac{n}{2})^2 + T(\frac{n}{4}) + T(\frac{n}{8}) + (\frac{n}{4})^2 + T(\frac{n}{8}) + T(\frac{n}{16})$$

$$= n^2 + \frac{5}{16}n^2 + T(\frac{n}{4}) + 2T(\frac{n}{8}) + T(\frac{n}{16})$$

$$= n^2 + \frac{5}{16}n^2 + (\frac{n}{4})^2 + T(\frac{n}{8}) + T(\frac{n}{16}) + 2(\frac{n}{8})^2 + 2T(\frac{n}{16}) + 2T(\frac{n}{32}) + (\frac{n}{16})^2 + T(\frac{n}{32}) + T(\frac{n}{64})$$

$$= n^2 + \frac{5}{16}n^2 + (\frac{5}{16})^2 n^2 + T(\frac{n}{8}) + 3T(\frac{n}{16}) + 3T(\frac{n}{32}) + T(\frac{n}{64})$$

$$= n^2 + \frac{5}{16}n^2 + (\frac{5}{16})^2 n^2 + T(\frac{n}{16}) + T(\frac{n}{32}) + (\frac{n}{8})^2 + 3T(\frac{n}{32}) + 3T(\frac{n}{64}) + 3(\frac{n}{16})^2 + 3T(\frac{n}{64}) + 3T(\frac{n}{128}) + 3(\frac{n}{32})^2 + T(\frac{n}{128}) + T(\frac{n}{256}) + (\frac{n}{64})^2$$

# Iteration Method : derivation

- T(n) = n$^2$ + T(n/2) + T(n/4)

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

$$= n^2 + (\frac{n}{2})^2 + T(\frac{n}{4}) + T(\frac{n}{8}) + (\frac{n}{4})^2 + T(\frac{n}{8}) + T(\frac{n}{16})$$

$$= n^2 + \frac{5}{16}n^2 + T(\frac{n}{4}) + 2T(\frac{n}{8}) + T(\frac{n}{16})$$

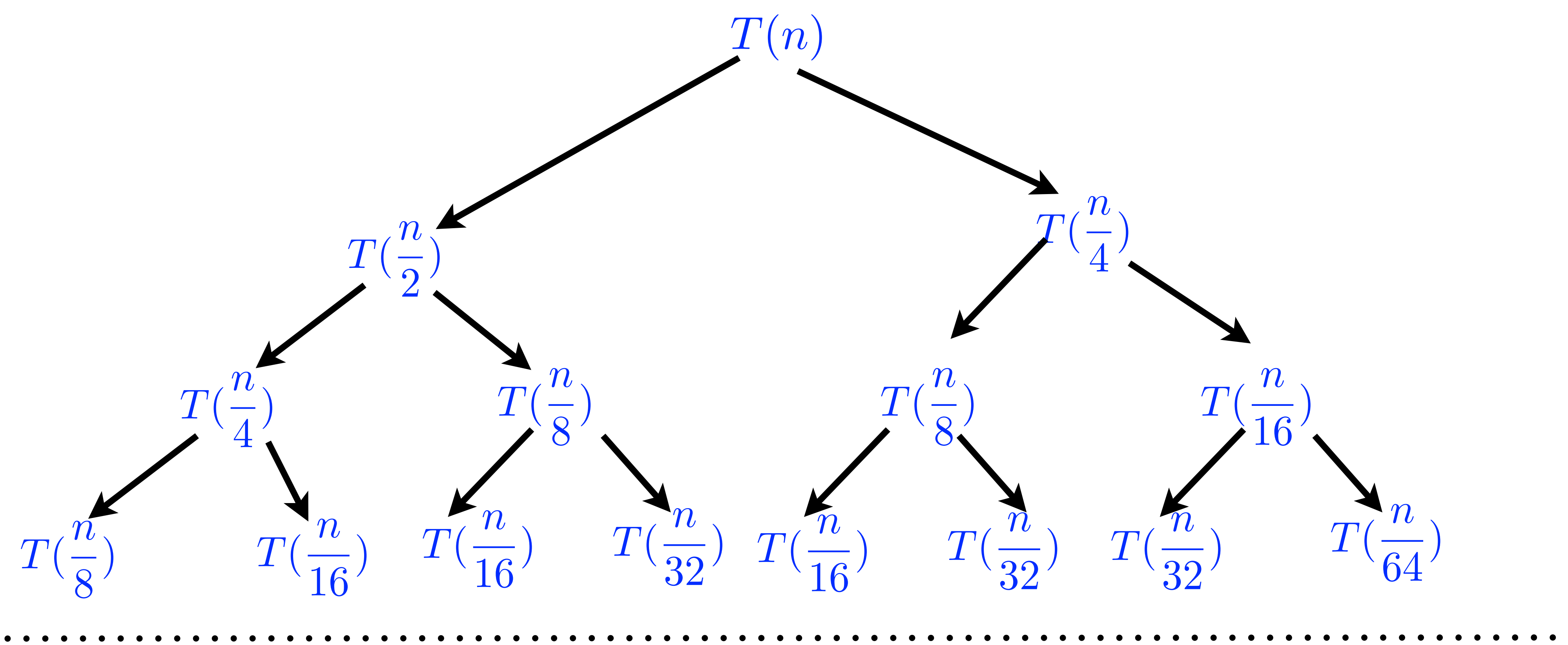
$$= n^2 + \frac{5}{16}n^2 + (\frac{n}{4})^2 + T(\frac{n}{8}) + T(\frac{n}{16}) + 2(\frac{n}{8})^2 + 2T(\frac{n}{16}) + 2T(\frac{n}{32}) + (\frac{n}{16})^2 + T(\frac{n}{32}) + T(\frac{n}{64})$$

