Recurrences: Methods and Examples

CSE 2320 – Algorithms and Data Structures
Alexandra Stefan

University of Texas at Arlington

Updated: 10/7/2018

Background

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

    $T(n/8) = \ldots\ldots\ldots\ldots\ldots\ldots$

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

    nodes per level

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

- 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.

  - In order to find their complexity, we need to:
    - Express the “running time” 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.
  - I will also accept this method as proof for the given bound (if done correctly).
- The Induction Method
  - 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

Code => Recurrence => Θ

```
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)

Master Theorem Method

Let $a \geq 1$ and $b > 1$, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence: $\boldsymbol{T(n) = aT\left(\frac{n}{b}\right) + f(n),}$

where we interpret $\frac{n}{b}$ to mean either $\lfloor n/b \rfloor$ or$\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{(log_b a)-\varepsilon})$ for some constant ε > 0, then $T(n) = \theta(n^{log_b a})$
2. If $f(n) = \theta(n^{log_b a})$, then $T(n) = \theta(n^{log_b a} lgn)$.
3. If $f(n) = \Omega(n^{(log_b a)+\varepsilon})$, for some constant $\varepsilon > 0$, and if a$f(\frac{n}{b}) \leq kf(n)$ for some constant $k < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Solve: T(n) = 9T(n/3)+n
T(n) = T(2n/3)+1
T(n) = 3T(n/4)+nlgn
Not applicable for:
T(n) = 2T(n/2)+nlgn
See CLRS , page 95, for solutions.

Master Theorem

Let $\boldsymbol{T(n) = aT\left(\frac{n}{b}\right) + f(n)}$, where $a \geq 1, b > 1$ and $n > 0$.

1. If $f(n) = O(n^{(\log_b a)-\varepsilon})$ for some constant ε > 0, then $T(n) = \theta(n^{\log_b a})$
2. If $f(n) = \theta(n^{\log_b a})$, then $T(n) = \theta(n^{\log_b a} lgn)$.
3. If $f(n) = \Omega(n^{(\log_b a)+\varepsilon})$, for some constant $\varepsilon > 0$, and if a$f(\frac{n}{b}) \leq kf(n)$ for some constant $k < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

$T(n) = 9T(n/3) + n$	$T(n) = T(2n/3) + 1$
$a = 9$	$a = 1$
$b = 3$	$b = (3/2)$
$\log_b a = \log_3 9 = 2 \Rightarrow n^{\log_b a} = n^2$	$\log_b a = \log_{(3/2)} 1 = 0 \Rightarrow n^{\log_b a} = n^0 = 1$
$f(n) = n = O(n^{2-0.5}), \varepsilon = 0.5 \Rightarrow case1$	$f(n) = 1 = \Theta(n^0) \Rightarrow case2$
$\Rightarrow T(n) = 9T(n/3) + n = \Theta(n^2)$	$\Rightarrow T(n) = T(2n/3) + 1 = \Theta(n^0 \lg n) = \Theta(\lg n)$

Master Theorem

Let $\boldsymbol{T(n) = aT\left(\frac{n}{b}\right) + f(n)},$ where $a \geq 1, b > 1$ and $n > 0$.

