Recurrences: Methods and Examples

CSE 3318 – Algorithms and Data Structures
Alexandra Stefan

University of Texas at Arlington

Background

- Solving Summations
  - Needed for the Tree Method
- Math substitution
  - Needed for Methods: Tree and Substitution(induction)
  - E.g. If $T(n) = 3T(n/8) + 4n^{2.5}\lg n$,

    $T(n/8) =$ ………………………………

    $T(n-1) =$ ………………………………
- Theory on trees
  - Given tree height & branching factor, compute:

    nodes per level

    total nodes in tree
- Logarithms
  - Needed for the Tree Method

- Notation: TC = Time Complexity (cost may also be used instead of time complexity)
- We will use different methods than what was done for solving recurrences in CSE 2315, but one may still benefit from reviewing that material.

Recurrences

- Recursive algorithms
  - It may not be clear what the complexity is, by just looking at the algorithm.

  - To find their complexity, we need to:
    - Express the TC of the algorithm as a recurrence formula. E.g.: f(n) = n + f(n-1)
    - Find the complexity of the recurrence:
      - Expand it to a summation with no recursive term.
      - Find a concise expression (or upper bound), E(n), for the summation.
      - Find Θ,ideally, or O (big-Oh) for E(n).

- Recurrence formulas may be encountered in other situations:
  - Compute the number of nodes in certain trees.
  - Express the complexity of non-recursive algorithms (e.g. selection sort).

Solving Recurrences Methods

- The Master Theorem
- The Recursion-Tree Method
  - Useful for guessing the bound.

- The Induction Method – not covered
  - Guess the bound, use induction to prove it.
  - Note that the book calls this the substitution method, but I prefer to call it the induction method

Recurrence - Recursion Tree Relationship

$T(1) = c$

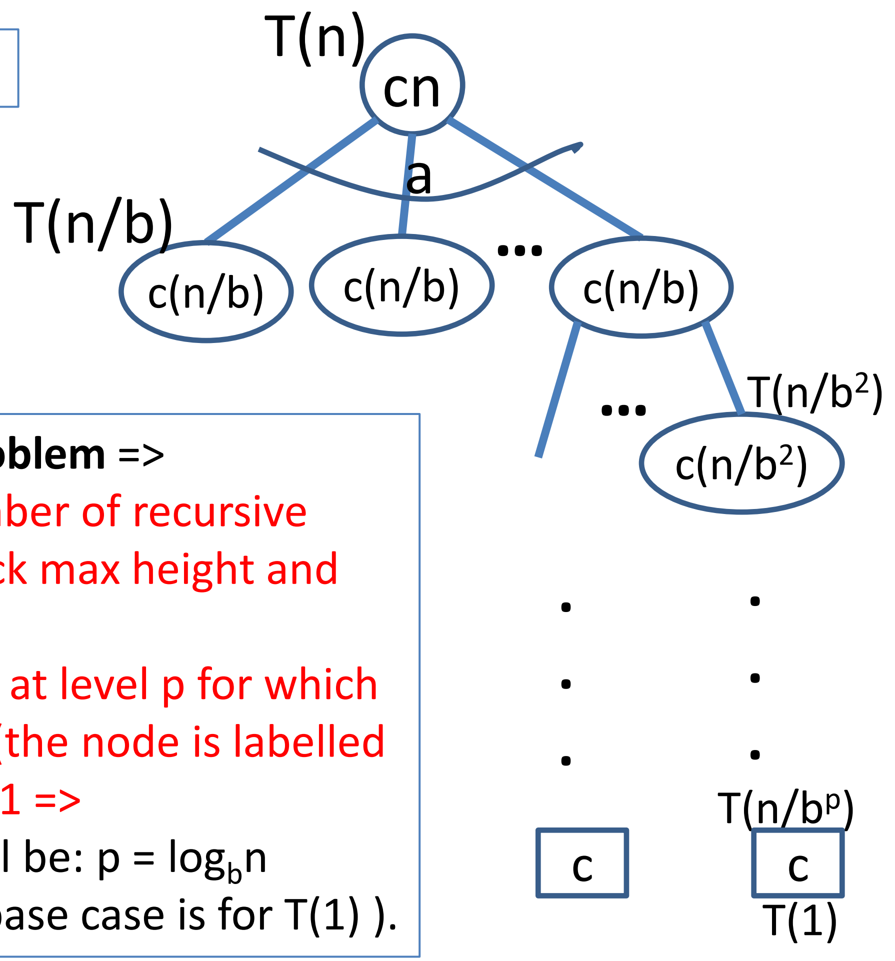
Problem size

The local TC at the node

$T(n) = a*T(n/b) + cn$

Number of subproblems =>
Number of children of a node in the recursion tree. =>
Affects the number of nodes per level. At level i there will be a^i nodes.
Affects the level TC.

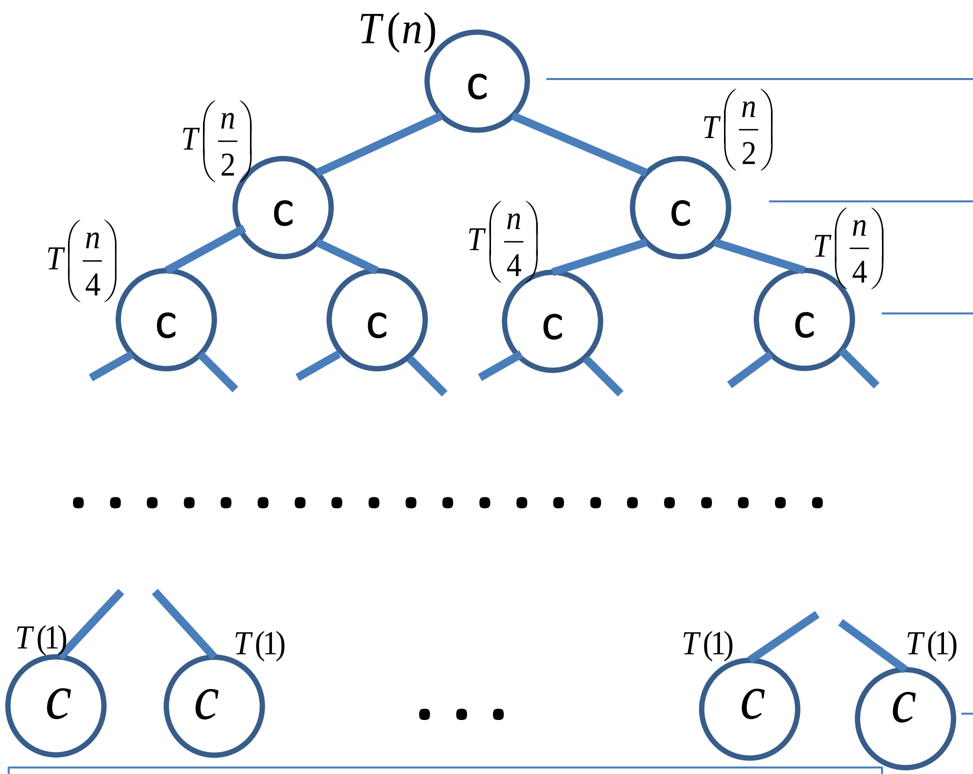
Size of a subproblem =>
Affects the number of recursive calls (frame stack max height and tree height)
Recursion stops at level p for which the pb size is 1 (the node is labelled T(1)) => $n/b^p = 1$ =>
Last level, p, will be: $p = \log_b n$
(assuming the base case is for T(1)).



TC = time complexity

Recursion Tree for: $T(n) = 2T(n/2)+c$

Base case: $T(1) = c$



Level	Arg/ pb size	TC of 1 node	Nodes per level	Level TC
0	n	c	1	c
1	n/2	c	2	2c
2	n/4	c	4	4c
…				
i	$n/2^i$	c	2^i	$2^i c$
…				
p=lgn	1 ($=n/2^p$)	c	2^p (=n)	$2^k c$

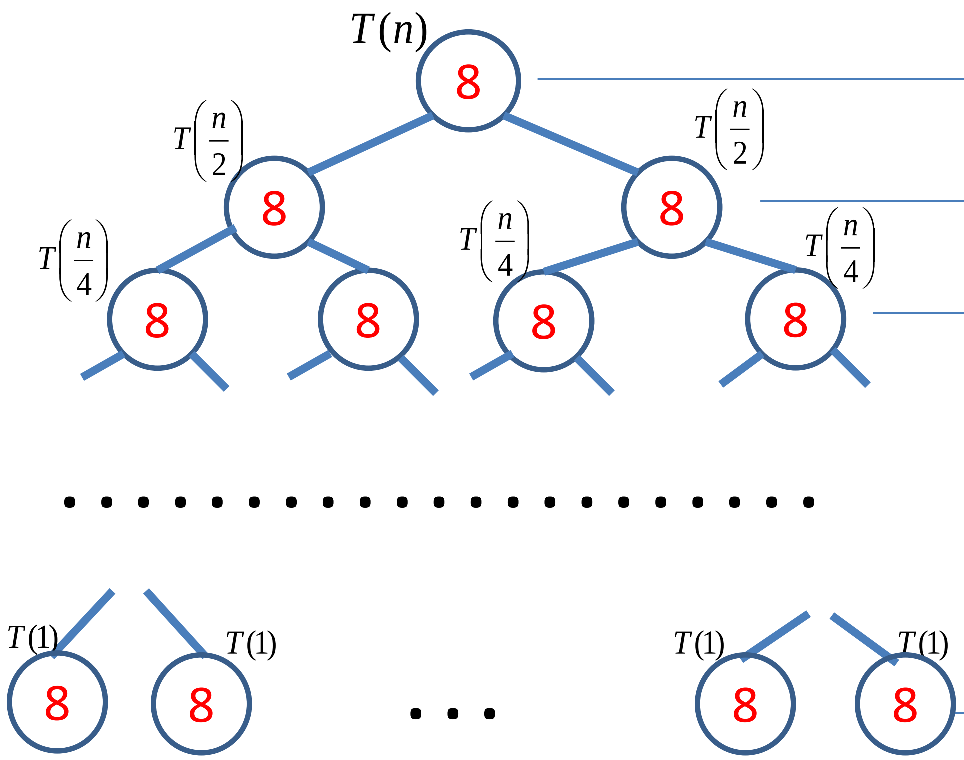
Stop at level p, when the subtree is T(1).
=> The problem size is 1, but the general formula for the problem size, at level p is:
$n/2^p \Rightarrow n/2^p = 1 \Rightarrow p = \lg n$

Tree TC $= c(1+2+2^2+2^3+\ldots+2^i+\ldots+2^p) = c2^{p+1}/(2-1)$
$= 2c2^p = 2cn = \Theta(n)$

Recursion Tree for: $T(n) = 2T(n/2)+8$

If specific value is given instead of c, use that. Here c=8.