$$= n^2 + \frac{5}{16}n^2 + (\frac{5}{16})^2 n^2 + T(\frac{n}{8}) + 3T(\frac{n}{16}) + 3T(\frac{n}{32}) + T(\frac{n}{64})$$

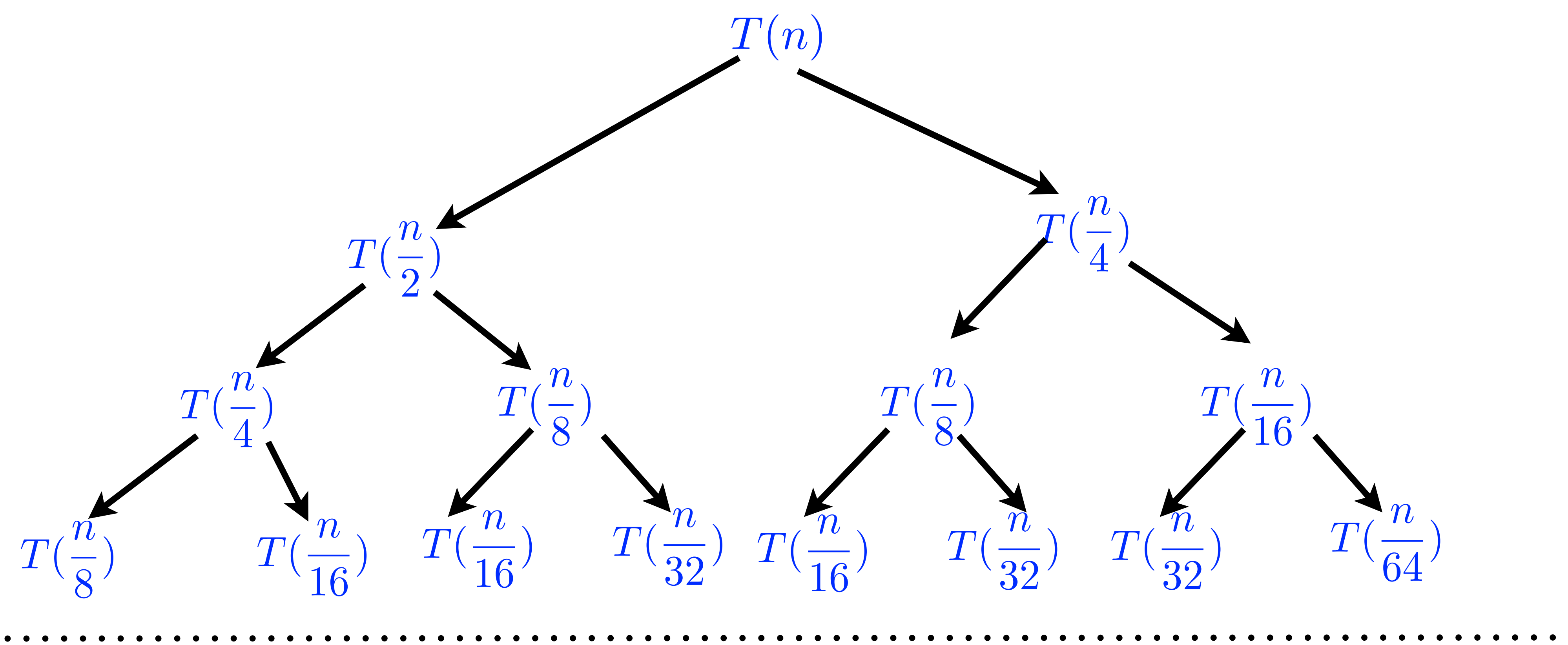
$$= n^2 + \frac{5}{16}n^2 + (\frac{5}{16})^2 n^2 + T(\frac{n}{16}) + T(\frac{n}{32}) + (\frac{n}{8})^2 + 3T(\frac{n}{32}) + 3T(\frac{n}{64}) + 3(\frac{n}{16})^2 + 3T(\frac{n}{64}) + 3T(\frac{n}{128}) + 3(\frac{n}{32})^2 + T(\frac{n}{128}) + T(\frac{n}{256}) + (\frac{n}{64})^2$$