1. If $f(n) = O(n^{(log_b a)-\varepsilon})$ for some constant ε > 0, then $T(n) = \theta(n^{log_b a})$
2. If $f(n) = \theta(n^{log_b a})$, then $T(n) = \theta(n^{log_b a} lgn)$.
3. If $f(n) = \Omega(n^{(log_b a)+\varepsilon})$, for some constant $\varepsilon > 0$, and if a$f\left(\frac{n}{b}\right) \leq kf(n)$ for some constant $k < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

T(n) = 3T($\lfloor n/4 \rfloor$) + nlgn
a=3
b=4
$\log_4 3 \approx 0.8$ => $n^{log_b a} \approx n^{0.8}$
f(n) = nlgn
nlgn=$\Omega(n^{0.8+0.1})$, ε=0.1 => case 3
We also need: af(n/b)≤kf(n) =>
3(n/4)lg(n/4)≤(3/4)nlgn ≤knlgn
True ∀ k≥(3/4). Need k<1 => k = ¾

=> T(n)=3T(n/4)+nlgn=Θ(nlgn)

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

$a = 2$

$b = 3$

$\log_2 2 = 1 \Rightarrow n^{\log_b a} = n^1 = n$

$f(n) = n\lg n$

$n\lg n = \Omega(n^1)$ but not by ε

$B.c., \forall \varepsilon > 0, \lg n = O(n^{\varepsilon}) \Rightarrow$

$n\lg n = O(n * n^{\varepsilon}) = O(n^{1+\varepsilon})$

=> Cannot apply case 3 (or cases 1 or 2) =>
Cannot solve it with the Master Theorem

Recurrences:
Recursion-Tree Method

1. Build the tree & fill-out the table
2. Compute cost 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 $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$

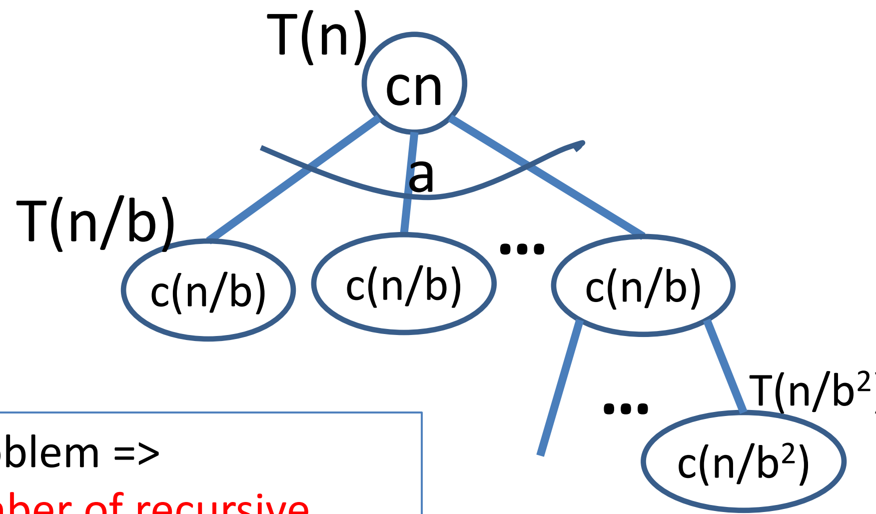
Example 2 : Solve $T(n) = T(n/3) + T(2n/3) + O(n)$

Recurrence - Recursion Tree Relationship

The local cost at the node

Problem size

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



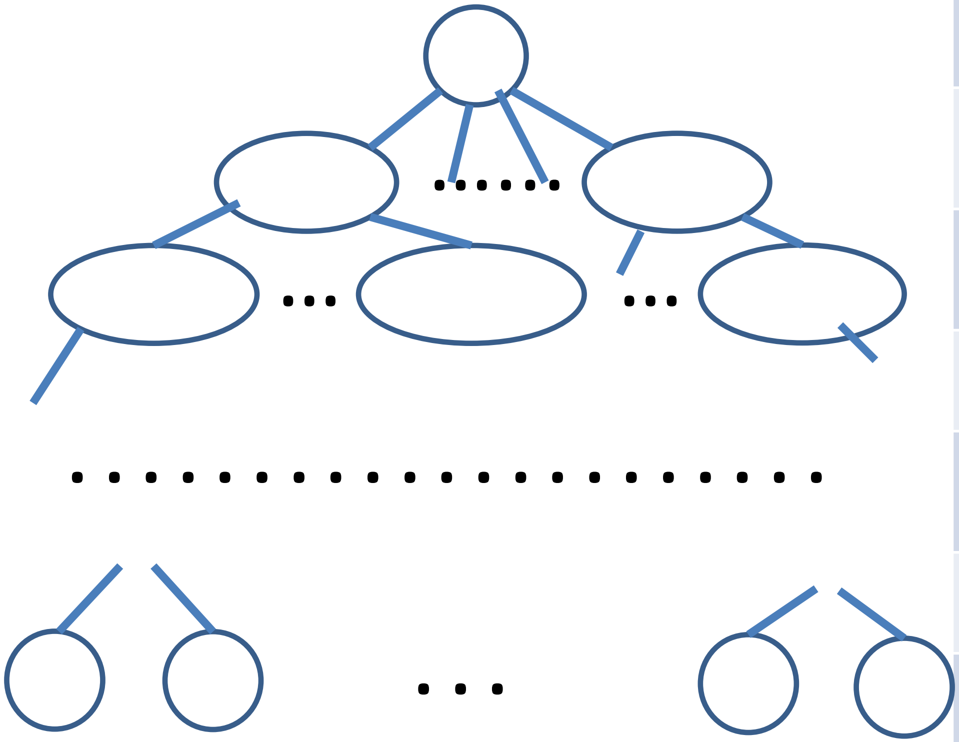
Number of subproblems =>
Number of children of a node in the recursion tree. =>
Affects the number of nodes per level. At level i there will be a^i nodes.
Affects the level cost.

