Recurrences:
Methods and Examples

CSE 2320 – 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 \( T(n) = 3T(n/8) + 4n^{2.5}\log n, \)
    \( T(n/8) = \ldots \)
    \( T(n-1) = \ldots \)

• 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 \( \Theta \), 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.
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(N) = N \cdot 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
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

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

\[ T(n) = a \cdot T\left(\frac{n}{b}\right) + cn \]

- **Problem size**
  - Number of subproblems \( \Rightarrow \) Number of children of a node in the recursion tree. \( \Rightarrow \) Affects the number of nodes per level. At level \( i \) there will be \( a^i \) nodes. Affects the level cost.

- **The local cost at the node**
  - \( cn \)

- **Size of a subproblem** \( \Rightarrow \) 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 \( T(1) \) ) \( \Rightarrow n/b^k = 1 \) \( \Rightarrow \) 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.
Recursion Tree for:
\[ T(n) = 7T(n/5) + cn^3 \], Base case: \( T(1) = c \)

<table>
<thead>
<tr>
<th>Level</th>
<th>Arg/ pb size</th>
<th>cost of 1 node</th>
<th>Nodes per level</th>
<th>Level cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( n )</td>
<td>( cn^3 )</td>
<td>1</td>
<td>( c*n^3 )</td>
</tr>
<tr>
<td>1</td>
<td>( n/5 )</td>
<td>( c(n/5)^3 )</td>
<td>7</td>
<td>( 7<em>c</em>(n/5)^3 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>= ( cn^3 (7/5^3) )</td>
</tr>
<tr>
<td>2</td>
<td>( n/5^2 )</td>
<td>( c(n/5^2)^3 )</td>
<td>7^2</td>
<td>( 7^2<em>c</em>(n/5^2)^3 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>= ( cn^3 (7/5^3)^2 )</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( i )</td>
<td>( n/5^i )</td>
<td>( c(n/5^i)^3 )</td>
<td>( 7^i )</td>
<td>( 7^i<em>c</em>(n/5^i)^3 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>= ( cn^3 (7/5^3)^i )</td>
</tr>
<tr>
<td>( k = \log_5 n )</td>
<td>( 1 = n/5^k )</td>
<td>( c = c*1 = c(n/5^k)^3 )</td>
<td>( 7^k )</td>
<td>( 7^k<em>c</em>(n/5^k)^3 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>= ( cn^3 (7/5^3)^k )</td>
</tr>
</tbody>
</table>

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:
\( n/5^k \Rightarrow n/5^k = 1 \Rightarrow k = \log_5 n \)

Where we used:
\( 7^i * (n/5^i)^3 = 7^i * n^3 (1/5^i)^3 = 7^i * n^3 (1/5^3)^i = n^3(7/5^3)^i \)
Recursion Tree for:
\[ T(n) = 5T(n-6) + c , \text{ Base case: } T(0) = c \]

<table>
<thead>
<tr>
<th>Level</th>
<th>Arg/ pb size</th>
<th>cost of 1 node</th>
<th>Nodes per level</th>
<th>Level cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>n</td>
<td>c</td>
<td>1</td>
<td>c</td>
</tr>
<tr>
<td>1</td>
<td>n-6</td>
<td>c</td>
<td>5</td>
<td>5*c</td>
</tr>
<tr>
<td>2</td>
<td>n-2\times6</td>
<td>c</td>
<td>5^2</td>
<td>5^2*c</td>
</tr>
<tr>
<td>\vdots</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>n-6i</td>
<td>c</td>
<td>5^i</td>
<td>5^i*c</td>
</tr>
<tr>
<td>\vdots</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>0 (=n-6k)</td>
<td>c</td>
<td>5^k</td>
<td>5^k*c</td>
</tr>
</tbody>
</table>

Stop at level \( k \), when the subtree is \( T(0) \).