Base case: $T(1) = 8$



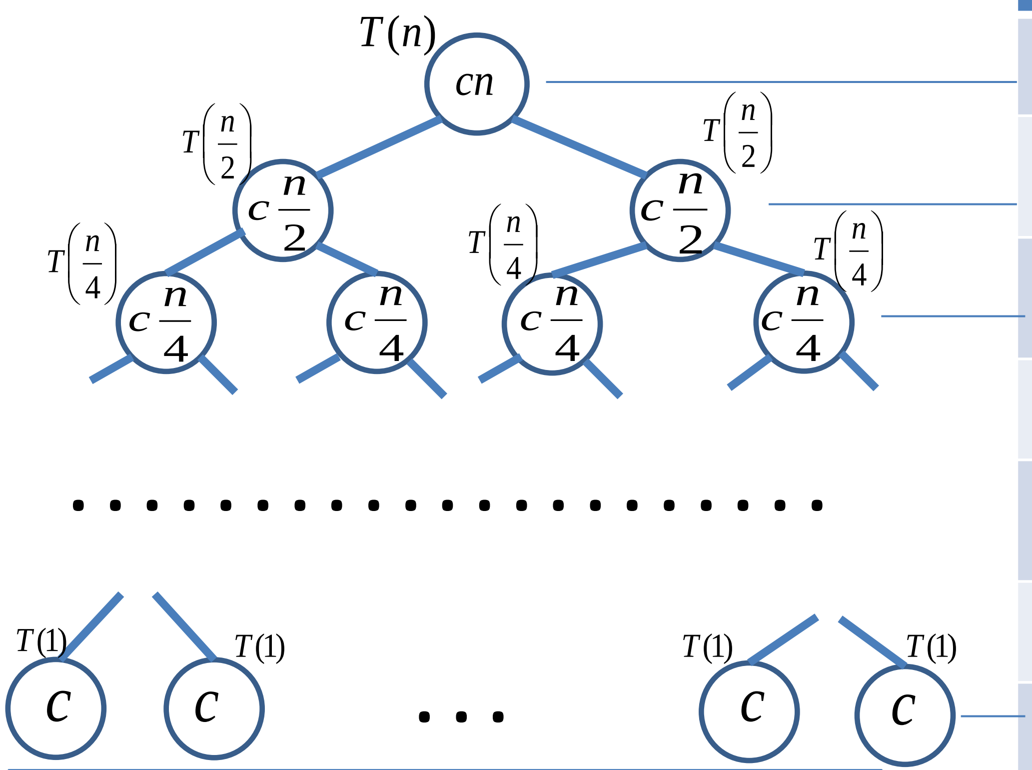
Level	Arg/ pb size	TC of 1 node	Nodes per level	Level TC
0	n	8	1	**8**
1	n/2	8	2	2***8**
2	n/4	8	4	4*8
…				
i	$n/2^i$	8	2^i	2^i*8
…				
k=lgn	1 ($=n/2^k$)	8	2^k (=n)	2^k*8

Stop at level p, when the subtree is T(1).
=> The problem size is 1, but the general formula for the problem size, at level p is:
$n/2^p \Rightarrow n/2^p = 1 \Rightarrow 2^p = n \Rightarrow p = \lg n$

Tree TC = $c(1+2+2^2+2^3+\ldots+2^i+\ldots+2^p) = 8*2^{p+1}/(2-1)$
$= 2*8*2^p = 16n = \Theta(n)$

Recursion Tree for: T(n) = 2T(n/2)+cn

Base case: T(1) = c



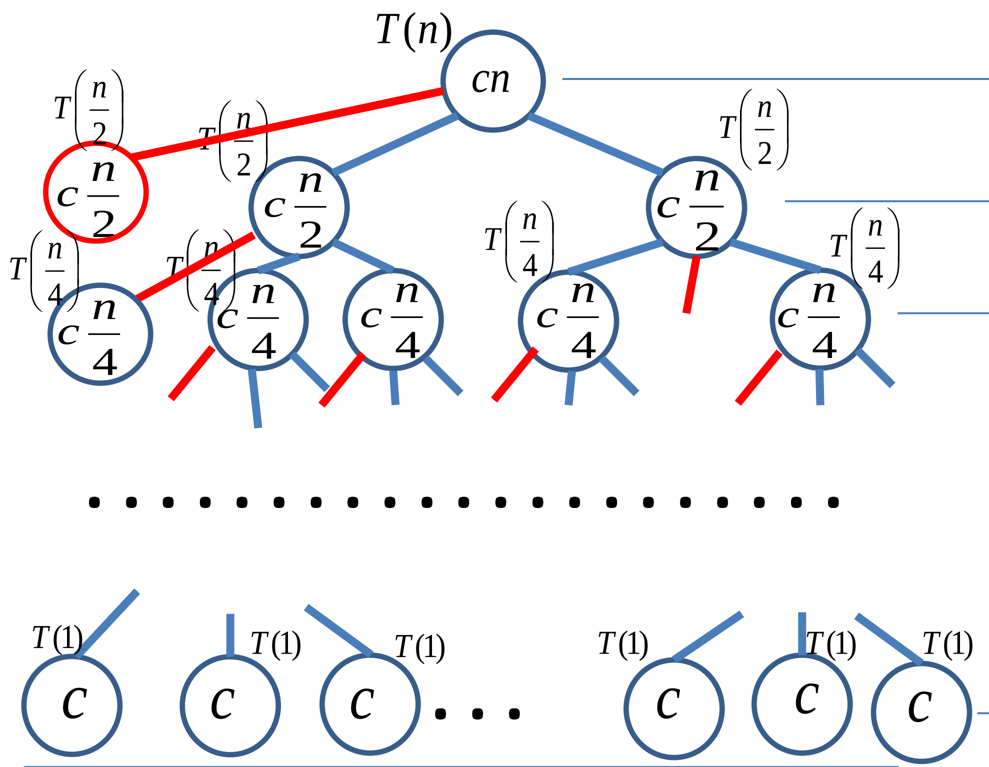
Level	Arg/ pb size	TC of 1 node	Nodes per level	Level TC
0	n	c*n	1	c*n
1	n/2	c*n/2	2	2*c*n/2 =c*n
2	n/4	c*n/4	4	4*c*n/4 =c*n
…				
i	$n/2^i$	$c*n/2^i$	2^i	$2^i*c*n/2^i$ =c*n
…				
p=lgn	1 $(=n/2^p)$	$c=c*1=c*n/2^p$	2^p (=n)	$2^p*c*n/2^p$ =c*n

Stop at level p, when the subtree is T(1).
=> The problem size is 1, but the general formula for the problem size, at level p is:
$n/2^p$ => $n/2^p=1$ => $2^p=n$ => p= lgn

Tree TC $= cn(p+1) = cn(1+lgn)$
$= cnlgn + cn = \theta(nlgn)$

Recursion Tree for T(n) = 3T(n/2)+c

Base case: T(1) = c

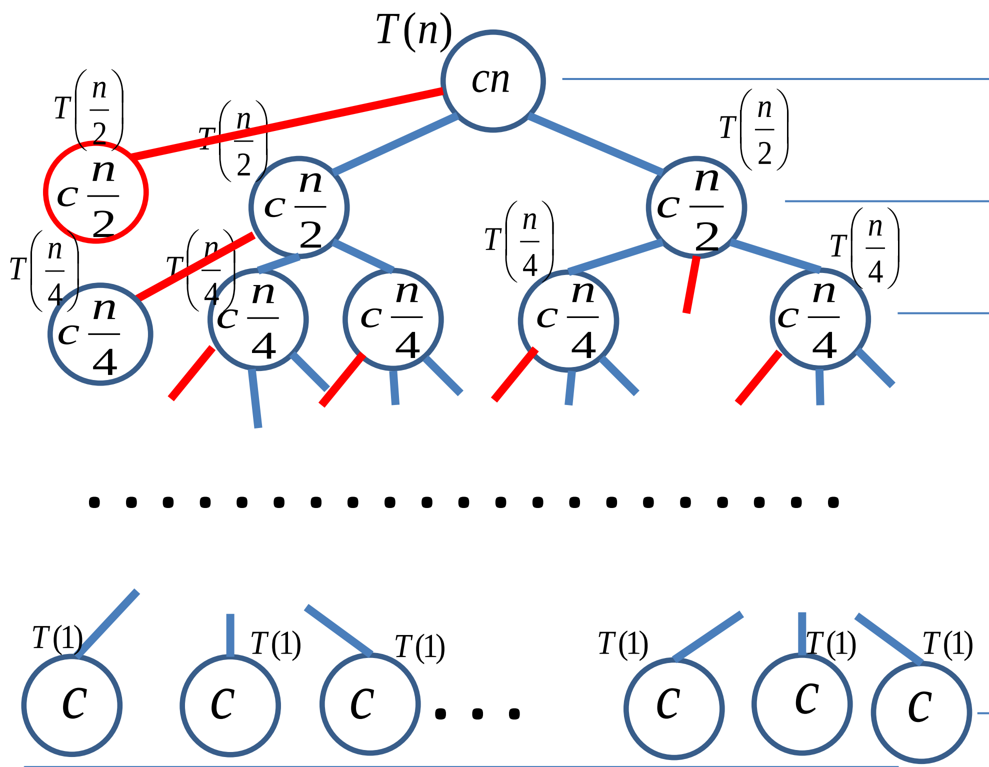


Level	Arg/ pb size	TC of 1 node	Nodes per level	Level TC
0	n	c*n	1	c
1	n/2	c*n/2	3	3*c =(3)*c
2	n/4	c*n/4	9	$(3)^2$*c
…				
i	$n/2^i$	$c*n/2^i$	3^i	$(3)^i$*c
…				
p=lgn	1 (=$n/2^p$)	c=c*1= $c*n/2^p$	3^p (≠n)	$(3)^p$*c

Stop at level p, when the subtree is T(1).
=> The problem size is 1, but the general formula for the problem size, at level p is:
$n/2^p$=> $n/2^p=1$ => $2^p=n$ => p = lgn

Recursion Tree for T(n) = 3T(n/2)+cn

Base case: T(1) = c



Level	Arg/ pb size	TC of 1 node	Nodes per level	Level TC
0	n	c*n	1	c*n
1	n/2	c*n/2	**3**	3*c*n/2 =**(3/2)***c*n
2	n/4	c*n/4	**9**	**$(3/2)^2$***c*n
…				
i	$n/2^i$	$c*n/2^i$	**3^i**	**$(3/2)^i$***c*n
…				
p=lgn	1 (=$n/2^p$)	c=c*1= $c*n/2^p$	**3^p** (≠n)	**$(3/2)^p$***c*n

Stop at level p, when the subtree is T(1).
=> The problem size is 1, but the general formula for the problem size, at level p is:
$n/2^p$=> $n/2^p$=1 => 2^p=n => p = lgn