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

Recursion Tree for: $T(n) = 7T(n/5)+cn^3$

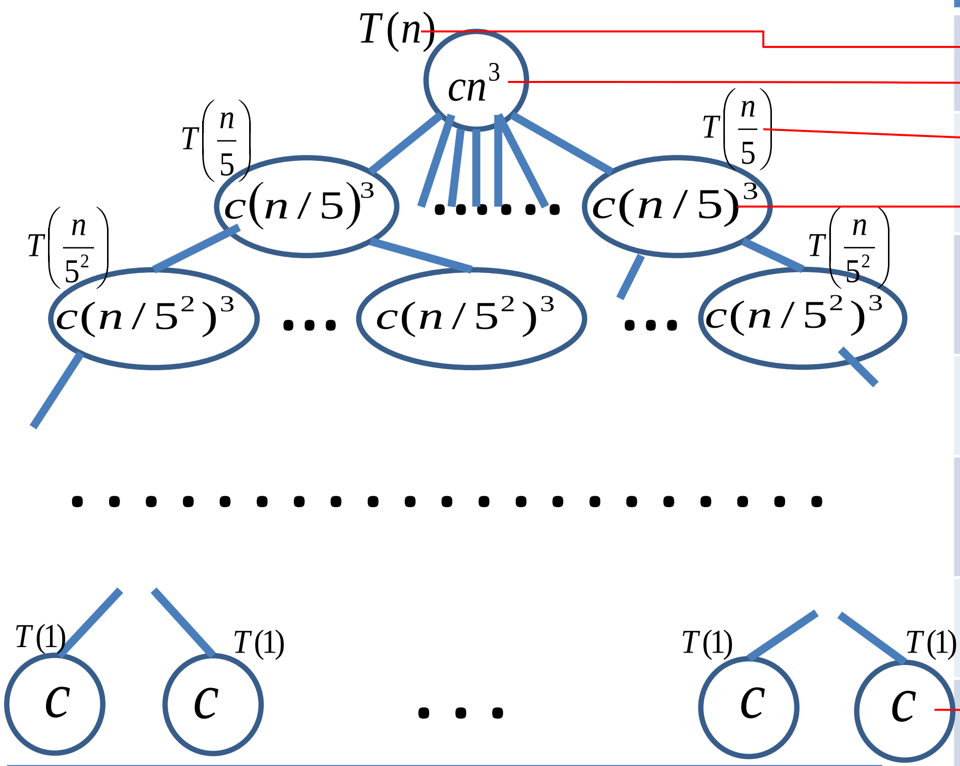
Base case: $T(1) = c$

Work it out by hand in class.



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

Recursion Tree for:
$T(n) = 7T(n/5)+cn^3$, Base case: $T(1) = c$



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

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

Where we used:
$7^i* (n/5^i)^3 = 7^i* n^3 (1/5^i)^3 = 7^i* n^3 (1/5^3)^i = n^3(7/5^3)^i$

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

More practice/ Special cases

Tree Method

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

- 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 cost is cn (the sum of costs of the children equals the parent: (1/3)*p_cost+(2/3) *p_cost)
  - $\Rightarrow$ Total cost for those: $cn*\log_3 n = \Theta(n\lg n)$
  - The number of incomplete levels should also be a multiple of lgn and the cost 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 cost 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).

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:

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

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:

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

$$want:$$

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

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):

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

Common Recurrences

	Local cost	*Number* of sub-problems	*Size* of sub-problem	T(n)	Description Example
1	Θ(1) c	1	n/2		*Halve* problem in const time
2	Θ(n) cn	1	n/2		*Halve* problem in linear time
3	Θ(1) c	**2**	n/2		Break (and put back together) the problem into *2 halves* in const time.
4	Θ(n) cn	**2**	n/2		Break (and put back together) the problem into *2 halves* in linear time.
5	Θ(1) c	1	n-1		Reduce the pb size *by 1* in const time.
6	Θ(n) cn	1	n-1		Reduce the pb size *by 1* in linear time.

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:

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

Recurrence => Code Worksheet

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

For $T(N) = N*T(N-1) + \Theta(N)$

Replace $\Theta(N)$ with cN and: $T(N) = N*T(N-1) + cN$

Recurrence => Code Answers

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

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

Compare

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

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

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

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

```
int foo3(int N){
     if (N <= 1) return 5;
     for(int i=1; i<=N; i++){
        return foo3(N-1);
   // No loop. Returns after the first iteration.