$$= n^2 + \frac{5}{16}n^2 + (\frac{5}{16})^2 n^2 + (\frac{5}{16})^3 n^2 + T(\frac{n}{16}) + 4T(\frac{n}{32}) + 6T(\frac{n}{64}) + 4T(\frac{n}{128}) + T(\frac{n}{256})$$

# Iteration Method : tree



# Iteration Method : tree



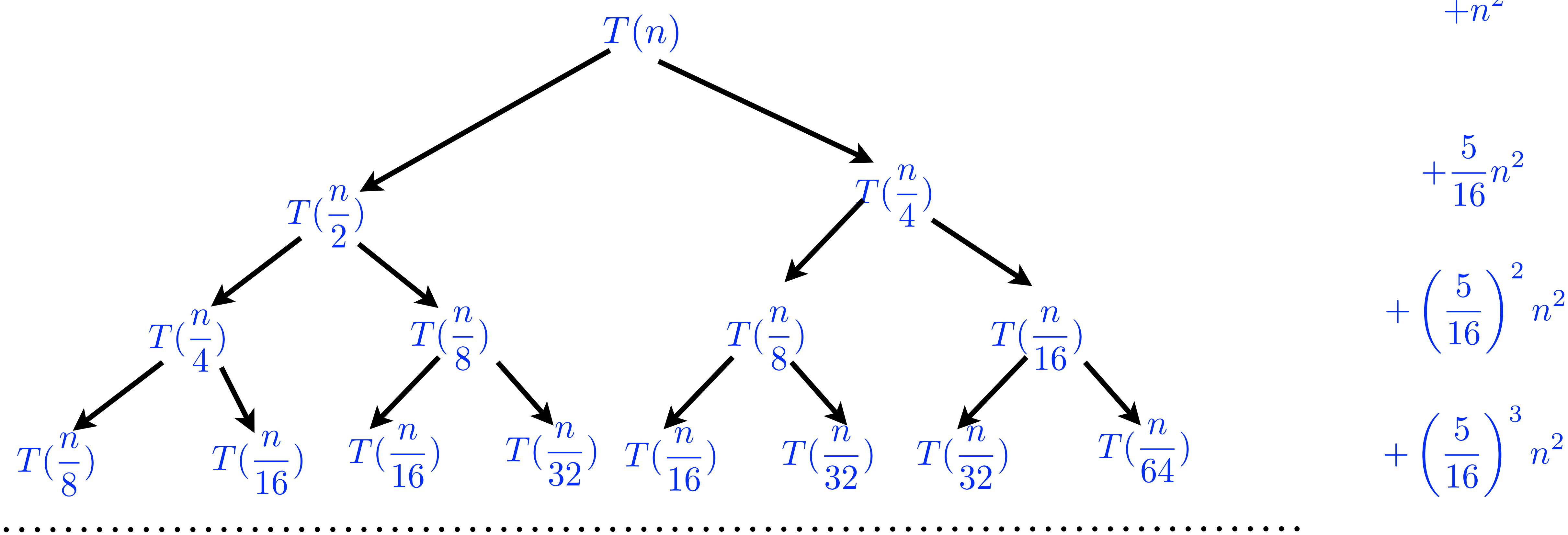

$$+n^2$$

$$+\frac{5}{16}n^2$$

$$+\left(\frac{5}{16}\right)^2 n^2$$

$$+\left(\frac{5}{16}\right)^3 n^2$$

# Iteration Method : tree




- depth : at most log(n)
- leaves: at most $2^{\log(n)}$ = n; computational cost nT(1) =O(n)
- work : $n^2 + \frac{5}{16}n^2 + (\frac{5}{16})^2 n^2 + (\frac{5}{16})^3 n^2 + ... \leq n^2 \sum_{i=0}^{\infty} (\frac{5}{16})^i = \frac{16}{11} n^2 = \Theta(n^2)$
- total $\Theta(n^2)$

# Master Theorem - simple

- simple general case $T(n) = aT(n/b) + \Theta(n^c)$

# Master Theorem – simple

- simple general case $T(n) = aT(n/b) + \Theta(n^c)$
- $R=a/b^c$, compare R with 1, or c with $\log_b(a)$

# Master Theorem – simple

- simple general case $T(n) = aT(n/b) + \Theta(n^c)$
- $R=a/b^c$, compare R with 1, or c with $\log_b(a)$

| | | |
|---|---|---|
| **Case 1:** | $c < \log_b a$ | $T(n) = \Theta(n^{\log_b a})$ |
| **Case 2:** | $c = \log_b a$ | $T(n) = \Theta(n^c \log n) = \Theta(n^{\log_b a} \log n)$ |