Total Tree TC for T(n) = 3T(n/2)+cn

Closed form

$$T(n) = cn + (3/2)cn + (3/2)^2 cn + \ldots (3/2)^i cn + \ldots (3/2)^{\lg n} cn =$$

$$= cn * [1 + (3/2) + (3/2)^2 + \ldots + (3/2)^{\lg n}] = cn \sum_{i=0}^{\lg n} (3/2)^i =$$

$$= cn * \boxed{\frac{(3/2)^{\lg n+1} - 1}{(3/2) - 1}} = 2cn[(3/2) * (3/2)^{\lg n} - 1] = 3cn * (3/2)^{\lg n} - 2cn$$

$$use : c^{\lg n} = n^{\lg c} \;=>\; (3/2)^{\lg n} = n^{\lg(3/2)} = n^{\lg 3 - \lg 2} = n^{\lg 3 - 1} \;=>$$

$$= 3cn * n^{\lg 3 - 1} - 2cn = 3cn^{1 + \lg 3 - 1} - 2cn = 3cn^{\lg 3} - 2cn = \Theta(n^{\lg 3})$$

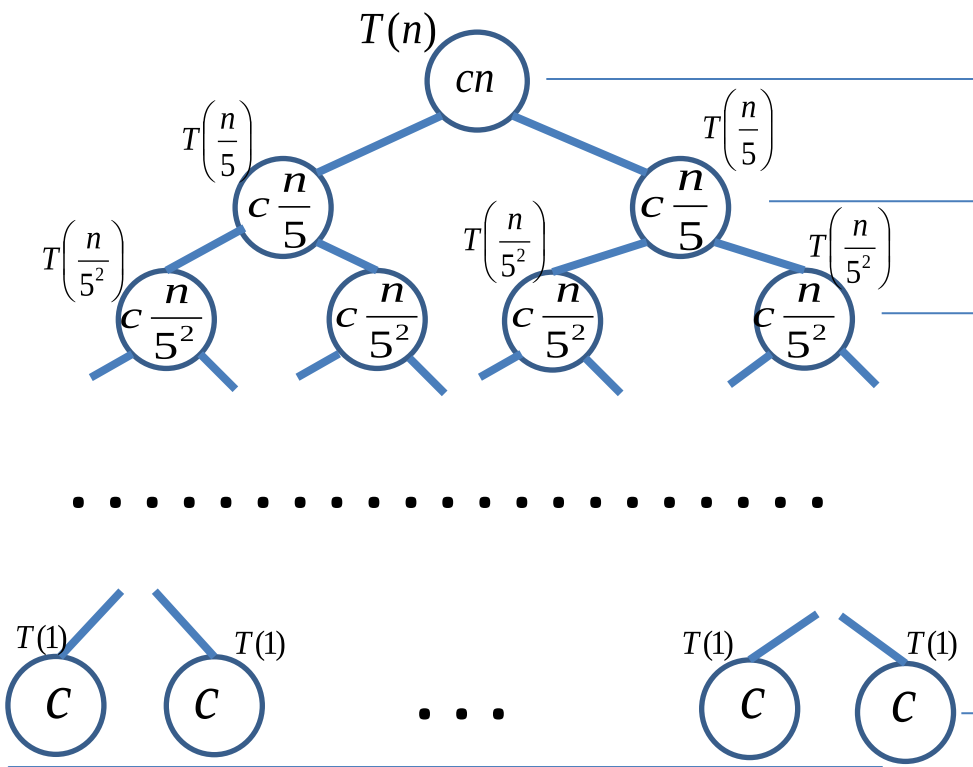
Explanation: since we need Θ, we can eliminate the constants and non-dominant terms earlier (after the closed form expression):

$$\ldots = cn * \frac{(3/2)^{\lg n+1} - 1}{(3/2) - 1} = \Theta(n * (3/2) * (3/2)^{\lg n}) = \Theta(n * (3/2)^{\lg n})$$

$$use\!: c \lg n \;= n^{\lg c} \;=>\; (3/2)^{\lg n} = n^{\lg(3/2)} = n^{\lg 3 - \lg 2} = n^{\lg 3 - 1} \;=>$$

$$= \Theta(n * n^{\lg 3 - 1}) = \Theta(n^{\lg 3})$$

Recursion Tree for: T(n) = 2T(n/5)+cn



Level	Arg/ pb size	TC of 1 node	Nodes per level	Level TC
0	n	c*n	1	c*n
1	n/**5**	c*n/**5**	2	2*c*n/**5** =**(2/5)***cn
2	$n/5^2$	$c*n/5^2$	4	4*c*n/ $=(2/5)^i cn$
…				
i	$n/5^i$	$c*n/5^i$	2^i	$2^i*c*n/5^i$ $=(2/5)^i cn$
…				
p= $\log_5 n$	1 $(=n/5^p)$	c=c*1= $c*n/5^p$	2^p (=n)	$2^p*c*n/5^p$ $=(2/5)^p cn$

Stop at level p, when the subtree is T(1).
=> The problem size is 1, but the general formula for the problem size, at level p is:
$n/5^p$=> $n/5^p$= 1 => 5^p=n => p = $\log_5 n$

Tree TC
(derivation similar to TC for T(n) = 3T(n/2)+cn)

Total Tree TC for T(n) = 2T(n/5)+cn

$$T(n) = cn + (2/5)cn + (2/5)^2 cn + \ldots (2/5)^i cn + \ldots (2/5)^{\log_5 n} cn =$$

$$= cn * [1 + (2/5) + (2/5)^2 + \ldots + (2/5)^{\log_5 n}] =$$

$$= cn \sum_{i=0}^{\log_5 n} (2/5)^i \leq cn \sum_{i=0}^{\infty} (2/5)^i =$$

$$= cn * \frac{1}{1-(2/5)} = (5/3)cn = O(n)$$

$$Also$$

$$T(n) = cn + \ldots \Rightarrow T(n) \geq cn \Rightarrow T(n) = \Omega(n)$$

$$\Rightarrow T(n) = \Theta(n)$$

Code => Recurrence

In the recursive case of the recurrence formula capture the number of times the recursive call ACTUALLY EXECUTES as you run the instructions in the function.

```
int foo(int N){
 int a,b,c;
 if(N<=3) return 1500; // Note N<=3
 a = 2*foo(N-1);
// a = foo(N-1)+foo(N-1);
 printf("A");
 b = foo(N/2);
 c = foo(N-1);
 return a+b+c;
}
```

Base case: T(__) = ___________

Recursive case: T(__) = _______________________

T(N) gives us the Time Complexity for foo(N). We need to solve it (find the closed form)

Code => Recurrence => Θ

In the recursive case of the recurrence formula capture the number of times the recursive call ACTUALLY EXECUTES as you run the instructions in the function.

```
void bar(int N){
 int i,k,t;
 if(N<=1) return;
 bar(N/5);
 for(i=1;i<=5;i++){
   bar(N/5);
 }
 for(i=1;i<=N;i++){
   for(k=N;k>=1;k--)
     for(t=2;t<2*N;t=t+2)
       printf("B");
 }
 bar(N/5);
}
```

T(N) = ……………………………………

Solve T(N)

Compare

```
void foo1(int N){
     if (N <= 1) return;
     for(int i=1; i<=N; i++){
         foo1(N-1);
     }
}

T(0)=T(1) = c
T(N) = N*T(N-1) + cN
```

```
void foo2(int N){
     if (N <= 5) return;
     for(int i=1; i<=N; i++){
         printf("A");
     }
     foo2(N-1); //outside of the loop
}

T(N) = c for all 0≤N≤5   (BaseCase(s))
T(N) = T(N-1) + cN    (Recursive Case)
```

```
int foo3(int N){
     if (N <= 20) return 500;
     for(int i=1; i<=N; i++){
         return foo3(N-1);
// No loop. Returns after the first iteration.
     }
}
```

T(N) = c for all 0≤N≤20   Do not confuse what the function returns with its time complexity. For the base case, c is not 500. At most, c is 2 (from the 2 instructions: one comparison, *N<=20*, and one return, *return 500*)

T(N) = T(N-1) + **c**

In the recursive case of the recurrence formula captures the number of times the recursive call **ACTUALLY EXECUTES** as you run the instructions in the function. E.g. pay attention to 2*foo(N/3) vs foo(N/3) + foo(N/3)

Code =>recurrence

```
int search(int A[], int L, int R, int v){
   int m = (L+R)/2;
   if (L > R) return -1;
   if (v == A[m]) return m;
   if (L == R) return -1;
   if (v < A[m]) return search(A,L,m-1,v);
   else          return search(A,m+1,R,v);
}
(Use:  N = R-L+1)
Here, for the same value of N, the behavior depends also on data in A and val.
Best case T(N) = c => search is Θ(1) in best case
Worst case: T(N) = T(N/2) + c  => T(N) = Θ(lg(N)) => search is Θ(lg(N))in worst case
⇒ We will report in general: search is O(lg(N))
```

Code => recurrence

```
int weird(int A[], int N){
   if (N<=4) return 100;
   if (N%5==0) return weird(A,N/5);
   else        return weird(A,N-4)+weird(A, N-4);
}
```

Here, the behavior depends on N so we can explicitly capture that in the recurrence formulas:

Base case(s): $T(N) = c$ for all $0 \le N \le 4$ (BC)

Recursive case(s):

$T(N) = T(N/5)+c$ for all $N>4$ that are multiples of 5 (RC1)

$T(N) = 2*T(N-4) + c$ for all other N (RC2)

For any N, in order to solve, we need to go through a mix of the 2 recursive cases => cannot easily solve. => try to find lower and upper bounds.

Note that RC1 has the best behavior: only one recurrence and smallest subproblem size (i.e. N/5) => use this for a lower bound =>

$T_{lower}(N) = T(N/5)+c = \Theta(\log_5 N)$, (and $T(N) \ge T_{lower}(N)$) => $\mathbf{T(N) = \Omega(\log_5 N)}$

Note that RC2 has the worst behavior: 2 recurrences and both of larger subproblem size (i.e. N-4) => use this for an upper bound =>

$T_{upper}(N) = 2*T(N-4)+c = \Theta(2^{N/4})$, (and $T(N) \le T_{upper}(N) = \Theta(2^{N/4})$) => $\mathbf{T(N) = O(2^{N/4})}$

We have **Ω and O** for T(N), but we cannot compute **Θ** for it.

Recurrence => Code Answers

- Give a piece of code/pseudocode for which the time complexity recursive formula is:
  - T(1) = c and
  - T(N) = N*T(N/2) + cN

```
void foo(int N){
     if (N <= 1) return;
     for(int i=1; i<=N; i++)
         foo(N/2);
}
```

Recurrences:
Recursion-Tree Method

1. Build the tree & fill-out the table
2. Compute TC per level
3. Compute number of levels (find last level as a function of N)
4. Compute total over levels.
    * Find closed form of that summation.