\[ T(n) = c(1+5+5^2 + 5^3+ \ldots +5^i+\ldots+5^k = c(5^{(k+1)}-1)/(5-1)=\Theta(5^k)= \Theta(5^{n/6}) \]
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: \((1/3) * p\_cost + (2/3) * p\_cost\))
  \[ \Rightarrow \text{Total cost for those: } cn * \log_3 n = \Theta(n \log n) \]
  - The number of incomplete levels should also be a multiple of \( \log n \) and the cost for each of those levels will be less than \( cn \)
  - => Guess that \( T(n) = O(n \log n) \)
- Use the substitution method to show \( T(n) = O(n \log n) \)
- If the recurrence was given with \( \Theta \) instead of \( O \), we could have shown \( T(n) = \Theta(n \log n) \)
  - with \( O \), we only know that: \( T(n) \leq T(n/3) + T(2n/3) + cn \)
  - 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 \( \Theta(n) \) in \( T_1(n) = 2T_1(n/3) + cn \) ?)
  - \( T_2(n) = 2T_2(2n/3) + cn \) (useful: \( \log_3 \approx 1.59 \))
  - Use them to bound \( T(n) \). How does that compare to the analysis in this slide? (The bounds are looser).
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 \( \frac{f(n)}{n^{\log_b a}} = \frac{n/\log n}{n^{\log_n a}} = \frac{n}{n \log n} = \frac{1}{\log n} \). 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 \( T(n) = \Theta(n \log \log n) \).
## Common Recurrences

<table>
<thead>
<tr>
<th></th>
<th>Local cost</th>
<th>Number of sub-problems</th>
<th>Size of sub-problem</th>
<th>T(n)</th>
<th>Description Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\Theta(1)$</td>
<td>1</td>
<td>n/2</td>
<td></td>
<td>Halve problem in constant time</td>
</tr>
<tr>
<td>2</td>
<td>$\Theta(n)$</td>
<td>1</td>
<td>n/2</td>
<td></td>
<td>Halve problem in linear time</td>
</tr>
<tr>
<td>3</td>
<td>$\Theta(1)$</td>
<td>2</td>
<td>n/2</td>
<td></td>
<td>Break (and put back together) the problem into 2 halves in constant time.</td>
</tr>
<tr>
<td>4</td>
<td>$\Theta(n)$</td>
<td>2</td>
<td>n/2</td>
<td></td>
<td>Break (and put back together) the problem into 2 halves in linear time.</td>
</tr>
<tr>
<td>5</td>
<td>$\Theta(1)$</td>
<td>1</td>
<td>n-1</td>
<td></td>
<td>Reduce the pb size by 1 in constant time.</td>
</tr>
<tr>
<td>6</td>
<td>$\Theta(n)$</td>
<td>1</td>
<td>n-1</td>
<td></td>
<td>Reduce the pb size by 1 in linear time.</td>
</tr>
</tbody>
</table>
Common Recurrences Review

1. \textit{Halve} problem in \textbf{constant} time:
   \[ T(n) = T(n/2) + c \ \Theta(\ lg(n) ) \]

2. \textit{Halve} problem in \textbf{linear} time:
   \[ T(n) = T(n/2) + n \ \Theta(n) \ (\sim 2n) \]

3. Break (and put back together) the problem into 2 \textit{halves} in \textbf{constant} time:
   \[ T(n) = 2T(n/2) + c \ \Theta(n) \ (\sim 2n) \]

4. Break (and put back together) the problem into 2 \textit{halves} in \textbf{linear} time:
   \[ T(n) = 2T(n/2) + n \ \Theta(\ n \ lg(n) ) \]

5. Reduce the problem size by 1 in \textbf{constant} time:
   \[ T(n) = T(n-1) + c \ \Theta(\ n) \]

6. Reduce the problem size by 1 in \textbf{linear} time:
   \[ T(n) = T(n-1) + n \ \Theta(\ n^2 ) \]
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: \( T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + n \)

1. Guess that \( T(n) = O(n \log n) \) =>
2. Prove that \( T(n) = O(n \log n) \) using \( T(n) \leq cn \log n \) (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.
2. Prove that $T(n) = O(n \lg n)$, using the definition: find $c$ and $n_0$ s.t. $T(n) \leq c \cdot n \lg n$
(here: $f(n) = T(n)$, $g(n) = n \lg n$)
Show with induction: $T(n) \leq c \cdot n \lg n$ (for some $c > 0$)

$$T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + n \leq 2 \cdot c \cdot \left\lfloor \frac{n}{2} \right\rfloor \cdot \lg \left( \left\lfloor \frac{n}{2} \right\rfloor \right) + n \leq 2 \cdot c \cdot \left( \frac{n}{2} \right) \cdot \lg \left( \frac{n}{2} \right) + n = cn \lg (n/2) + n = cn \lg n - cn + n = cn \lg n + n(1-c)
$$

want:

$$\leq cn \lg n \Rightarrow n(1-c) \leq 0 \Rightarrow 1-c \leq 0 \Rightarrow c \geq 1$$

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 $n \geq n_0$.