     }
}
T(N) = T(N-1) + c
```

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:

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

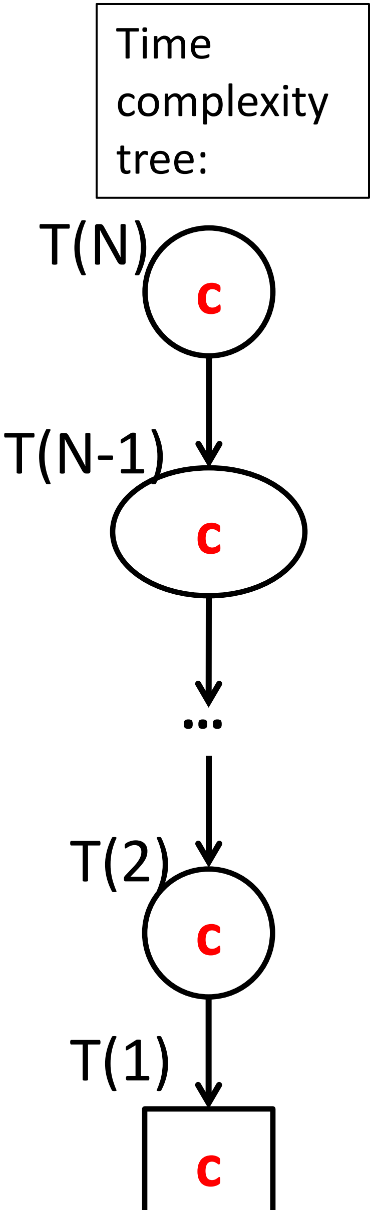
$T(n) = 7T(n/3)+cn^3$

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

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

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

fact(N)




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

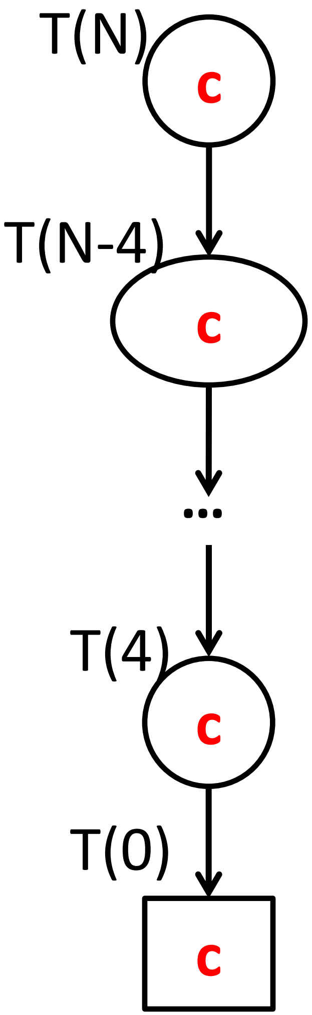
Time complexity of fact(N) ? T(N) = …

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

Levels: N
Each node has cost c =>
$T(N) = c*N = \Theta(N)$

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

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) = …