Example 1 : Solve $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$

Example 2 : Solve $T(n) = T(n/3) + T(2n/3) + O(n)$

Recurrence - Recursion Tree Relationship

$T(1) = c$

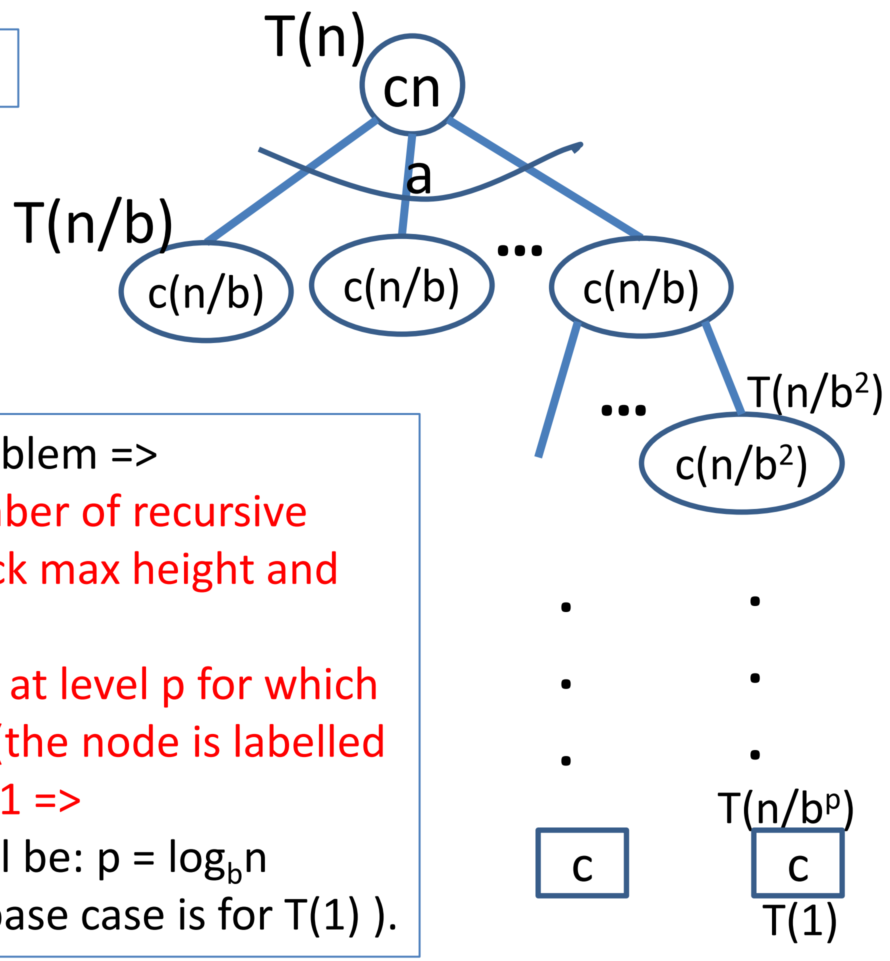
Problem size

The local TC at the node

$$T(n) = a*T(n/b) + cn$$

Number of subproblems => Number of children of a node in the recursion tree. => Affects the number of nodes per level. At level i there will be a^i nodes.
Affects the level TC.

Size of a subproblem =>
Affects the number of recursive calls (frame stack max height and tree height)
Recursion stops at level p for which the pb size is 1 (the node is labelled T(1)) => $n/b^p = 1$ =>
Last level, p, will be: $p = \log_b n$ (assuming the base case is for T(1)).



$T(n) = 7T(n/5)+cn^3$, If n is not a multiple of 5, use round down for n/5

$T(1) = c$, $T(0) = c$

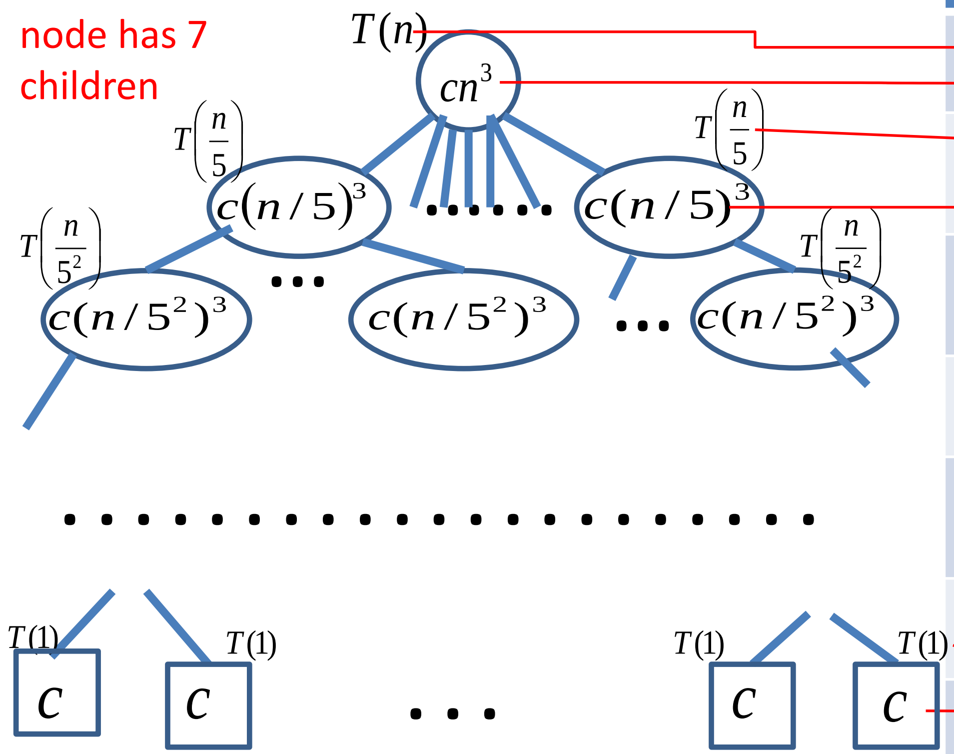
Level	Arg/ pb size	TC of 1 node	Nodes per level	Level TC
0				
1				
2				
…				
i				
…				
p=				

Work it out by hand in class.
Draw tree, fill out table.

$T(n) = 7T(n/5)+cn^3$, If n is not a multiple of 5, use round down for n/5

$T(1) = c$, $T(0) = c$

Each internal node has 7 children



Level	Arg/ pb size	TC of 1 node	Nodes per level	Level TC
0	n	cn^3	1	$c*n^3$
1	n/5	$c(n/5)^3$	7	$7*c*(n/5)^3$ $=cn^3(7/5^3)$
2	$n/5^2$	$c(n/5^2)^3$	7^2	$7^2*c*(n/5^2)^3$ $=cn^3(7/5^3)^2$
…				
i	$n/5^i$	$c(n/5^i)^3$	7^i	$7^i*c*(n/5^i)^3$ $=cn^3(7/5^3)^i$
…				
p= $\log_5 n$	1 $(=n/5^p)$	$c=c*1=$ $c(n/5^p)^3$	7^p	$7^p*c*(n/5^p)^3$ $=cn^3(7/5^3)^p$

Stop at level p, when the subtree is T(1). => The problem size is 1, but the general formula for the problem size, at level p is: $n/5^p$=> $n/5^p= 1$ => $p = \log_5 n$

Where we used: $7^i(\frac{n}{5^i})^3 = 7^i n^3 (\frac{1}{5^i})^3 = 7^i n^3 (\frac{1}{5^3})^i = n^3 (\frac{7}{5^3})^i$

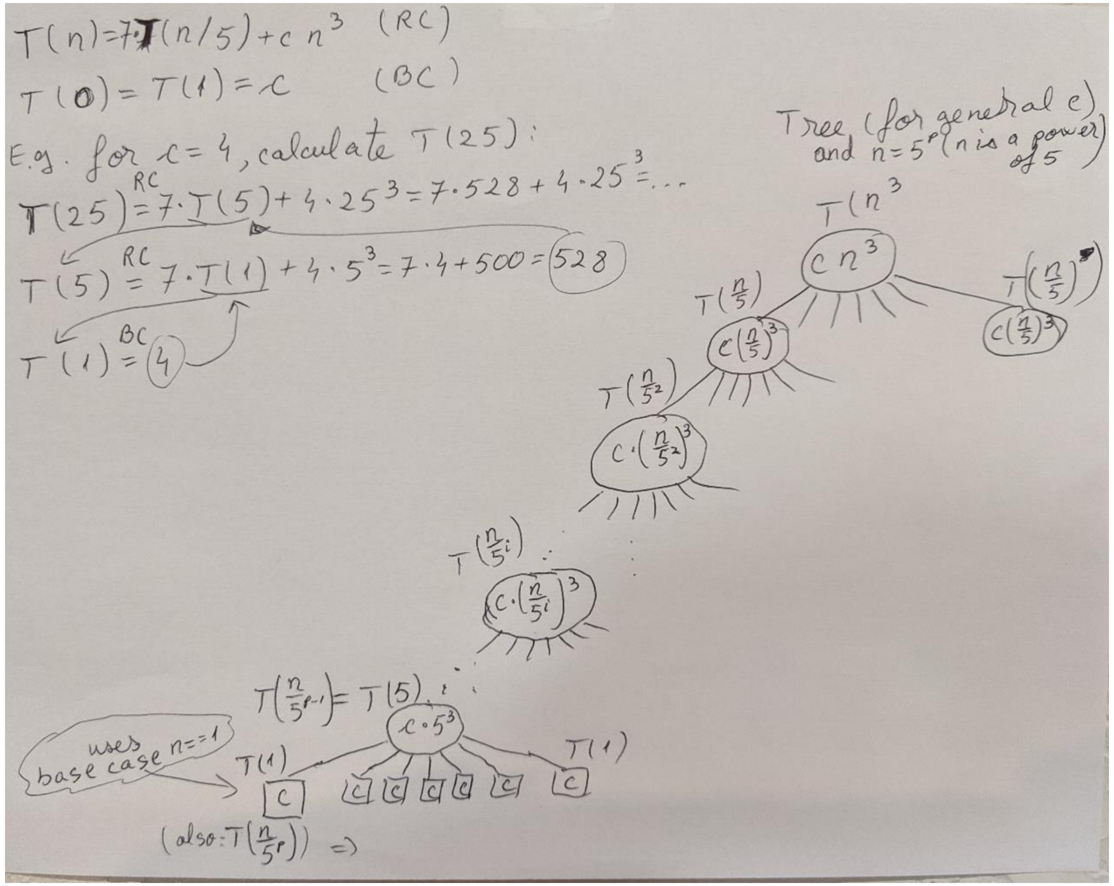
Tree TC: $T(n) = \sum_{i=0}^{\log_5 n} cn^3 (\frac{7}{5^3})^i = cn^3 \sum_{i=0}^{\log_5 n} (\frac{7}{5^3})^i =$