| **Case 3:** | $c > \log_b a$ | $T(n) = \Theta(n^c)$ |

# Master Theorem – simple

- simple general case $T(n) = aT(n/b) + \Theta(n^c)$
- $R=a/b^c$, compare R with 1, or c with $\log_b(a)$

| | | |
|---|---|---|
| **Case 1:** | $c < \log_b a$ | $T(n) = \Theta(n^{\log_b a})$ |
| **Case 2:** | $c = \log_b a$ | $T(n) = \Theta(n^c \log n) = \Theta(n^{\log_b a} \log n)$ |
| **Case 3:** | $c > \log_b a$ | $T(n) = \Theta(n^c)$ |

  - MergeSort $T(n) = 2T(n/2) + \Theta(n)$; $a=2$ $b=2$ $c=1$

    case 2 ; $T(n) = \Theta(n\log n)$

# Master Theorem – simple

- simple general case $T(n) = aT(n/b) + \Theta(n^c)$
- $R=a/b^c$, compare R with 1, or c with $\log_b(a)$

| | | |
|---|---|---|
| **Case 1:** | $c < \log_b a$ | $T(n) = \Theta(n^{\log_b a})$ |
| **Case 2:** | $c = \log_b a$ | $T(n) = \Theta(n^c \log n) = \Theta(n^{\log_b a} \log n)$ |
| **Case 3:** | $c > \log_b a$ | $T(n) = \Theta(n^c)$ |

- MergeSort $T(n) = 2T(n/2) + \Theta(n)$; a=2 b=2 c=1

  case 2 ; $T(n) = \Theta(n\log n)$
- Strassen's $T(n) = 7T(n/2) + \Theta(n^2)$ ; a=7 b=2 c=2

  case 1, $T(n) = \Theta(n^{\log_2(7)})$

# Master Theorem – simple

- simple general case $T(n) = aT(n/b) + \Theta(n^c)$
- $R=a/b^c$, compare R with 1, or c with $\log_b(a)$

