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 $T(n) = 3T(n/8) + 4n^{2.5}\log n$,
    
    $T(n/8) =$ ........................................
    $T(n-1) =$ ........................................

- **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.

  – 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
Recurrence - Recursion Tree Relationship

\[ T(1) = c \]

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

- 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: $T(1) = c$

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: $n/2^p = 1$ => $p = \log n$

Tree TC $= c(1+2+2^2+2^3+...+2^i+...+2^p) = c2^{p+1}/(2-1)$

$$= 2c2^p = 2cn = \Theta(n)$$
Recursion Tree for: \( T(n) = 2T(n/2) + 8 \)

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

Base case: \( T(1) = 8 \)

Stop at level \( p \), when the subtree is \( T(1) \).

\( \Rightarrow \) The problem size is 1, but the general formula for the problem size, at level \( p \) is:

\[ n/2^p = 1 \Rightarrow 2^p = n \Rightarrow p = \log n \]

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

Tree TC = \( c(1+2+2^2+2^3+\ldots+2^i+\ldots+2^p)=8 \times 2^{p+1}/(2-1) \)

= \( 2 \times 8 \times 2^p = 16n = \Theta(n) \)
Recursion Tree for: \( T(n) = 2T(n/2)+cn \)

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

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

Stop at level \( p \), when the subtree is \( T(1) \).

\( \Rightarrow \) The problem size is 1, but the general formula for the problem size, at level \( p \) is:

\[ n/2^p \Rightarrow n/2^p = 1 \Rightarrow 2^p = n \Rightarrow p = \log n \]

**Tree TC** \( = cn(p + 1) = cn(1 + \log n) = cn\log n + cn = \theta(n \log n) \)
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:

$n/2^p \Rightarrow n/2^p = 1 \Rightarrow 2^p = n \Rightarrow p = \log n$
Total Tree TC for $T(n) = 3T(n/2) + cn$

**Closed form**

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

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

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

**use:** $c^{\lg n} = n^{\lg c} \Rightarrow (3/2)^{\lg n} = n^{\lg(3/2)} = n^{\lg3 - \lg2} = n^{\lg3 - 1} \Rightarrow$

$= 3cn \times n^{\lg3 - 1} - 2cn = 3cn^{1+\lg3 - 1} - 2cn = 3cn^{\lg3} - 2cn = \Theta(n^{\lg3})$

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

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

**use:** $c^{\lg n} = n^{\lg c} \Rightarrow (3/2)^{\lg n} = n^{\lg(3/2)} = n^{\lg3 - \lg2} = n^{\lg3 - 1} \Rightarrow$

$= \Theta(n \times n^{\lg3 - 1}) = \Theta(n^{\lg3})$
Recursion Tree for: $T(n) = 2T(n/5) + cn$

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: $n/5^p \Rightarrow n/5^p = 1 \Rightarrow 5^p = n \Rightarrow p = \log_5 n$

Tree TC
(derivation similar to TC for $T(n) = 3T(n/2) + cn$)
Total Tree TC for $T(n) = 2T(n/5) + cn$

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

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

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

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

Also

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

$\Rightarrow T(n) = \Theta(n)$
Code => Recurrence

```c
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(\_\_) = \_\_

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.

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