$cn^3 \frac{1-(7/125)^{1+\log_5 n}}{1-(7/125)} < cn^3 \frac{1}{1-7/125} = \Theta(n^3) \Rightarrow T(n) = O(n^3)$

But $T(n) = \Omega(n^3) \Rightarrow T(n) = \Theta(n^3)$

$T(n) = 7T(n/5)+cn^3$, If n is not a multiple of 5, use round down for n/5

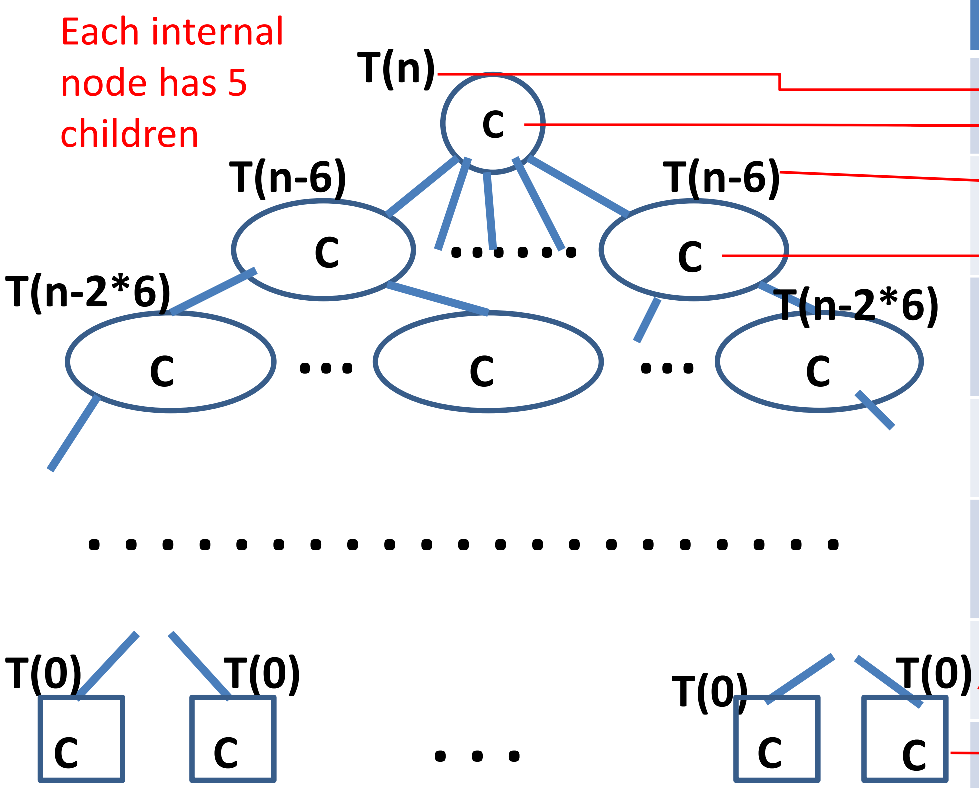
$T(1) = c,\ T(0) = c$



$T(n) = 5T(n-6)+c$

$T(n) = c$ for all $0 \le n \le 5$ (i.e. $T(0)=T(1)=T(2)=T(3)=T(4)=T(5)=c$)

Assume n is a multiple of 6



Level	Arg/ pb size	TC of 1 node	Nodes per level	Level TC
0	n	c	1	c
1	n-6	c	5	5*c
2	n-2*6	c	5^2	5^2*c
…				
i	n-6i	c	5^i	5^i*c
…				
p= n/6	0 (=n-6p)	c	5^p	5^p*c

Stop at level p, when the subtree is T(0).
=> The problem size is 0, but the general formula for the problem size, at level p is:
n-6p=> n-6p= 0 => p = n/6

$T(n) = c(1+5+5^2 + 5^3+ \ldots +5^i+\ldots+5^p)=$
$c(5^{(p+1)}-1)/(5-1)=\Theta(5^p)= \Theta(5^{n/6})$

- Rounding up or down the size of subproblems does not affect Theta. All four recurrences below have the same Theta:

$$T(N) = 2T\left(\frac{N}{3}\right) + c\ ,$$

$$T(N) = 2T\left(\left\lfloor\frac{N}{3}\right\rfloor\right) + c$$

$$T(N) = 2T\left(\left\lceil\frac{N}{3}\right\rceil\right) + c\ ,$$

$$T(N) = T\left(\left\lfloor\frac{N}{3}\right\rfloor\right) + T\left(\left\lceil\frac{N}{3}\right\rceil\right) + c$$

- See more solved examples later in the presentation. Look for page with title:

More practice/ Special cases

Tree Method for lower/upper bounds

$$T(n) = T(n/3) + T(2n/3) + O(n)$$

- Draw the tree, notice the shape, see length of shortest and longest paths.
- Notice that:
  - as long as the levels are full (all nodes have 2 children) the level TC is cn (the sum of TC of the children equals the parent: (1/3)*p_TC+(2/3) *p_TC)
  - $\Rightarrow$ Total TC for those: $cn*\log_3 n = \Theta(n\lg n)$
  - The number of incomplete levels should also be a multiple of lgn and the TC for each of those levels will be less than cn
  - => Guess that T(n) = O(nlgn)
- Use the substitution method to show T(n) = O(nlgn)
- If the recurrence was given with Θ instead of O, we could have shown T(n) = Θ(nlgn)
  - with O, de only know that: T(n) **≤** T(n/3)+T(2n/3)+**cn**
  - The local TC could even be constant: T(n) = T(n/3)+T(2n/3) + c
- Exercise: Solve
  - $T_1(n) = 2T_1(n/3) + cn$ (Why can we use cn instead of Θ(n) in $T_1(n) = 2T_1(n/3) + cn$?)
  - $T_2(n) = 2T_2(2n/3) + cn$ (useful: lg3 ≈1.59)
  - Use them to bound T(n). How does that compare to the analysis in this slide? (The bounds are looser).

Common Recurrences Review

1. *Halve* problem in <u>constant</u> time :

   $T(n) = T(n/2) + c$ $\Theta(\lg(n))$

2. *Halve* problem in <u>linear</u> time :

   $T(n) = T(n/2) + n$ $\Theta(n)$ (~2n)

3. Break (and put back together) the problem into *2 halves* in <u>constant</u> time:

   $T(n) = 2T(n/2) + c$ $\Theta(n)$ (~2n)

4. Break (and put back together) the problem into *2 halves* in <u>linear</u> time:

   $T(n) = 2T(n/2) + n$ $\Theta(n \lg(n))$

5. Reduce the problem size by 1 in <u>constant</u> time:

   $T(n) = T(n-1) + c$ $\Theta(n)$

6. Reduce the problem size by 1 in <u>linear</u> time:

   $T(n) = T(n-1) + n$ $\Theta(n^2)$

Master theorem

- We will use the Master Theorem from wikipedia as it covers more cases:

https://en.wikipedia.org/wiki/Master_theorem_(analysis_of_algorithms)

- Check the above webpage and the notes handwritten in class.
- Discussion:

On Wikipedia, below the inadmissible equations there is the justification pasted below.

However the cases given for the Master Theorem on Wikipedia, do not include any ε in the discussion. Where does that ε come from? Can you do math derivations that start from the formulation of the relevant case of the Theorem and result in the ε and the inequality shown above?

In the second inadmissible example above, the difference between $f(n)$ and $n^{\log_b a}$ can be expressed with the ratio $\frac{f(n)}{n^{\log_b a}} = \frac{n/\log n}{n^{\log_2 2}} = \frac{n}{n \log n} = \frac{1}{\log n}$. It is clear that $\frac{1}{\log n} < n^{\epsilon}$ for any constant $\epsilon > 0$. Therefore, the difference is not polynomial and the basic form of the Master Theorem does not apply. The extended form (case 2b) does apply, giving the solution $T(n) = \Theta(n \log \log n)$.

Recurrences:
Induction Method

1. Guess the solution
2. Use induction to prove it.
3. Check it at the boundaries (recursion base cases)

Example: Find upper bound for: $T(n) = 2T(\lfloor n/2 \rfloor) + n$

1. Guess that *T(n) = O(nlgn)* =>
2. Prove that *T(n) = O(nlgn)* using *T(n) <= cnlgn* (for some *c*)
    1. Assume it holds for all *m<n*, and prove it holds for *n*.
3. Assume base case (boundary): T(1) = 1.

   Pick c and n_0 s.t. it works for sufficient base cases and applying the inductive hypotheses.

Recurrences: Induction Method

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

2. Prove that T(n) = O(nlgn), using the definition:

find c and n_0 s.t. T(n) ≤ c*nlgn

(here: f(n) = T(n), g(n) = nlgn)

Show with induction: T(n) ≤ c*nlgn (for some *c>0*)

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \le 2 * c * \lfloor n/2 \rfloor * \lg(\lfloor n/2 \rfloor) + n \le$$

$$\le 2 * c * (n/2) * \lg(n/2) + n = cn\lg(n/2) + n =$$

$$= cn(\lg n - \lg 2) + n = cn(\lg n - 1) + n = cn\lg n - cn + n =$$

$$= cn\lg n + n(1-c)$$

$$want:$$

$$\le cn\lg n \Rightarrow$$

$$n(1-c) \le 0 \Rightarrow 1 - c \le 0 => c \ge 1$$

Pick c = 2 (the largest of both 1 and 2).
Pick n_0 = 2

3. Base case (boundary):
Assume T(1) = 1
Find n_0 s.t. the induction holds for all n≥ n_0.
n=1: 1=T(1) ≤ c*1*lg1 =c*0 =0
FALSE. => n_0 cannot be 1.

n=2: T(2) = 2*T(1) + 2 = 2+2=4
Want T(2) ≤ c*2lg2=2c, True for: c≥2

n=3: T(3)=2*T(1)+3=2+3=5
Want 5=T(3) ≤ c*3*lg3
True for: c≥2

Here we need 2 base cases for the induction: n=2, and n=3

Recurrences: Induction Method Various Issues

- Subtleties (stronger condition needed)
  - Solve: $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$ *with* $T(1) = 1$ *and* $T(0) = 1$
  - Use a stronger condition: off by a constant, subtract a constant
- Avoiding pitfalls
  - Wrong: In the above example, stop at T(n)≤cn+1 and conclude that T(n) =O(n)
  - See also book example of wrong proof for $T(n) = 2T(\lfloor n/2 \rfloor) + n$ is O(n)
- Making a good guess
  - Solve: $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$
  - Find a similar recursion
  - Use looser upper and lower bounds and gradually tighten them
- Changing variables
  - Recommended reading, not required (page 86)

Stronger Hypothesis for

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$$

*Show T(n) = O(n) using the definition: find c and n_0 s.t. T(n) ≤ c*n*

*(here: f(n) = T(n), g(n) = n). Use induction to show T(n) ≤ c*n*

Inductive step: assume it holds for all m<n, show for n:

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 \le c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 =$$
$$= c(\lfloor n/2 \rfloor + \lceil n/2 \rceil) + 1 = cn + 1$$

We're stuck. We CANNOT say that T(n) =O(n) at this point. We must prove the hypothesis exactly: T(n) ≤ cn   (not: T(n)≤cn+1).