$n=1$: $1 = T(1) \leq c \cdot 1 \cdot \lg 1 = c \cdot 0 = 0$
FALSE. => $n_0$ cannot be 1.

$n=2$: $T(2) = 2 \cdot T(1) + 2 = 2 + 2 = 4$
Want $T(2) \leq c \cdot 2 \lg 2 = 2c$, True for: $c \geq 2$

$n=3$: $T(3) = 2 \cdot T(1) + 3 = 2 + 3 = 5$
Want $5 = T(3) \leq c \cdot 3 \cdot \lg 3$
True for: $c \geq 2$

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 \( T(n) \leq cn+1 \) 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

\[ T(n) = T\left(\lfloor n/2 \rfloor\right) + T\left(\lceil n/2 \rceil\right) + 1 \]

Show \( T(n) = O(n) \) using the definition: find \( c \) and \( n_0 \) s.t. \( T(n) \leq c \cdot n \)
(here: \( f(n) = T(n), \ g(n) = n \)). Use induction to show \( T(n) \leq c \cdot n \).

**Inductive step:** assume it holds for all \( m < n \), show for \( n \):

\[
T(n) = T\left(\lfloor n/2 \rfloor\right) + T\left(\lceil n/2 \rceil\right) + 1 \leq c \cdot \lfloor n/2 \rfloor + c \cdot \lceil n/2 \rceil + 1 = c\left(\lfloor n/2 \rfloor + \lceil n/2 \rceil\right) + 1 = cn + 1
\]

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

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

\[
T(n) = T\left(\lfloor n/2 \rfloor\right) + T\left(\lceil n/2 \rceil\right) + 1 \leq c \cdot \lfloor n/2 \rfloor - d + c \cdot \lceil n/2 \rceil - d + 1 = c\left(\lfloor n/2 \rfloor + \lceil n/2 \rceil\right) + 1 - 2d = cn - d + 1 - d
\]

want :

\[
\leq cn - d \Rightarrow 1 - d \leq 0 \Rightarrow d \geq 1
\]
Extra material – Solve:

\[ T(n) = 3T\left(\left\lfloor n/4 \right\rfloor\right) + \Theta(n^2) \]

- 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):
  \[ T(n) = 3T(\left\lfloor n/4 \right\rfloor) + \Theta(n^2) \]
More practice/ Special cases
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)**

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

Time complexity of `fact(N)`? \( T(N) = \ldots \)

\[ T(N) = T(N-1) + c \]
\[ T(1) = c \]
\[ T(0) = c \]

Levels: \( N \)
Each node has cost \( c \) =>
\[ T(N) = c*N = \Theta(N) \]
T(N) = T(N-4) + c

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: ≈N/4
Each node has cost c =>
T(N) = c*N/4 = Θ(N)
T(N) = T(N-1) + cN