```c
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)
```
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 (Base Case(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, \(N\leq20\), and one return, \(\text{return 500}\))

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 \(2*\text{foo(N/3)}\) vs \(\text{foo(N/3)} + \text{foo(N/3)}\)
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 \( \Theta(1) \) in best case

Worst case: \( T(N) = T(N/2) + c \) => \( T(N) = \Theta(\log(N)) \) => search is \( \Theta(\log(N)) \) in worst case

\[\Rightarrow\] We will report in general: search is \( O(\log(N)) \)
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 $0 \leq N \leq 4$ \hspace{1cm} (BC)

Recursive case(s):
$T(N) = T(N/5)+c$ for all $N>4$ that are multiples of 5 \hspace{1cm} (RC1)
$T(N) = 2\times T(N-4) + c$ for all other $N$ \hspace{1cm} (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 =>
$T_{\text{lower}}(N) = T(N/5)+c = \Theta(\log_5 N)$ , (and $T(N) \geq T_{\text{lower}}(N)$) => $T(N) = \Omega(\log_5 N)$

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 =>
$T_{\text{upper}}(N) = 2\times T(N-4)+c = \Theta(2^{N/4})$ , (and $T(N) \leq T_{\text{upper}}(N)=\Theta(2^{N/4})$ ) => $T(N) = O(2^{N/4})$

We have $\Omega$ and $O$ for $T(N)$, but we cannot compute $\Theta$ for it.
Recurrence => Code

Answers

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

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

Example 2 : Solve
\[ T(n) = T(n/3) + T(2n/3) + O(n) \]
Recurrence - Recursion Tree Relationship

\[ T(1) = c \]

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

- 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 \( n/5 \)  
\[ T(1) = c, \ T(0) = c \]

<table>
<thead>
<tr>
<th>Level</th>
<th>Arg/ pb size</th>
<th>TC of 1 node</th>
<th>Nodes per level</th>
<th>Level TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( i )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( p= )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 \( n/5 \)

\[ T(1) = c, \quad T(0) = c \]

Each internal node has 7 children

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:

\[ n/5^p = n/5^p = 1 \Rightarrow p = \log_5 n \]

Where we used:

\[ 7^i \left( \frac{n}{5} \right)^3 = 7^i n^3 \left( \frac{1}{5^i} \right)^3 = 7^i n^3 \left( \frac{1}{5^3} \right)^i = n^3 \left( \frac{7}{5^3} \right)^i \]

Tree TC: \( T(n) = \sum_{i=0}^{\log_5 n} cn^3 \left( \frac{7}{5^3} \right)^i = cn^3 \sum_{i=0}^{\log_5 n} \left( \frac{7}{5^3} \right)^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 \( T(n) = \Omega(n^3) \Rightarrow T(n) = \Theta(n^3) \)
\[ T(n) = 7T\left(\frac{n}{5}\right) + cn^3, \text{ if } n \text{ is not a multiple of 5, use round down for } n/5 \]
\[ T(1) = c, \quad T(0) = c \]
\[ T(n) = 5T(n-6)+c \]

\[ T(n) = c \text{ for all } 0 \leq n \leq 5 \quad \text{(i.e. } T(0)=T(1)=T(2)=T(3)=T(4)=T(5)=c \text{ )} \]

Assume \( n \) is a multiple of 6

Each internal
node has 5
children

\[
\begin{array}{cccc}
\text{Level} & \text{Arg/} & \text{TC of 1} & \text{Nodes} \\
0 & n & c & 1 \\
1 & n-6 & c & 5 \\
2 & n-2*6 & c & 5^2 \\
\vdots & \vdots & \vdots & \vdots \\
i & n-6i & c & 5^i \\
\end{array}
\]

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

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

\[
\begin{array}{cccc}
\text{Level TC} & \text{Nodes per level} & \text{TC of 1 node} & \text{Arg/} \\
0 & c & n & \text{TC} \\
1 & c & n-6 & \text{TC} \\
2 & c & n-2*6 & \text{TC} \\
\vdots & \vdots & \vdots & \vdots \\
i & c & n-6i & \text{TC} \\
\end{array}
\]