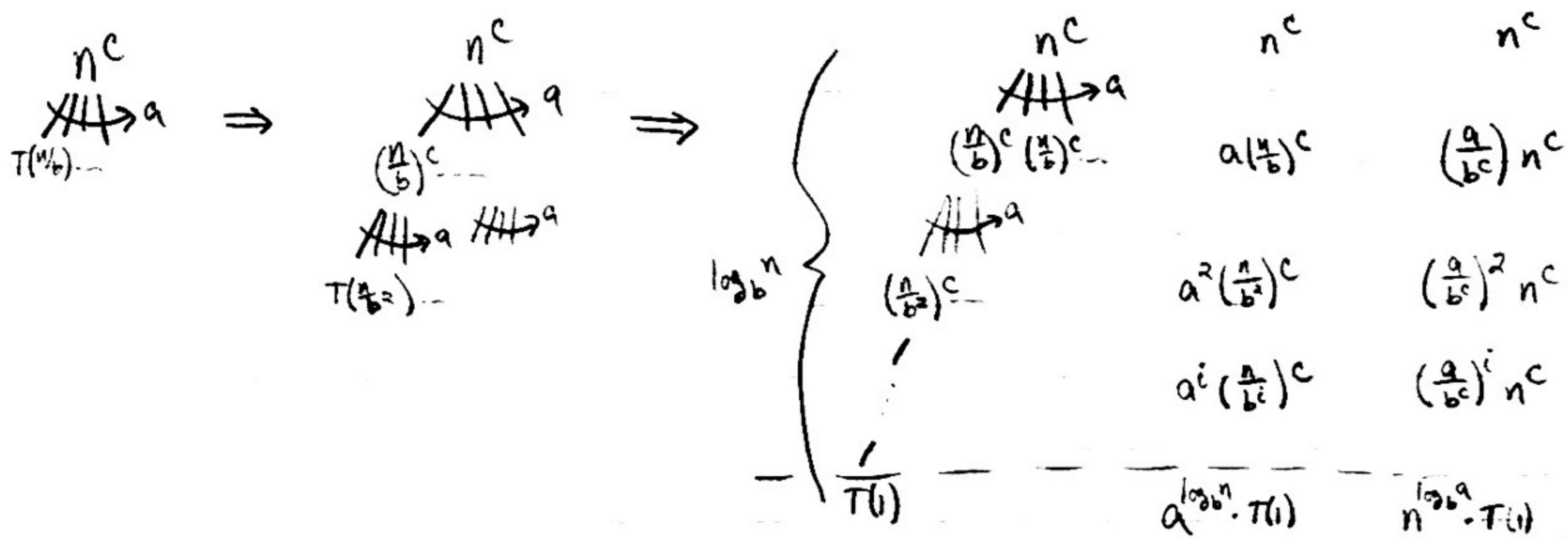
| | | |
|---|---|---|
| **Case 1:** | $c < \log_b a$ | $T(n) = \Theta(n^{\log_b a})$ |
| **Case 2:** | $c = \log_b a$ | $T(n) = \Theta(n^c \log n) = \Theta(n^{\log_b a} \log n)$ |
| **Case 3:** | $c > \log_b a$ | $T(n) = \Theta(n^c)$ |

- MergeSort $T(n) = 2T(n/2) + \Theta(n)$; a=2 b=2 c=1
  case 2 ; $T(n) = \Theta(n\log n)$
- Strassen's $T(n) = 7T(n/2) + \Theta(n^2)$ ; a=7 b=2 c=2
  case 1, $T(n) = \Theta(n^{\log_2(7)})$
- Binary Search $T(n)=T(n/2) + \Theta(1)$; a=1 b=2 c=0
  case 2, $T(n) = \Theta(\log n)$

# Master Theorem – why 3 cases

$T(n) = aT(n/b) + n^c$ (for simplicity, eliminate $\Theta$)

Recursion tree:




So, total is $n^c \sum_{i=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^i + \Theta(n^{\log_b a})$

- that sum is geometric progression with base $R = a/b^c$
- it comes down to R being <1 , =1 , >1 . So three