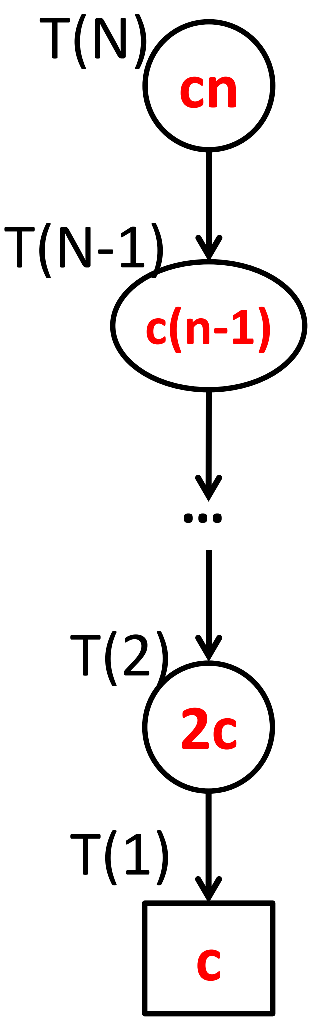
$T(N) = T(N-4) + c$
$T(3) = c$
$T(2) = c$
$T(1) = c$
$T(0) = c$

Levels: $\approx N/4$
Each node has cost c =>
$T(N) = c*N/4 = \Theta(N)$

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

selection_sort_rec(N)

Time complexity tree:



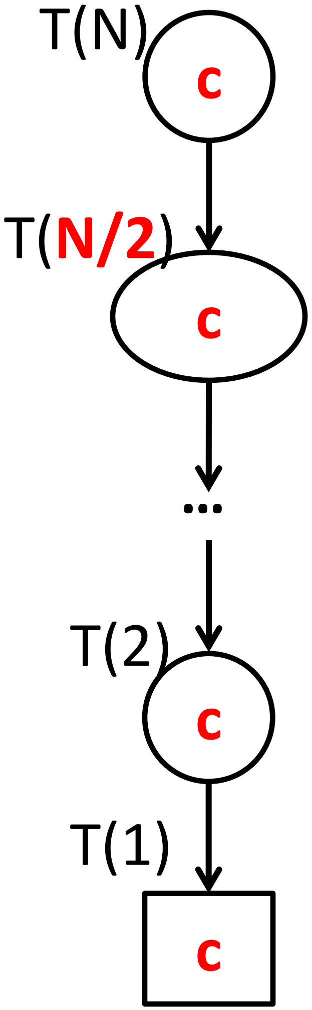

```
int fact(int N, int st, int[] A, ){
    if (st >= N-1) return;
    idx = min_index(A,st,N); // Θ(N-st)
    A[st] <-> A[idx]
    return sel_sort_rec(A,st+1,N);
}
```

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

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

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

Time complexity tree:

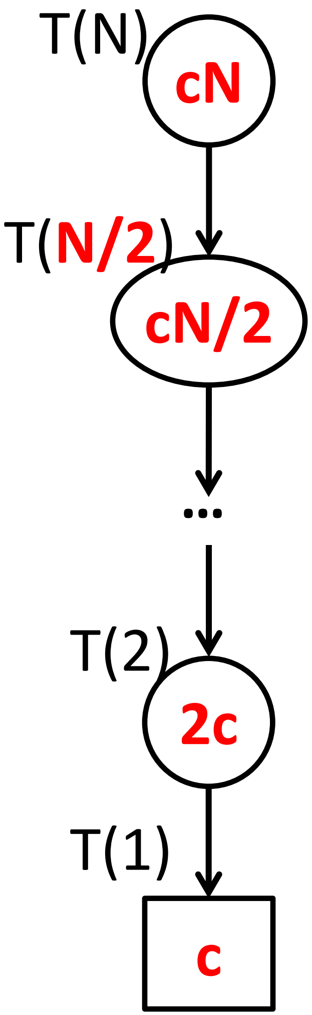


$T(N) = T(N/2) + c$