`selection_sort_rec(N)`

```
int fact(int N, int st, int[] A, ){
    if (st >= N-1) return;
    idx = min_index(A, st, N);  // Θ(N-st)
    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)+...ci+...c = cN(N+1)/2 = Θ(N²)
\[ T(N) = T(N/2) + c \]

Time complexity tree:

- \[ T(N) = c \]
- \[ T(N/2) = c \]
- \[ T(2) = c \]
- \[ T(1) = c \]

Levels: \( \approx \lg N \) (from base case: \( N/2^k=1 \Rightarrow k=\lg N \))

Each node has cost \( c \) =>

\[ T(N) = c \times \lg N = \Theta(\lg N) \]
\[ T(N) = T(N/2) + cN \]

Time complexity tree:

\[ T(N) \]

\[ cN \]

\[ T(N/2) \]

\[ cN/2 \]

\[ ... \]

\[ T(2) \]

\[ 2c \]

\[ T(1) \]

\[ c \]

Levels: \( \approx \lg N \) (from base case: \( N/2^k = 1 \Rightarrow k = \lg N \))

Node at level \( i \) has cost \( cN/2^i \) =>

\[ T(N) = c(N + N/2 + N/2^2 + \ldots + N/2^i + \ldots + N/2^k) = \]

\[ = cN(1 + 1/2 + 1/2^2 + \ldots + 1/2^i + \ldots + 1/2^k) = \]

\[ = cN[1 + (1/2) + (1/2)^2 + \ldots + (1/2)^i + \ldots + (1/2)^k] = \]

\[ = cN \times \text{constant} = \Theta(N) \]
Recursion Tree for: $T(n) = 2T(n/2) + c$

Base case: $T(1) = c$

Stop at level $k$, when the subtree is $T(1)$.

$\Rightarrow$ The problem size is 1, but the general formula for the problem size, at level $k$ is: $n/2^k = 1$ $\Rightarrow$ $k = \log_2 n$

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: \( T(1) = c \)

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: \( n/2^k \Rightarrow n/2^k = 1 \Rightarrow k = \log n \)

<table>
<thead>
<tr>
<th>Level</th>
<th>Arg/ pb size</th>
<th>cost of 1 node</th>
<th>Nodes per level</th>
<th>Level cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( n )</td>
<td>8</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>( n/2 )</td>
<td>8</td>
<td>2</td>
<td>2*8</td>
</tr>
<tr>
<td>2</td>
<td>( n/4 )</td>
<td>8</td>
<td>4</td>
<td>4*8</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( i )</td>
<td>( n/2^i )</td>
<td>8</td>
<td>( 2^i )</td>
<td>( 2^i*8 )</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( k = \log n )</td>
<td>( =n/2^k )</td>
<td>8</td>
<td>( 2^k )</td>
<td>( 2^k*8 )</td>
</tr>
</tbody>
</table>

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:** \( T(1) = c \)

**Table:**

<table>
<thead>
<tr>
<th>Level</th>
<th>Arg/pb size</th>
<th>cost of 1 node</th>
<th>Nodes per level</th>
<th>Level cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( n )</td>
<td>( c*n )</td>
<td>1</td>
<td>( c*n )</td>
</tr>
<tr>
<td>1</td>
<td>( n/2 )</td>
<td>( c*n/2 )</td>
<td>2</td>
<td>2( c<em>n/2 ) = ( c</em>n )</td>
</tr>
<tr>
<td>2</td>
<td>( n/4 )</td>
<td>( c*n/4 )</td>
<td>4</td>
<td>4( c<em>n/4 ) = ( c</em>n )</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( i )</td>
<td>( n/2^i )</td>
<td>( c*n/2^i )</td>
<td>( 2^i )</td>
<td>( 2^i<em>c</em>n/2^i ) = ( c*n )</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( k = \lg n )</td>
<td>( 1 ) ((=n/2^k))</td>
<td>( c=c<em>1=c</em>n/2^k )</td>
<td>( 2^k ) ((=n))</td>
<td>( 2^k<em>c</em>n/2^k ) = ( c*n )</td>
</tr>
</tbody>
</table>

**Tree cost:**

\[
T(n) = cn * (k + 1) = cn * (1 + \lg n) = cn \lg n + cn = \Theta(n \lg n)
\]
Recursion Tree for \( T(n) = 3T(n/2) + cn \)

Base case: \( T(1) = c \)

<table>
<thead>
<tr>
<th>Level</th>
<th>Arg/ pb size</th>
<th>cost of 1 node</th>
<th>Nodes per level</th>
