Summation: Guess and Test

Technique 1: Guess the solution and use induction to test.

Technique 1a: Guess the form of the solution, and use simultaneous equations to generate constants. Finally, use induction to test.

Summation Example

Let $S(n) = \sum_{i=0}^{n} i^2$.

Guess that $S(n)$ is a polynomial $\leq n^3$.
Equivalently, guess that it has the form $S(n) = an^3 + bn^2 + cn + d$.

For $n = 0$ we have $S(0) = 0$ so $d = 0$.
For $n = 1$ we have $a + b + c + 0 = 1$.
For $n = 2$ we have $8a + 4b + 2c = 5$.
For $n = 3$ we have $27a + 9b + 3c = 14$.
Solving these equations yields $a = \frac{1}{3}, b = \frac{1}{2}, c = \frac{1}{6}$.
Now, prove the solution with induction.

Technique 2: Shifted Sums

Given a sum of many terms, shift and subtract to eliminate intermediate terms.

$$G(n) = \sum_{i=0}^{n} ar^i = a + ar + ar^2 + \cdots + ar^n$$
Shift by multiplying by $r$.

$$rG(n) = ar + ar^2 + \cdots + ar^n + ar^{n+1}$$
Subtract.

$$G(n) - rG(n) = G(n)(1 - r) = a - ar^{n+1}$$
$$G(n) = \frac{a - ar^{n+1}}{1 - r} \quad r \neq 1$$

Example 3.3

$$G(n) = \sum_{i=1}^{n} 2i = 1 \cdot 2 + 2 \cdot 2^2 + 3 \cdot 2^3 + \cdots + n \cdot 2^n$$
Multiply by 2.

$$2G(n) = 1 \cdot 2^2 + 2 \cdot 2^3 + 3 \cdot 2^4 + \cdots + n \cdot 2^{n+1}$$
Subtract (Note: $\sum_{i=1}^{n} 2^i = 2^{n+1} - 2$)

$$2G(n) - G(n) = n2^{n+1} - 2^n \cdots 2^2 - 2$$
$$G(n) = n2^{n+1} - 2^{n+1} + 2$$
$$= (n - 1)2^{n+1} + 2$$
Recurrence Relations

- A (math) function defined in terms of itself.
- Example: Fibonacci numbers:
  \[ F(n) = F(n-1) + F(n-2) \] general case
  \[ F(1) = F(2) = 1 \] base cases
- There are always one or more general cases and one or more base cases.
- We will use recurrences for time complexity of recursive (computer) functions.
- General format is \( T(n) = E(T, n) \) where \( E(T, n) \) is an expression in \( T \) and \( n \).
  - \( T(n) = 2T(n/2) + n \)
  - Alternately, an upper bound: \( T(n) \leq E(T, n) \).

Solving Recurrences

We would like to find a closed form solution for \( T(n) \) such that:

\[ T(n) = \Theta(f(n)) \]

Alternatively, find lower bound
- Not possible for inequalities of form \( T(n) \leq E(T, n) \).

Methods:
- Guess (and test) a solution
- Expand recurrence
- Theorems

Guessing

\[ T(n) = 2T(n/2) + 5n^2 \quad n \geq 2 \]
\[ T(1) = 7 \]
Note that \( T \) is defined only for powers of 2.

Guess a solution: \( T(n) \leq c_1 n^2 = f(n) \)
\[ T(1) = 7 \] implies that \( c_1 \geq 7 \)

Inductively, assume \( T(n/2) \leq f(n/2) \).

\[ T(n) = 2T(n/2) + 5n^2 \leq 2c_1(n/2)^2 + 5n^2 \leq c_1(n^2/4) + 5n^2 \leq c_1 n^2 \text{ if } c_1 \geq 20/3. \]

Guessing (cont)

Therefore, if \( c_1 = 7 \), a proof by induction yields:
\[ T(n) \leq 7n^2 \]
\[ T(n) \in \Theta(n^2) \]
Is this the best possible solution?
Spring 2014 42 / 51

Guessing (cont)

Guess again.

\[ T(n) \leq c_2 n^2 = g(n) \]

\[ T(1) = 7 \] implies \( c_2 \geq 7. \)

Inductively, assume \( T(n/2) \leq g(n/2). \)

\[ T(n) = 2T(n/2) + 5n^2 \]
\[ \leq 2c_2(n/2)^2 + 5n^2 \]
\[ = c_2(n^2/2) + 5n^2 \]
\[ \leq c_2 n^2 \text{ if } c_2 \geq 10 \]

Therefore, if \( c_2 = 10, \) \( T(n) \leq 10n^2. \) \( T(n) = O(n^2). \)

Is this the best possible upper bound?

Guessing (cont)

Now, reshape the recurrence so that \( T \) is defined for all values of \( n. \)

\[ T(n) \leq 2T([n/2]) + 5n^2, \quad n \geq 2 \]

For arbitrary \( n, \) let \( 2^k - 1 < n \leq 2^k. \)

We have already shown that \( T(2^k) \leq 10(2^k)^2. \)

\[ T(n) \leq T(2^k) \leq 10(2^k)^2 \]
\[ = 10(2^k/n)^2 n^2 \leq 10(2^2) n^2 \]
\[ \leq 40n^2 \]

Hence, \( T(n) = O(n^2) \) for all values of \( n. \)

Typically, the bound for powers of two generalizes to all \( n. \)

Expanding Recurrences

Usually, start with equality version of recurrence.

\[ T(n) = 2T(n/2) + 5n^2 \]
\[ T(1) = 7 \]

Assume \( n \) is a power of \( 2; \) \( n = 2^k. \)