$T(1) = c$
$T(0) = c$

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

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

Time complexity tree:



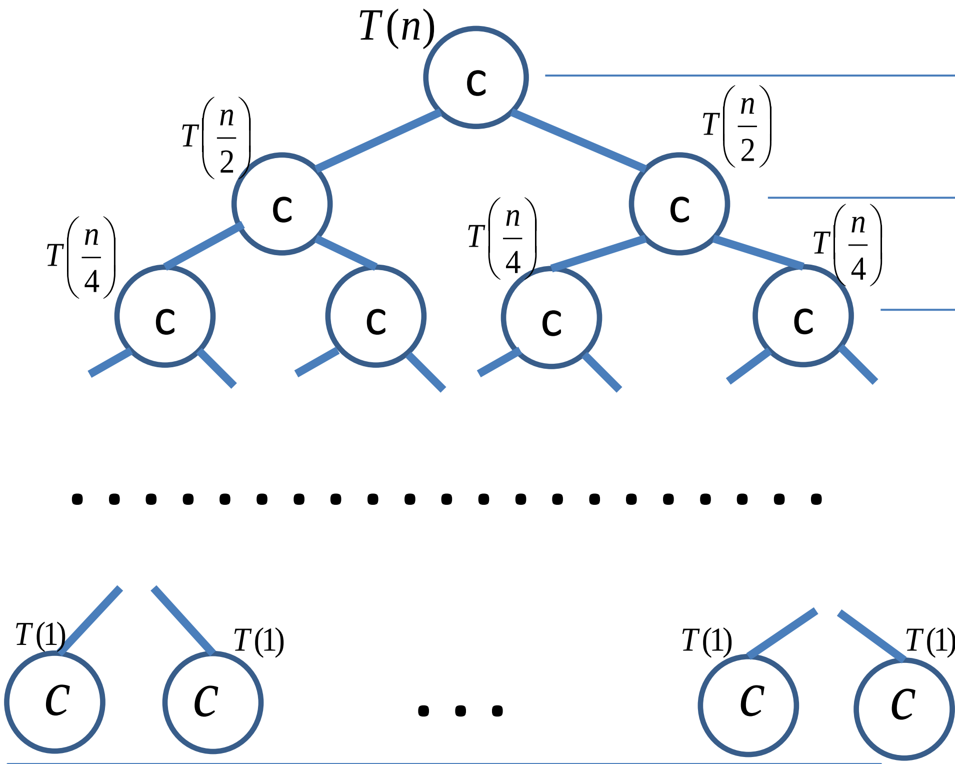
$T(N) = T(N/2) + cN$
$T(1) = c$
$T(0) = c$

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

$$T(N) = c(N + N/2 + N/2^2 + \ldots N/2^i + \ldots + N/2^k) =$$
$$= cN(1 + 1/2 + 1/2^2 + \ldots 1/2^i + \ldots + 1/2^k) =$$
$$= cN[1 + (1/2) + (½)^2 + \ldots (½)^i + \ldots + (½)^k] =$$
$$= cN*\text{constant}$$
$$= \Theta(N)$$

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

Base case: $T(1) = c$



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

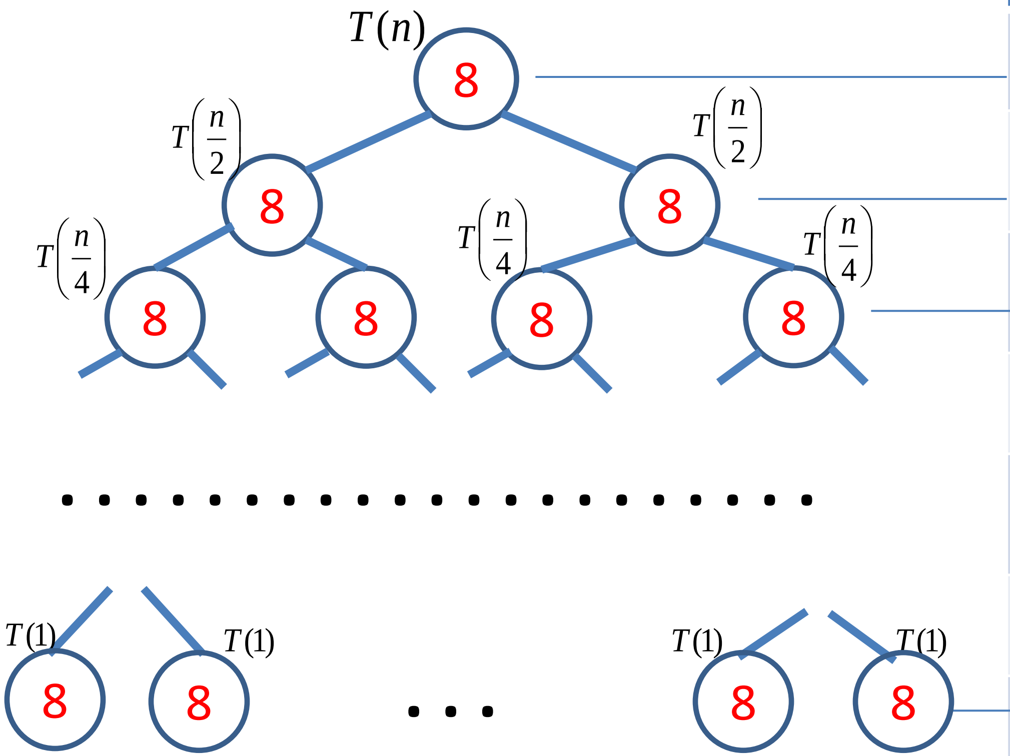
Stop at level k, when the subtree is T(1).