<th>Level cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>n</td>
<td>( c*n )</td>
<td>1</td>
<td>( c*n )</td>
</tr>
<tr>
<td>1</td>
<td>( n/2 )</td>
<td>( c*n/2 )</td>
<td>( 3 )</td>
<td>( 3<em>c</em>n/2 ) ( = (3/2)<em>c</em>n )</td>
</tr>
<tr>
<td>2</td>
<td>( n/4 )</td>
<td>( c*n/4 )</td>
<td>( 9 )</td>
<td>( (3/2)^2<em>c</em>n )</td>
</tr>
<tr>
<td>( \cdots )</td>
<td>( \cdots )</td>
<td>( \cdots )</td>
<td>( \cdots )</td>
<td>( \cdots )</td>
</tr>
<tr>
<td>( i )</td>
<td>( n/2^i )</td>
<td>( c*n/2^i )</td>
<td>( 3^i )</td>
<td>( (3/2)^i<em>c</em>n )</td>
</tr>
<tr>
<td>( \cdots )</td>
<td>( \cdots )</td>
<td>( \cdots )</td>
<td>( \cdots )</td>
<td>( \cdots )</td>
</tr>
<tr>
<td>( k=\lceil \log n \rceil )</td>
<td>1 ( (=n/2^k) )</td>
<td>( c=c<em>1=c</em>n/2^k )</td>
<td>( 3^k ) ( (\neq n) )</td>
<td>( (3/2)^k<em>c</em>n )</td>
</tr>
</tbody>
</table>

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:
\[ n/2^k \Rightarrow n/2^k = 1 \Rightarrow k = \lceil \log n \rceil \]
Total Tree Cost for $T(n) = 3T(n/2) + cn$

Closed form

$T(n) = cn + (3/2)cn + (3/2)^2 cn + ... (3/2)^i cn + ... (3/2)^{\lfloor \log n \rfloor} cn =
= cn \cdot [1 + (3/2) + (3/2)^2 + ... + (3/2)^{\lfloor \log n \rfloor}] = cn \sum_{i=0}^{\lfloor \log n \rfloor} (3/2)^i =
= cn \cdot \frac{(3/2)^{\lfloor \log n \rfloor + 1} - 1}{(3/2) - 1} = 2cn[(3/2) \cdot (3/2)^{\lfloor \log n \rfloor} - 1] = 3cn \cdot (3/2)^{\lfloor \log n \rfloor} - 2cn$

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

$= 3cn \cdot n^{\lfloor \log 3 \rfloor - 1} - 2cn = 3cn^{1+\lfloor \log 3 \rfloor - 1} - 2cn = 3cn^{\lfloor \log 3 \rfloor} - 2cn = \Theta(n^{\lfloor \log 3 \rfloor})$

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

$... = cn \cdot \frac{(3/2)^{\lfloor \log n \rfloor + 1} - 1}{(3/2) - 1} = \Theta(n \cdot (3/2) \cdot (3/2)^{\lfloor \log n \rfloor + 1}) = \Theta(n \cdot (3/2)^{\lfloor \log n \rfloor})$

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

$= \Theta(n \cdot n^{\lfloor \log 3 \rfloor - 1}) = \Theta(n^{\lfloor \log 3 \rfloor})$
Recursion Tree for: $T(n) = 2T(n/5) + cn$

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

• $T(n) = 7T(n/3) + cn$

• $T(n) = 7T(n/3) + c n^5$
  – 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: \( T(1) = d \neq c \).
  – At most, it affects the constant for the dominant term.

<table>
<thead>
<tr>
<th>Level</th>
<th>Arg/ pb size</th>
<th>Cost of 1 node</th>
<th>Nodes per level</th>
<th>Level cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( n )</td>
<td>( c*n )</td>
<td>1</td>
<td>( c*n )</td>
</tr>
<tr>
<td>1</td>
<td>( n/2 )</td>
<td>( c*n/2 )</td>
<td>2</td>
<td>( 2<em>c</em>n/2 ) = ( c*n )</td>
</tr>
<tr>
<td>2</td>
<td>( n/4 )</td>
<td>( c*n/4 )</td>
<td>4</td>
<td>( 4<em>c</em>n/4 ) = ( c*n )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>( i )</td>
<td>( n/2^i )</td>
<td>( c*n/2^i )</td>
<td>( 2^i )</td>
<td>( 2^i<em>c</em>n/2^i ) = ( c*n )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>( k = \log n )</td>
<td>1 (=( n/2^k ))</td>
<td>2( k ) (=( n ))</td>
<td>( d*n )</td>
<td></td>
</tr>
</tbody>
</table>

Tree cost:
\[
= cnk + dn = cn \log n + dn = \Theta(n \log n)
\]
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 $\log(N)$ 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.