Expanding Recurrences (cont)

\[ T(n) = 2T(n/2) + 5n^2 \]
\[ = 2(2T(n/4) + 5(n/2)^2) + 5n^2 \]
\[ = 2(2(2T(n/8) + 5(n/4)^2) + 5(n/2)^2) + 5n^2 \]
\[ = 2^k T(1) + 2^{k-1} \cdot 5(n/2^{k-1})^2 + 2^{k-2} \cdot 5(n/2^{k-2})^2 \]
\[ \cdots + 2 \cdot 5(n/2)^2 + 5n^2 \]
\[ = 7n + 5 \sum_{i=0}^{k-1} n^2/2^i = 7n + 5n^2 \sum_{i=0}^{k-1} 1/2^i \]
\[ = 7n + 5n^2 (2 - 1/2^{k-1}) \]
\[ = 7n + 5n^2(2 - 2/n). \]

This is the exact solution for powers of \( 2. \) \( T(n) = \Theta(n^2). \)
Divide and Conquer Recurrences

These have the form:

\[ T(n) = aT(n/b) + cn^k \]

\[ T(1) = c \]

... where \( a, b, c, k \) are constants.

A problem of size \( n \) is divided into \( k \) subproblems of size \( n/b \), while \( cn^k \) is the amount of work needed to combine the solutions.

Divide and Conquer Recurrences (cont)

Expand the sum; \( n = b^m \).

\[ T(n) = a(aT(n/b^2) + c(n/b)^k) + cn^k \]

\[ = a^m T(1) + a^{m-1}c(n/b^{m-1})^k + \cdots + ac(n/b)^k + cn^k \]

\[ = ca^m \sum_{i=0}^{m} (b^k/a)^i \]

\[ a^m = a^{\log_b n} = n^{\log_b a} \]

The summation is a geometric series whose sum depends on the ratio \( r = b^k/a \).

There are 3 cases.

D & C Recurrences (cont)

1. \( r < 1 \).

\[ \sum_{i=0}^{m} r^i < 1/(1 - r), \quad \text{a constant.} \]

\[ T(n) = \Theta(a^m) = \Theta(n^{\log_b a}). \]

2. \( r = 1 \).

\[ \sum_{i=0}^{m} r^i = m + 1 = \log_b n + 1 \]

\[ T(n) = \Theta(n^{\log_b a} \log n) = \Theta(n^k \log n) \]

D & C Recurrences (Case 3)

3. \( r > 1 \).

\[ \sum_{i=0}^{m} r^i = \frac{r^{m+1} - 1}{r - 1} = \Theta(r^m) \]

So, from \( T(n) = ca^m \sum r^i \),

\[ T(n) = \Theta(a^m r^m) \]

\[ = \Theta(a^m (b^k/a)^m) \]

\[ = \Theta(b^{km}) \]

\[ = \Theta(n^k) \]
Summary

Theorem 3.4:

\[ T(n) = \begin{cases} \Theta(n^{\log_2 a}) & \text{if } a > b^k \\ \Theta(n^k \log n) & \text{if } a = b^k \\ \Theta(n^k) & \text{if } a < b^k \end{cases} \]

Apply the theorem:

\[ T(n) = 3T(n/5) + 8n^2. \]

\[ a = 3, b = 5, c = 8, k = 2. \]

\[ b^k/a = 25/3. \]

Case (3) holds: \( T(n) = \Theta(n^2) \).

Examples

- Mergesort: \( T(n) = 2T(n/2) + n \).
  
  \[ 2^{\log 2} = 1, \text{ so } T(n) = \Theta(n \log n). \]

- Binary search: \( T(n) = T(n/2) + 1 \).
  
  \[ 2^{\log 1} = 1, \text{ so } T(n) = \Theta(n/2) \text{ work}. \]

- Insertion sort: \( T(n) = T(n - 1) + n \).
  
  Can’t apply the theorem. Sorry!

- Standard Matrix Multiply (recursively): \( T(n) = 8T(n/2) + n^2 \).

  \[ 2^{\log 1/2} = 1/2 \text{ so } T(n) = \Theta(n^{2 \log_2 1/2}) = \Theta(n^2). \]

Useful log Notation

- If you want to take the log of \( \log n \), it is written \( \log \log n \).
- \( \log n^2 \) can be written \( \log^2 n \).
- Don’t get these confused!
- \( \log^n \) means “the number of times that the log of \( n \) must be taken before \( n \leq 1 \).”
  - For example, \( 65536 = 2^{16} \) so \( \log \) 65536 = 4 since \( \log 65536 = 16, \log 16 = 4, \log 4 = 2, \log 2 = 1 \).

Amortized Analysis

Consider this variation on STACK:

```c
void init(STACK S); 
void examineTop(STACK S); 
void push(element x, STACK S); 
void pop(int k, STACK S); 
```

... where \( \text{pop} \) removes \( k \) entries from the stack.

“Local” worst case analysis for \( \text{pop} \):

\( O(n) \) for \( n \) elements on the stack.

Given \( m_1 \) calls to \( \text{push} \), \( m_2 \) calls to \( \text{pop} \):

Naive worst case: \( m_1 + m_2 \cdot n = m_1 + m_2 \cdot m_1 \).
Alternate Analysis

Use amortized analysis on multiple calls to push, pop:

Cannot pop more elements than get pushed onto the stack.

After many pushes, a single pop has high potential.

Once that potential has been expended, it is not available for future pop operations.

The cost for \( m_1 \) pushes and \( m_2 \) pops:

\[
m_1 + (m_2 + m_1) = O(m_1 + m_2)
\]

Actual number of (constant time) push calls + (Actual number of pop calls + Total potential for the pops)

CLR has an entire chapter on this – we won’t go into this much, but we use Amortized Analysis implicitly sometimes.