Use a stronger hypothesis: prove that T(n) ≤cn-d, for some const d>0:

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 \le c\lfloor n/2 \rfloor - d + c\lceil n/2 \rceil - d + 1 =$$
$$= c(\lfloor n/2 \rfloor + \lceil n/2 \rceil) + 1 - 2d = cn - d + 1 - d$$

$want:$

$$\boxed{\begin{array}{l} \le cn - d \Rightarrow \\ 1 - d \le 0 => d \ge 1 \end{array}}$$

Extra material – Solve:

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$

- Use the tree method to make a guess for:

$$T(n) = 3T(n/4) + \Theta(n^2)$$

- Use the induction method for the original recurrence (with rounding down):

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$

More practice/ Special cases

Recurrences solved in following slides

Recurrences solved in following slides:

$T(n) = T(n-1)+c$

$T(n) = T(n-4)+c$

$T(n) = T(n-1)+cn$

$T(n) = T(n/2)+c$

$T(n) = T(n/2)+cn$

$T(n) = 2T(n/2)+c$

$T(n) = 2T(n/2)+8$

$T(n) = 2T(n/2)+cn$

$T(n) = 3T(n/2)+cn$

$T(n) = 3T(n/5)+cn$

Recurrences left as individual practice:

$T(n) = 7T(n/3)+cn$

$T(n) = 7T(n/3)+cn^3$

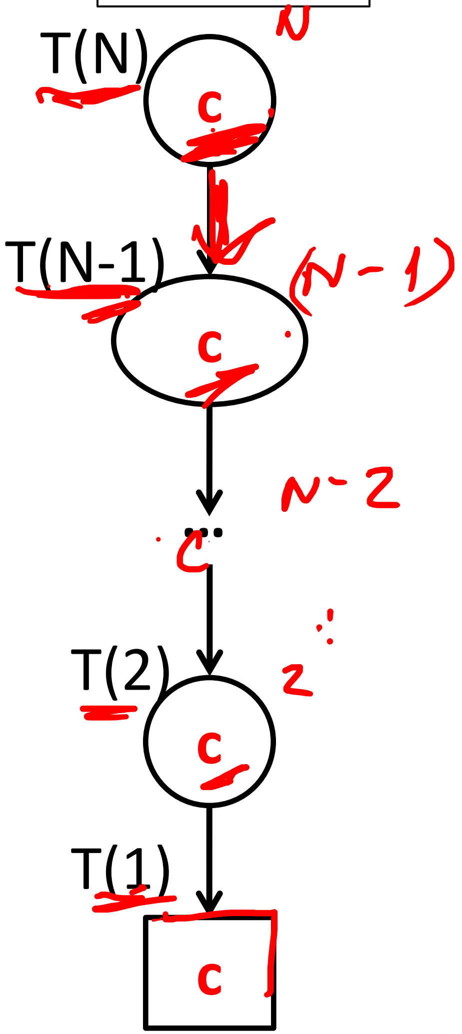
$T(n) = T(n/2)+n$

See also “recurrences practice” problems on the Exams page.

$$T(N) = T(N-1) + c$$

fact(N)

Time complexity tree:

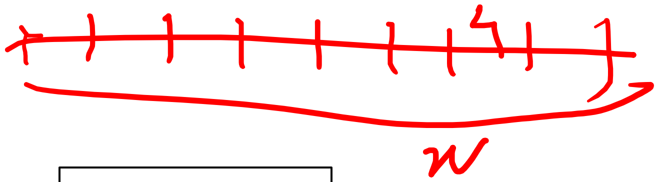



```
int fact(int N)
{
    if (N <= 1) return 1;
    return N*fact(N-1);
}
```

Time complexity of fact(N) ? T(N) = …

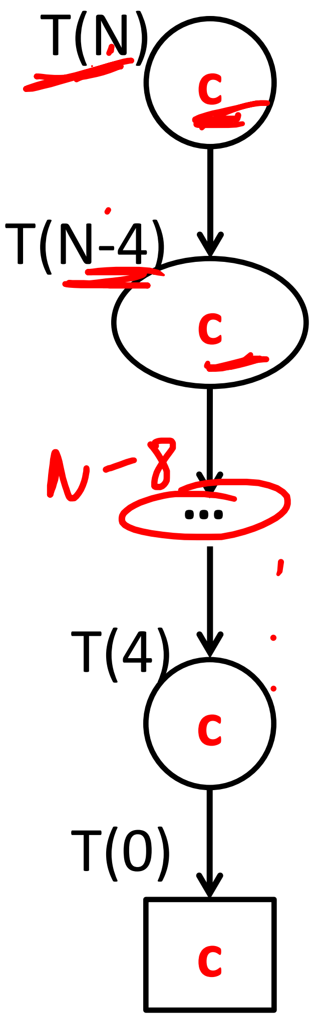
T(N) = T(N-1) + c
T(1) = c
T(0) = c

Levels: N
Each node has TC c =>
T(N) = c*N = Θ(N)



$T(N) = T(N-4) + c$

Time complexity tree:




```
int fact4(int N)
{
    if (N <= 1) return 1;
    if (N == 2) return 2;
    if (N == 3) return 6
    return N*(N-1)*(N-2)*(N-3)*fact4(N-4);
}
```

Time complexity of fact4(N) ? T(N) = …

$T(N) = T(N-4) + c$

$T(3) = c$

$T(2) = c$

$T(1) = c$

$T(0) = c$

$T(N) = c \quad \forall N \le 3$

Levels: ≈N/4

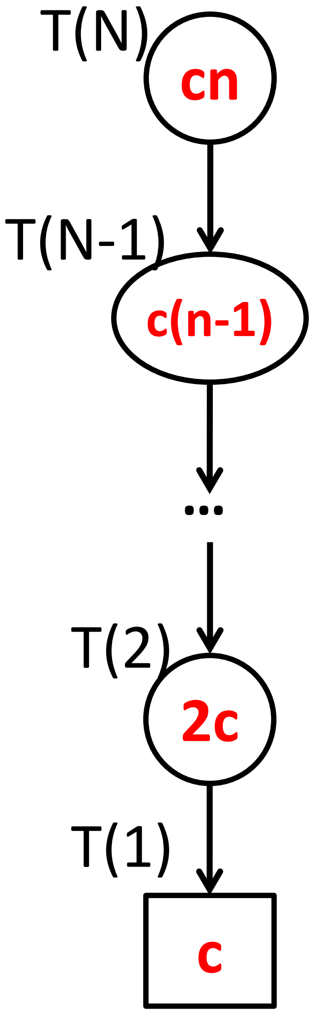
Each node has T c =>

$T(N) = c*N/4 = \Theta(N)$

$T(N) = T(N-1) + cN$

selection_sort_rec(N)

Time complexity tree:



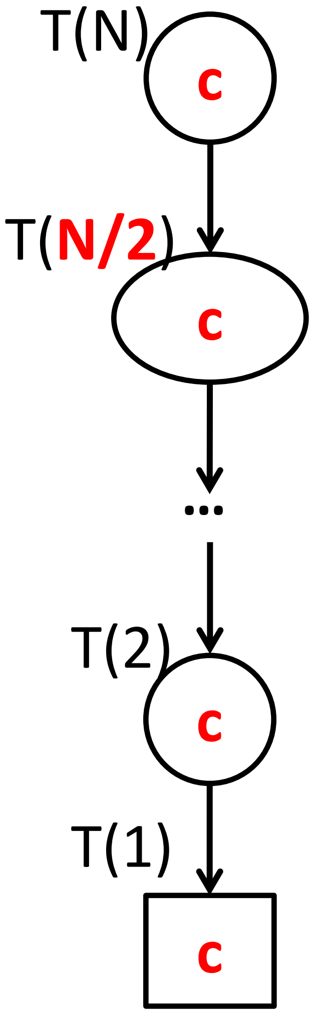

```
int fact(int N, int st, int[] A, ){
    if (st >= N-1) return;
    idx = min_index(A,st,N); // Θ(N-st)
    A[st] <-> A[idx]
    return sel_sort_rec(A,st+1,N);
}
```

$T(N) = T(N-1) + cN$
$T(1) = c$
$T(0) = c$

Levels: N
Node at level i has TC c(N-i) =>
$T(N) = cN+c(N-1)+\ldots ci+\ldots c = cN(N+1)/2 = \Theta(N^2)$

$$T(N) = T(N/2) + c$$

Time complexity tree:

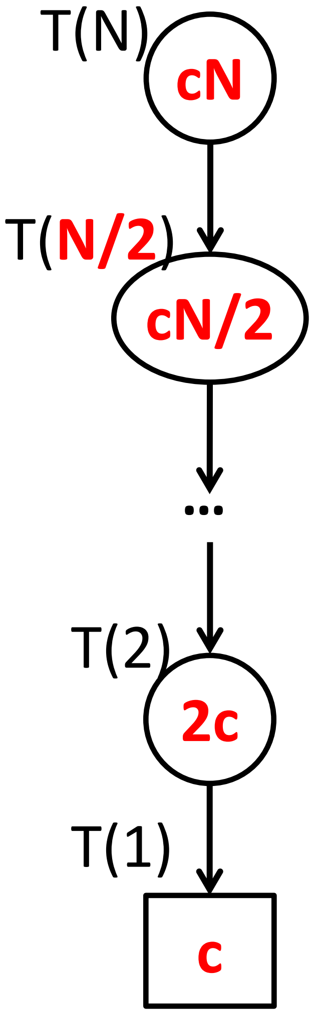


$T(N) = T(N/2) + c$
$T(1) = c$
$T(0) = c$

Levels: $\approx \lg N$ (from base case: $N/2^p=1 \Rightarrow p=\lg N$)
Each node has TC c =>
$T(N) = c*\lg N = \Theta(\lg N)$

$T(N) = T(N/2) + cN$

Time complexity tree:



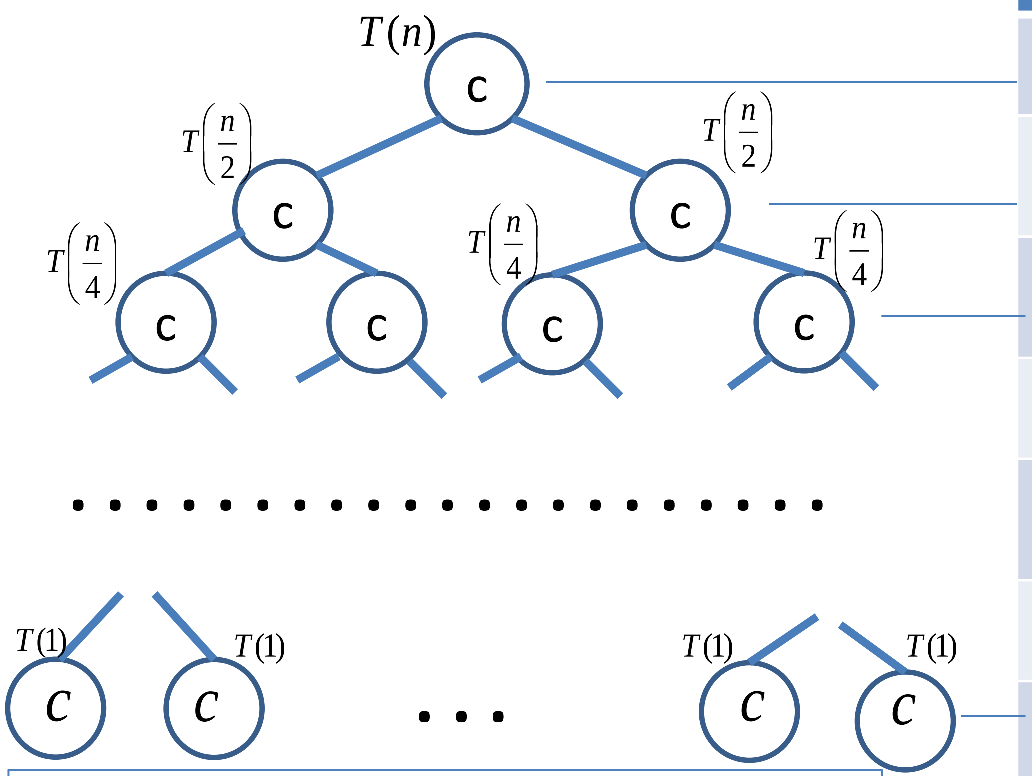
$T(N) = T(N/2) + cN$
$T(1) = c$
$T(0) = c$

Levels: $\approx \lg N$ (from base case: $N/2^p=1 \Rightarrow p=\lg N$)
Node at level i has TC $cN/2^i$ =>

$$
\begin{aligned}
T(N) &= c(N + N/2 + N/2^2 + \dots N/2^i + \dots + N/2^k) = \\
&= cN(1 + 1/2 + 1/2^2 + \dots 1/2^i + \dots + 1/2^k) = \\
&= cN[1 + (1/2) + (½)^2 + \dots (½)^i + \dots + (½)^p] = \\
&= cN^*\text{constant} \\
&= \Theta(N)
\end{aligned}
$$

Recursion Tree for: T(n) = 2T(n/2)+c

Base case: T(1) = c



Level	Arg/ pb size	TC of 1 node	Nodes per level	Level TC
0	n	c	1	c
1	n/2	c	2	2c
2	n/4	c	4	4c
…				
i	$n/2^i$	c	2^i	$2^i c$
…				
p=lgn	1 ($=n/2^p$)	c	2^p (=n)	$2^k c$

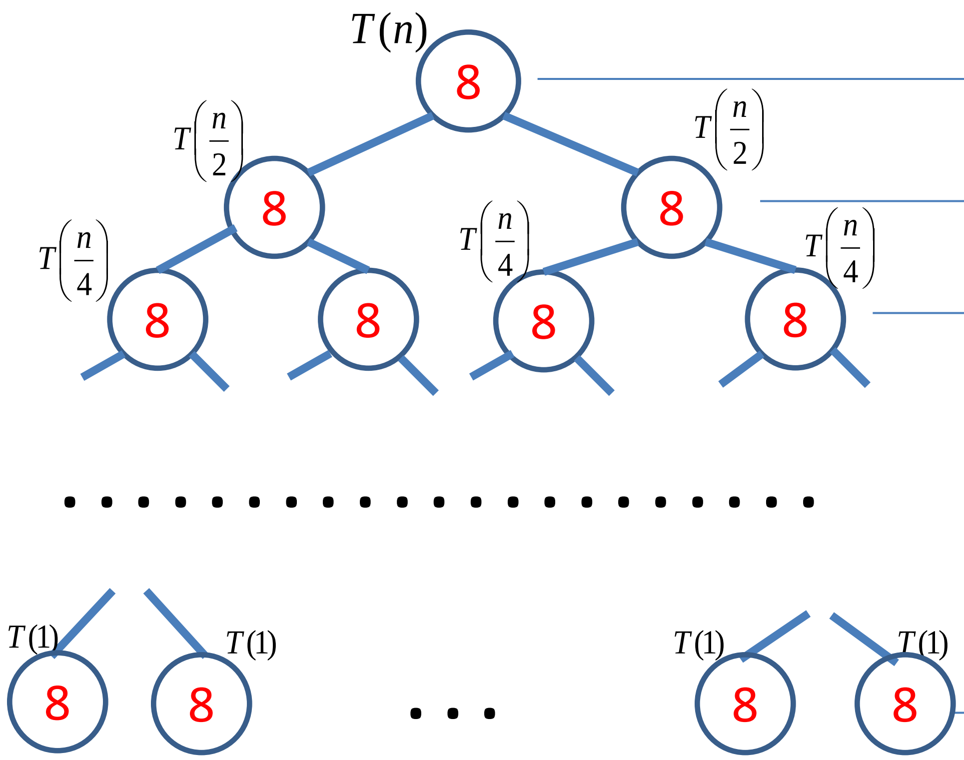
Stop at level p, when the subtree is T(1).
=> The problem size is 1, but the general formula for the problem size, at level p is: $n/2^p$=> $n/2^p$= 1 => p = lgn

Tree TC $= c(1+2+2^2+2^3+\ldots+2^i+\ldots+2^p) = c2^{p+1}/(2-1)$
$= 2c2^p = 2cn = \Theta(n)$

Recursion Tree for: T(n) = 2T(n/2)+8

If specific value is given instead of c, use that. Here c=8.

Base case: T(1) = 8



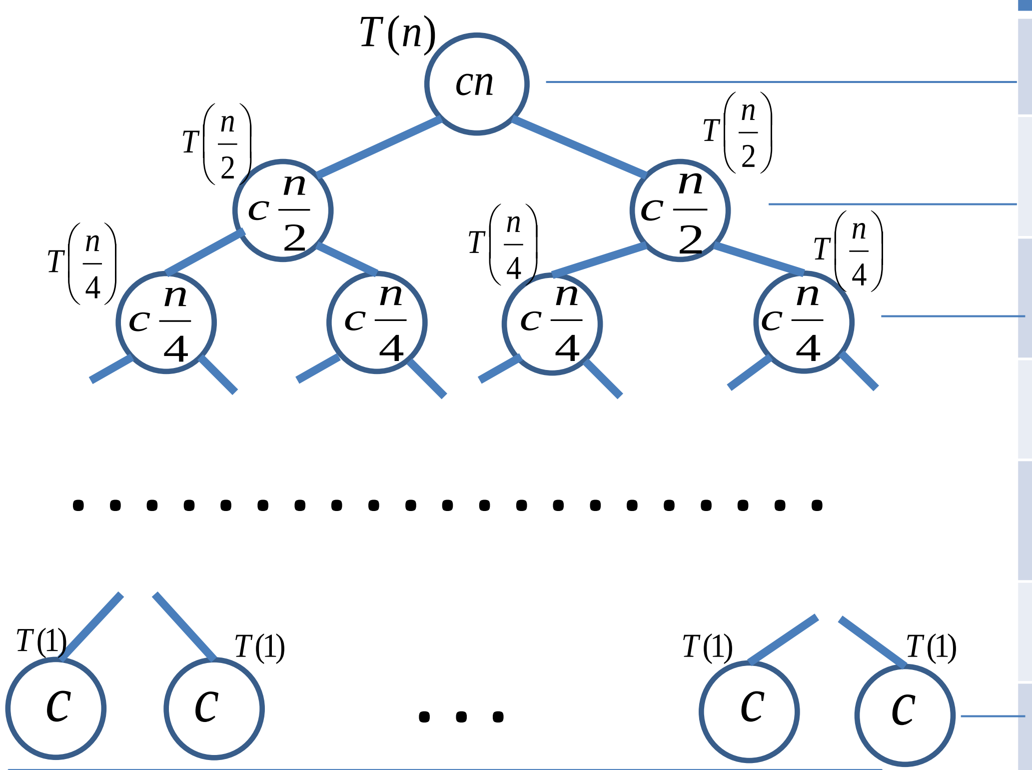
Level	Arg/ pb size	TC of 1 node	Nodes per level	Level TC
0	n	8	1	**8**
1	n/2	8	2	2***8**
2	n/4	8	4	4*8
…				
i	$n/2^i$	8	2^i	2^i*8
…				
k=lgn	1 (=$n/2^k$)	8	2^k (=n)	2^k*8

Stop at level p, when the subtree is T(1).
=> The problem size is 1, but the general formula for the problem size, at level p is:
$n/2^p$=> $n/2^p$= 1 => 2^p= n => p = lgn

Tree TC = $c(1+2+2^2+2^3+\ldots+2^i+\ldots+2^p)=8*2^{p+1}/(2-1)$
$= 2*8*2^p = 16n = \Theta(n)$