• 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: \( (1/3) \times p_{TC} + (2/3) \times p_{TC} \))

  \[ \Rightarrow \text{Total TC for those: } cn \times \log_3 n = \Theta(n \log n) \]

  - The number of incomplete levels should also be a multiple of \( \log n \) and the TC for each of those levels will be less than \( cn \)

  \[ \Rightarrow \text{Guess that } T(n) = O(n \log n) \]

- 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 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 \( \Theta(n) \) 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).
Common Recurrences Review

1. *Halve* problem in **constant** time:
   \[ T(n) = T(n/2) + c \quad \Theta(\ lg(n) ) \]

2. *Halve* problem in **linear** time:
   \[ T(n) = T(n/2) + n \quad \Theta(n) \quad (~2n) \]

3. Break (and put back together) the problem into 2 *halves* in **constant** time:
   \[ T(n) = 2T(n/2) + c \quad \Theta(n) \quad (~2n) \]

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

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

6. Reduce the problem size by 1 in **linear** time:
   \[ T(n) = T(n-1) + n \quad \Theta(\ n^2 ) \]
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 $\varepsilon$ in the discussion. Where does that $\varepsilon$ come from? Can you do math derivations that start from the formulation of the relevant case of the Theorem and result in the $\varepsilon$ 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 $\frac{f(n)}{n^{\log_b a}} = \frac{n/\log n}{n^{\log_b 2}} = \frac{n}{n \log n} = \frac{1}{\log n}$. It is clear that $\frac{1}{\log n} < n^\varepsilon$ for any constant $\varepsilon > 0$. 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: \( T(n) = 2T\left(\lfloor n / 2 \rfloor\right) + n \)
   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 \( T(n) = O(\lg n) \), using the definition: find \( c \) and \( n_0 \) s.t. \( T(n) \leq c \cdot n \cdot \lg n \) (here: \( f(n) = T(n) \), \( g(n) = n \cdot \lg n \))

Show with induction: \( T(n) \leq c \cdot n \cdot \lg n \) (for some \( c > 0 \))

Base case (boundary): Assume \( T(1) = 1 \)

Find \( n_0 \) s.t. the induction holds for all \( n \geq n_0 \).

For \( n=1 \):
\[
T(1) = 2T(\frac{n}{2}) + n = 2(1) + 1 = 3
\]

Want \( T(1) = 3 \leq c \cdot 1 \cdot \lg 1 = c \cdot 0 = 0 \) FALSE. \( \Rightarrow n_0 = 2 \) cannot be 1.

For \( n=2 \):
\[
T(2) = 2T(1) + 2 = 2 + 2 = 4
\]

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

For \( n=3 \):
\[
T(3) = 2T(1) + 3 = 2 + 3 = 5
\]

Want \( T(3) = 5 \leq c \cdot 3 \cdot \lg 3 \), True for: \( c \geq 2 \)

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

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

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

Show \( T(n) = O(n) \) using the definition: find \( c \) and \( n_0 \) s.t. \( T(n) \leq c \cdot n \).
(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 \( n \):

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

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

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

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

want:

\[ \leq cn - d \Rightarrow \\
1 - d \leq 0 \Rightarrow d \geq 1 \]
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):
  \[ T(n) = 3T\left(\left\lfloor n/4 \right\rfloor \right) + \Theta(n^2) \]
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) ?**

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

- \( T(1) = c \)
- \( T(0) = c \)

Levels: \( N \)

Each node has TC \( c \) =>

\[ T(N) = cN = \Theta(N) \]
\[ T(N) = T(N-4) + c \]