=> The problem size is 1, but the general formula for the problem size, at level k is:
$n/2^k$ => $n/2^k = 1$ => $k = \lg n$

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

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

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

Base case: $T(1) = c$



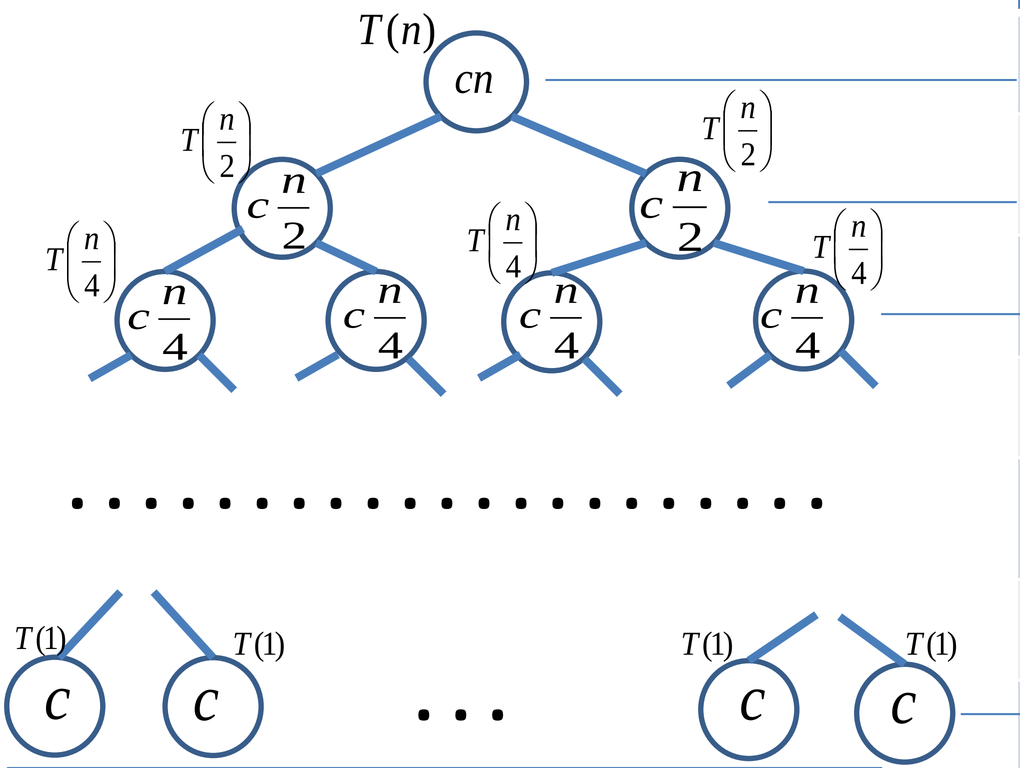
Level	Arg/ pb size	cost of 1 node	Nodes per level	Level cost
0	n	8	1	8
1	n/2	8	2	2*8
2	n/4	8	4	4*8
…				
i	$n/2^i$	8	2^i	2^i*8
…				
k=lgn	1 ($=n/2^k$)	8	2^k (=n)	2^k*8

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

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

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

Base case: T(1) = c



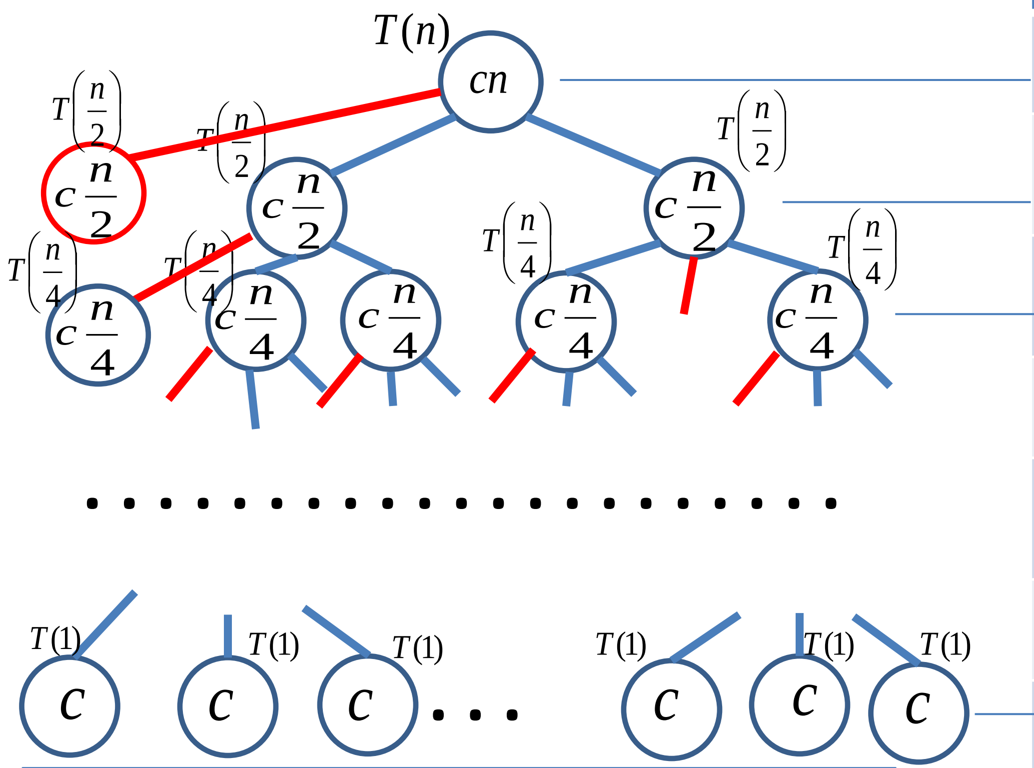
Level	Arg/ pb size	cost of 1 node	Nodes per level	Level cost
0	n	c*n	1	c*n
1	n/2	c*n/2	2	2*c*n/2 =c*n
2	n/4	c*n/4	4	4*c*n/4 =c*n
…				
i	$n/2^i$	$c*n/2^i$	2^i	$2^i*c*n/2^i$ =c*n
…				
k=lgn	1 $(=n/2^k)$	c=c*1= $c*n/2^k$	2^k (=n)	$2^k*c*n/2^k$ =c*n

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

Tree cost $= cn*(k+1) = cn*(1+\lg n) =$
$= cn\lg n + cn = \Theta(n\lg n)$

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

Base case: T(1) = c



Level	Arg/ pb size	cost of 1 node	Nodes per level	Level cost
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
…				
k=lgn	1 ($=n/2^k$)	c=c*1= $c*n/2^k$	**3^k** ($\neq$n)	**$(3/2)^k$***c*n

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

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

Closed form

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

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