# Master Theorem – why 3 cases

Case 1 $\quad c < \log_b a \iff \frac{a}{b^c} > 1$ $\quad$ – work increases geometrically

$$\text{sum} = n^c \sum_{i=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^i + \Theta\left(n^{\log_b a}\right)$$

$$= n^c \frac{\left(\frac{a}{b^c}\right)^{\log_b n} - 1}{\left(\frac{a}{b^c}\right) - 1} + \Theta\left(n^{\log_b a}\right)$$

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

$$= \Theta\left(n^c \frac{n^{\log_b a}}{n^c}\right) + \Theta\left(n^{\log_b a}\right)$$

$$= \Theta\left(n^{\log_b a}\right)$$

$(b^c)^{\log_b n} = b^{c \cdot \log_b n} = (b^{\log_b n})^c = n^c$

∴ work at each level increases geometrically;

constant fraction of work is in leaves...

# Master Theorem – why 3 cases

Case 2 $\quad c = \log_b a \iff a/b^c = 1$ — work constant at each level

$$\text{sum} = n^c \sum_{i=0}^{\log_b n - 1} (1)^i + \Theta(n^{\log_b a}) = n^c \log_b n + \Theta(n^{\log_b a}) = \Theta(n^c \log_b n)$$

$\therefore$ work at each level is $n^c$ ($= n^{\log_b a}$); $\log_b n$ levels;

answer is $\Theta(n^c \log_b n)$

# Master Theorem – why 3 cases

Case 3 $\quad c > \log_b a \iff \frac{a}{b^c} < 1$ $\quad$ – work decreases geometrically

$$\text{sum} = n^c \sum_{i=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^i + \Theta\left(n^{\log_b a}\right)$$

$$= n^c \, \Theta(1) + \Theta\left(n^{\log_b a}\right)$$

$$= \Theta(n^c) + \Theta\left(n^{\log_b a}\right)$$

$$= \Theta(n^c)$$

Note:

① $\sum_{i=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^i \geq \left(\frac{a}{b^c}\right)^0 = 1$

$\Rightarrow \sum_{i=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^i = \Omega(1)$

② $\sum_{i=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^i < \sum_{i=0}^{\infty} \left(\frac{a}{b^c}\right)^i$

$= \frac{1}{1 - a/b^c}$ (constant)

$\Rightarrow \sum_{i=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^i = O(1)$

$\therefore \sum_{i=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^i = \Theta(1)$

$\therefore$ work at each level decreases geometrically;

constant fraction of work is at root...

# Master Theorem

- general case T(n) = aT(n/b) + f(n)
- CASE 1 :

$$f(n) = O(n^{\log_b a - \epsilon}) \Rightarrow T(n) = \Theta(n^{\log_b a})$$

- CASE 2:

$$f(n) = \Theta(n^{\log_b a} \log^k n) \Rightarrow T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$$

- CASE 3:

$$f(n) = \Omega(n^{\log_b a + \epsilon}); \frac{af(n/b)}{f(n)} < 1 - \epsilon \Rightarrow T(n) = \Theta(f(n))$$

# Master Theorem Example

- recurrence: $T(n)=4T(n/2) + \Theta(n^2 \log n)$
- Master Theorem: $a=4$; $b=2$; $f(n)= n^2 \log n$
  - $f(n) / n^{\log_b a} = f(n)/n^2 = \log n$, so case 2 with $k=1$
- solution $T(n)=\Theta(n^2 \log^2 n)$

# Master Theorem Example

- $T(n)=4T(n/2) + \Theta(n^3)$
- Master Theorem: $a=4$; $b=2$; $f(n)=n^3$
  - $f(n) / n^{\log_b a} = f(n)/n^2 = n$, so case 3
  - check case 3 condition:
    - $4f(n/2)/f(n) = 4(n/2)^3/n^3 = 1/2 < 1-\varepsilon$
- solution $T(n) = \Theta(n^3)$

# NON-Master Theorem Example

- $T(n) = 4T(n/2) + n^2/\log n$ ; $f(n) = n^2/\log n$
- $f(n) / n^{\log_b a} = f(n)/n^2 = 1/\log n$
  - case1:
  - case2:
  - case3:
- no case applies - cant use Master Theorem
- use iteration method for guess, and substitution for a proof
  - see attached pdf