```
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 \( \text{fact4}(N) \) ? \( T(N) = \ldots \)

\[
\begin{align*}
T(N) &= T(N-4) + c \\
T(3) &= c \\
T(2) &= c \\
T(1) &= c \\
T(0) &= c
\end{align*}
\]

Levels: \( \approx N/4 \)
Each node has \( T \) \( c \) =>
\[ T(N) = c \times N/4 = \Theta(N) \]
\[ T(N) = T(N-1) + cN \]

**selection_sort_rec(N)**

```c
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 TC \( c(N-i) \) =>

\[ T(N) = cN+c(N-1)+...ci+...c = cN(N+1)/2 = \Theta(N^2) \]
\[ T(N) = T(N/2) + c \]

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

Each node has TC \( c \) =>

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

**Time complexity tree:**

- \( T(N) \)
- \( cN \)
- \( T(N/2) \)
- \( cN/2 \)
- \( 2c \)
- \( T(2) \)
- \( c \)
- \( T(1) \)

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

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

\[
T(N) = c(N + N/2 + N/2^2 + \ldots N/2^i + \ldots + N/2^k) =
\]
\[
= cN(1 + 1/2 + 1/2^2 + \ldots 1/2^i + \ldots + 1/2^k) =
\]
\[
= cN[1 + (1/2) + (½)^2 + \ldots (½)^i + \ldots + (½)^p] =
\]
\[
= cN \times \text{constant} = \Theta(N)
\]
Recursion Tree for:  \( T(n) = 2T(n/2)+c \)

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

Stop at level \( p \), when the subtree is \( T(1) \).

\[ \text{Tree TC} = c(1+2+2^2+2^3+\ldots+2^i+\ldots+2^p) = c2^{p+1}/(2-1) = 2c2^p = 2cn = \Theta(n) \]
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 | \( 2^i \) | \( 2^i*8 \)
... | ... | ... | ... | ...
\( k=\log n \) | \( 1 \) (=\( n/2^k \)) | 8 | \( 2^k \) (=\( n \)) | \( 2^k*8 \)

Stop at level \( p \), when the subtree is \( T(1) \).

\( \Rightarrow \) The problem size is 1, but the general formula for the problem size, at level \( p \) is:
\( n/2^p \Rightarrow n/2^p = 1 \Rightarrow 2^p = n \Rightarrow p = \log n \)

Tree TC = \( c(1+2+2^2+2^3+...+2^i+...+2^p) = 8*2^{p+1}/(2-1) \)

\( = 2*8*2^p = 16n = \Theta(n) \)
Recursion Tree for:  \( T(n) = 2T(n/2)+cn \)

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

<table>
<thead>
<tr>
<th>Level</th>
<th>Arg/ pb size</th>
<th>TC of 1 node</th>
<th>Nodes per level</th>
<th>Level TC</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 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( =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 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( =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 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( =c*n )</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p=( \log n )</td>
<td>1 (( =n/2^p ))</td>
<td>( c=c<em>1=c</em>n/2^p )</td>
<td>( 2^p ) (( =n ))</td>
<td>( 2^p<em>c</em>n/2^p )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( =c*n )</td>
</tr>
</tbody>
</table>

Stop at level \( p \), when the subtree is \( T(1) \).

\( \Rightarrow \) The problem size is 1, but the general formula for the problem size, at level \( p \) is: \( n/2^p \Rightarrow n/2^p=1 \Rightarrow 2^p=n \Rightarrow p=\log n \)

Tree TC \( = cn(p + 1) = cn(1 + \log n) = cn\log n + cn = \theta(n\log n) \)
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:

$$\frac{n}{2^p} = 1 \Rightarrow 2^p = n \Rightarrow p = \log n$$
Total Tree TC for $T(n) = 3T(n/2) + cn$

Closed form

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

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

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

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

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

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

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

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

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

Stop at level \( p \), when the subtree is \( T(1) \).

\( \Rightarrow \) The problem size is 1, but the general formula for the problem size, at level \( p \) is:

\( n/5^p \Rightarrow n/5^p = 1 \Rightarrow 5^p = n \Rightarrow p = \log_5 n \)

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

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

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

• $T(n) = 7T(n/3) + cn$

• $T(n) = 7T(n/3) + cn^5$
  – 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.
• 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>TC of 1 node</th>
<th>Nodes per level</th>
<th>Level TC</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</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>=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</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>=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</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>=c*n</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p=\lg n</td>
<td>1 (=n/2^p)</td>
<td>2^p (=n)</td>
<td></td>
<td>=d*n</td>
</tr>
</tbody>
</table>

Tree TC = \( cnp + dn = cn\lg n + dn = \theta(n\lg n) \)
Permutations without repetitions
(Harder Example)

• Covering this material is subject to time availability

• Time complexity
  – Tree, intuition (for moving the local 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 $\log(N)$ 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.