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

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

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

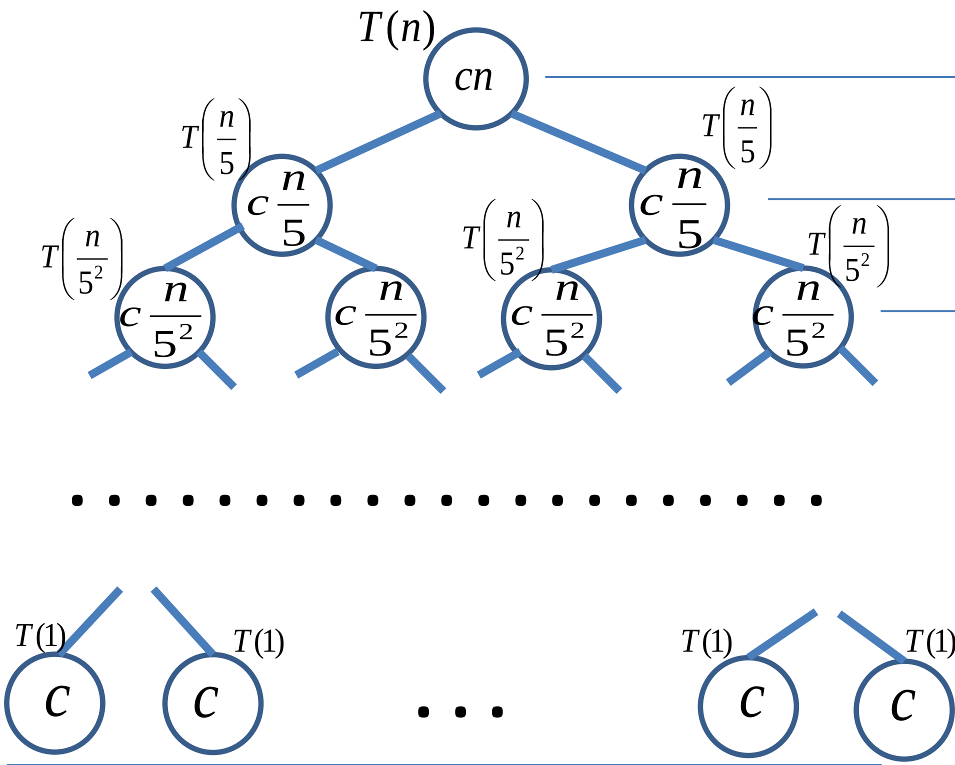
Explanation: since we need Θ, we can eliminate the constants and non-dominant terms earlier (after the closed form expression):

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

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

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

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



Level	Arg/ pb size	cost of 1 node	Nodes per level	Level cost
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	$2^i*c*n/5^i$ $=(2/5)^i cn$
…				
k=lgn	1 $(=n/5^k)$	c=c*1= $c*n/5^k$	2^k (=n)	$2^k*c*n/5^k$ $=(2/5)^k cn$

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

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

Total Tree Cost 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

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

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

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 $(7/3^{\mathbf{5}})$

- $T(n) = T(n/2) + n$
  - 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	Cost of 1 node	Nodes per level	Level cost
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
…				
k=lgn	1 $(=n/2^k)$		2^k (=n)	=**d*n**

Tree cost

$$= cnk + dn = cn\lg n + dn = \Theta(n\lg n)$$

Permutations without repetitions (Harder Example)

- Covering this material is subject to time availability

- Time complexity
  - Tree, intuition (for moving the local cost in the recursive call cost), 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.