Master Theorem Solutions

Algorithm Analysis: foo(n), bar(n), and baz(n)

Algorithm 1: foo(n)

Code Analysis:

```
foo(n)
if (n == 1)
 return
for i = 1 to n
{
 for j = 1 to n
 {
 //some computations done here
 }
}
for i = 1 to 8
{
 foo(n/2)
}
```

Step 1: Determine f(n), a, and b

- **Work done before recursion (f(n)):** The nested loops `for i = 1 to n` and `for j = 1 to n` perform $O(n^2)$ operations
- **Number of recursive calls (a):** 8 calls to foo(n/2)
- **Size of each subproblem (n/b):** Each call processes n/2, so b = 2

Step 2: Write the recurrence relation $T(n) = 8T(n/2) + O(n^2)$

Step 3: Apply Master Theorem

- a = 8, b = 2
- $f(n) = O(n^2)$, so d = 2
- $\log_b(a) = \log_2(8) = 3$

Comparison: $f(n) = O(n^2)$ vs $n^{\log_b(a)} = n^3$

- Since $f(n) = O(n^{3-\varepsilon})$ where $\varepsilon = 1 > 0$

Master Theorem Case: Case 1 applies **Solution:** $T(n) = \Theta(n^3)$

Algorithm 2: bar(n)

Code Analysis:

```
bar(n)
if (n == 1)
 return
for i = 1 to n
{
 for j = i+1 to n
 {
 //some computations done here
 }
}
for k = 1 to 9
 bar(n/3)
```

Step 1: Determine f(n), a, and b

- **Work done before recursion (f(n)):** The nested loops where j runs from i+1 to n:
  - For i=1: j runs n-1 times
  - For i=2: j runs n-2 times
  - ...
  - For i=n-1: j runs 1 time
  - Total: $(n-1) + (n-2) + ... + 1 = n(n-1)/2 = O(n^2)$
- **Number of recursive calls (a):** 9 calls to bar(n/3)
- **Size of each subproblem (n/b):** Each call processes n/3, so b = 3

Step 2: Write the recurrence relation $T(n) = 9T(n/3) + O(n^2)$

Step 3: Apply Master Theorem

- a = 9, b = 3
- $f(n) = O(n^2)$, so d = 2
- $\log_b(a) = \log_3(9) = 2$

Comparison: $f(n) = O(n^2)$ vs $n^{\log_b(a)} = n^2$

- Since $f(n) = \Theta(n^2)$

Master Theorem Case: Case 2 applies **Solution:** $T(n) = \Theta(n^2 \lg n)$

Algorithm 3: baz(n)

Code Analysis:

```
baz(a, n) // a is an array of size n
if (n == 1)
 return
for i = 1 to n
{
 for j = 1 to i
 {
```

```
  for k = 1 to j
  {
  //some computations done here
  }
  }
 }
 m = n/2
 baz(a[1..m], n/2) // a[1..m] represents left half of array a
 baz(a[m+1..n], n/2) // a[m+1..n] represents right half of array a
```

Step 1: Determine f(n), a, and b

- **Work done before recursion (f(n)):** The triple nested loops:
  - For i=1: j runs 1 time, k runs 1 time → 1 operation
  - For i=2: j runs 1,2 times, k runs 1,1,2 times → 1+1+2 = 4 operations
  - For i=3: j runs 1,2,3 times, k runs 1,1,2,1,2,3 times → 1+2+3+3 = 9 operations
  - Total: $\Sigma_{i=1}^{n} \Sigma_{j=1}^{i} j = \Sigma_{i=1}^{n} i(i+1)/2 = O(n^3)$
- **Number of recursive calls (a):** 2 calls to baz(n/2)
- **Size of each subproblem (n/b):** Each call processes n/2, so b = 2

Step 2: Write the recurrence relation $T(n) = 2T(n/2) + O(n^3)$

Step 3: Apply Master Theorem

- a = 2, b = 2
- $f(n) = O(n^3)$, so d = 3
- $\log_b(a) = \log_2(2) = 1$

Comparison: $f(n) = O(n^3)$ vs $n^{\log_b(a)} = n^1 = n$

- Since $f(n) = \Omega(n^{1+\varepsilon})$ where $\varepsilon = 2 > 0$
- Need to verify regularity condition: $a \cdot f(n/b) \leq c \cdot f(n)$
- $2 \cdot (n/2)^3 \leq c \cdot n^3 \rightarrow 2 \cdot n^3/8 \leq c \cdot n^3 \rightarrow n^3/4 \leq c \cdot n^3$
- Choose c = 1/2, condition satisfied for large n

Master Theorem Case: Case 3 applies **Solution:** $T(n) = \Theta(n^3)$

Summary

Algorithm	Recurrence	Master Theorem Case	Time Complexity
foo(n)	$T(n) = 8T(n/2) + O(n^2)$	Case 1	$\Theta(n^3)$
bar(n)	$T(n) = 9T(n/3) + O(n^2)$	Case 2	$\Theta(n^2 \lg n)$
baz(n)	$T(n) = 2T(n/2) + O(n^3)$	Case 3	$\Theta(n^3)$

Course content developed by Declan Gray-Mullen for WNEU with Claude