Recursion Tree for: T(n) = 2T(n/2)+cn

Base case: T(1) = c



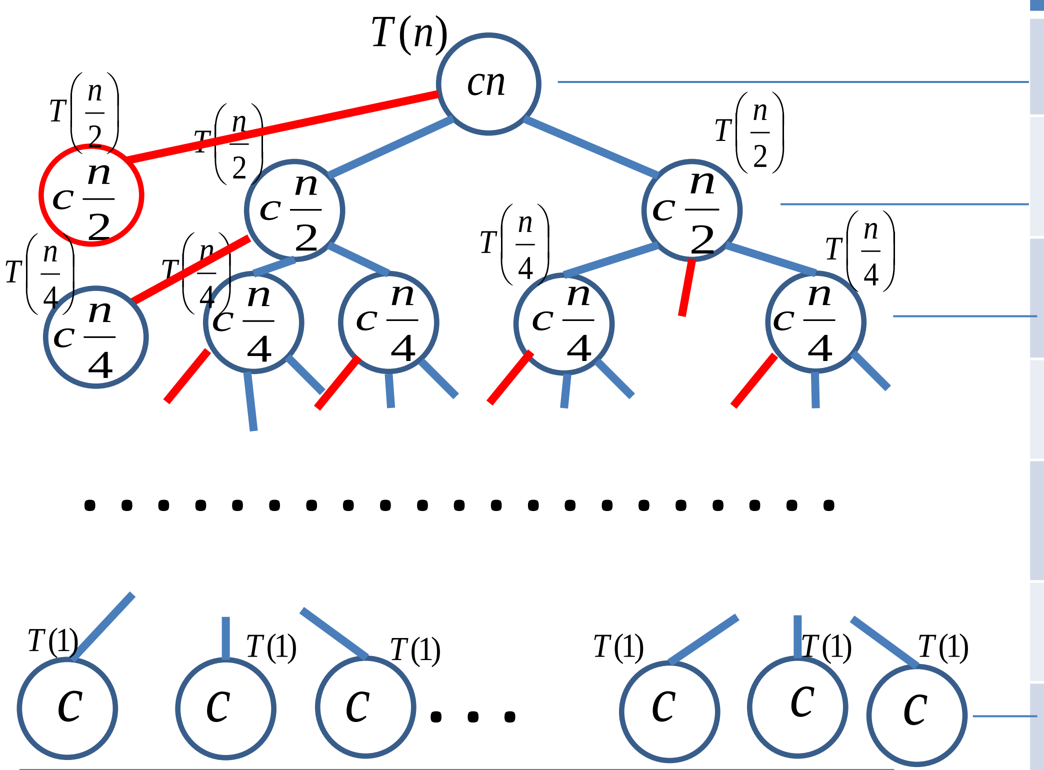
Level	Arg/ pb size	TC of 1 node	Nodes per level	Level TC
0	n	c*n	1	c*n
1	n/2	c*n/2	2	2*c*n/2 =c*n
2	n/4	c*n/4	4	4*c*n/4 =c*n
…				
i	$n/2^i$	$c*n/2^i$	2^i	$2^i*c*n/2^i$ =c*n
…				
p=lgn	1 $(=n/2^p)$	$c=c*1= c*n/2^p$	2^p (=n)	$2^p*c*n/2^p$ =c*n

Stop at level p, when the subtree is T(1).
=> The problem size is 1, but the general formula for the problem size, at level p is:
$n/2^p$ => $n/2^p=1$ => $2^p=n$ => p= lgn

Tree TC $= cn(p+1) = cn(1+lgn)$
$= cnlgn + cn = \theta(nlgn)$

Recursion Tree for T(n) = 3T(n/2)+cn

Base case: T(1) = c



Stop at level p, when the subtree is T(1).
=> The problem size is 1, but the general formula for the problem size, at level p is:
$n/2^p$=> $n/2^p=1$ => $2^p=n$ => p = lgn

Level	Arg/ pb size	TC of 1 node	Nodes per level	Level TC
0	n	c*n	1	c*n
1	n/2	c*n/2	3	3*c*n/2 =(3/2)*c*n
2	n/4	c*n/4	9	$(3/2)^2$*c*n
…				
i	$n/2^i$	$c*n/2^i$	3^i	$(3/2)^i$*c*n
…				
p=lgn	1 ($=n/2^p$)	c=c*1= $c*n/2^p$	3^p (≠n)	$(3/2)^p$*c*n

Total Tree TC for T(n) = 3T(n/2)+cn

Closed form

$$T(n) = cn + (3/2)cn + (3/2)^2 cn + \ldots (3/2)^i cn + \ldots (3/2)^{\lg n} cn =$$

$$= cn * [1 + (3/2) + (3/2)^2 + \ldots + (3/2)^{\lg n}] = cn \sum_{i=0}^{\lg n} (3/2)^i =$$

$$= cn * \boxed{\frac{(3/2)^{\lg n+1} - 1}{(3/2) - 1}} = 2cn[(3/2)*(3/2)^{\lg n} - 1] = 3cn*(3/2)^{\lg n} - 2cn$$

$$use : c^{\lg n} = n^{\lg c} \Rightarrow (3/2)^{\lg n} = n^{\lg(3/2)} = n^{\lg 3 - \lg 2} = n^{\lg 3 - 1} \Rightarrow$$

$$= 3cn * n^{\lg 3 - 1} - 2cn = 3cn^{1+\lg 3 - 1} - 2cn = 3cn^{\lg 3} - 2cn = \Theta(n^{\lg 3})$$

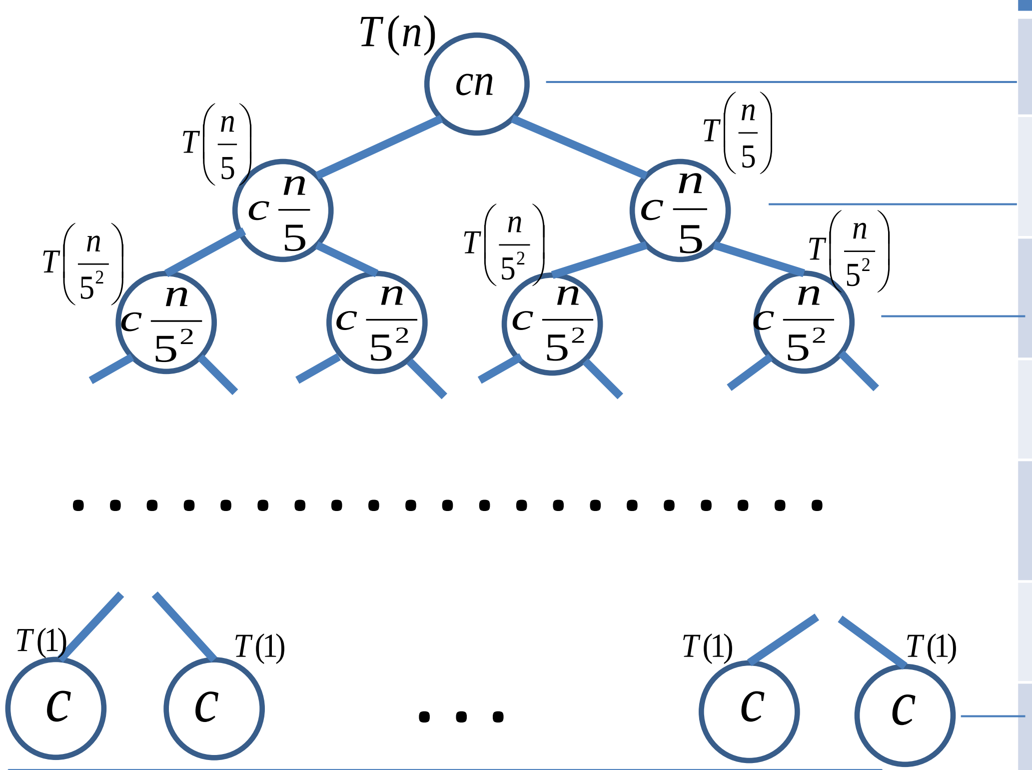
Explanation: since we need Θ, we can eliminate the constants and non-dominant terms earlier (after the closed form expression):

$$\ldots = cn * \frac{(3/2)^{\lg n+1} - 1}{(3/2) - 1} = \Theta(n*(3/2)*(3/2)^{\lg n+1}) = \Theta(n*(3/2)^{\lg n})$$

$$use : c^{\lg n} = n^{\lg c} \Rightarrow (3/2)^{\lg n} = n^{\lg(3/2)} = n^{\lg 3 - \lg 2} = n^{\lg 3 - 1} \Rightarrow$$

$$= \Theta(n * n^{\lg 3 - 1}) = \Theta(n^{\lg 3})$$

Recursion Tree for: $T(\boxed{n}) = 2T(n/5)+c\boxed{n}^3$



Level	Arg/ pb size	TC of 1 node	Nodes per level	Level TC
0	n	c*n	1	c*n
1	n/5	$c*\left(n/5\right)^3$	2	2*c*n/5 =(2/5)*cn
2	$n/5^2$	$c*\left(n/5^2\right)^3$	4	4*c*n/ $=(2/5)^i$cn
…				
i	$n/5^i$	$c*n/5^i$	2^i	$2^i*c*n/5^i$ $=(2/5)^i$cn
…				
p= $\log_5 n$	1 (=$n/5^p$)	c=c*1= $c*n/5^p$	2^p (=n)	$2^p*c*n/5^p$ $=(2/5)^p$cn

Stop at level p, when the subtree is T(1).
=> The problem size is 1, but the general formula for the problem size, at level p is:
$n/5^p$=> $n/5^p$= 1 => 5^p=n => p = $\log_5 n$

Tree TC
(derivation similar to TC for T(n) = 3T(n/2)+cn)

Total Tree TC for T(n) = 2T(n/5)+cn

$$T(n) = cn + (2/5)cn + (2/5)^2 cn + ...(2/5)^i cn + ...(2/5)^{\log_5 n} cn =$$

$$= cn * [1 + (2/5) + (2/5)^2 + ... + (2/5)^{\log_5 n}] =$$

$$= cn \sum_{i=0}^{\log_5 n} (2/5)^i \le cn \sum_{i=0}^{\infty} (2/5)^i =$$

$$= cn * \frac{1}{1-(2/5)} = (5/3)cn = O(n)$$

$$Also$$

$$T(n) = cn + ... \Rightarrow T(n) \ge cn \Rightarrow T(n) = \Omega(n)$$

$$\Rightarrow T(n) = \Theta(n)$$

Other Variations

- $T(n) = \mathbf{7}T(n/\mathbf{3})+cn$
- $T(n) = 7T(n/3)+ c\mathbf{n^5}$
  - Here instead of $(7/3)$ we will use $(7/3^5)$
- $T(n) = T(n/2) + n$
  - The tree becomes a chain (only one node per level)

Additional materials

Practice/Strengthen understanding Problem

- Look into the derivation if we had: $T(1) = d \neq c$.
  - In general, at most, it affects the constant for the dominant term.

Practice/Strengthen understanding Answer

- Look into the derivation if we had: T(1) = d ≠ c.
  - At most, it affects the constant for the dominant term.

Level	Arg/ pb size	TC of 1 node	Nodes per level	Level TC
0	n	c*n	1	c*n
1	n/2	c*n/2	2	2*c*n/2 =c*n
2	n/4	c*n/4	4	4*c*n/4 =c*n
…				
i	$n/2^i$	$c*n/2^i$	2^i	$2^i*c*n/2^i$ =c*n
…				
p=lgn	1 $(=n/2^p)$		2^p (=n)	=**d*n**

Tree TC $= cnp + dn = cnlgn + dn = \theta(nlgn)$

Permutations without repetitions (Harder Example)

- Covering this material is subject to time availability

- Time complexity
  - Tree, intuition (for moving the local TC in the recursive call TC), math justification
  - induction

More Recurrences
Extra material – not tested on

M1. Reduce the problem size by 1 in logarithmic time
- E.g. Check lg(N) items, eliminate 1

M2. Reduce the problem size by 1 in N^2 time
- E.g. Check N^2 pairs, eliminate 1 item

M3. Algorithm that:
- takes $\Theta(1)$ time to go over N items.
- calls itself 3 times on data of size N-1.
- takes $\Theta(1)$ time to combine the results.

M4. ** Algorithm that:
- calls itself N times on data of size N/2.
- takes $\Theta(1)$ time to combine the results.
- This generates a difficult recursion.