Divide-and-Conquer Algorithms and The Master Theorem

Instructor: İsıl Dillig

Divide-and-Conquer Algorithms

▶ Divide-and-conquer algorithms are recursive algorithms that:
1. Divide problem into \( k \) smaller subproblems of the same form
2. Solve the subproblems
3. Conquer the original problem by combining solutions of subproblems

Example I: Binary Search

▶ Problem: Given sorted array of integers, is \( i \) in the array?
▶ Binary search algorithm:
1. Compare \( i \) with middle element \( m \) of array
2. If \( i > m \), then recursively search right half
3. Otherwise, recursively search left half
▶ Classic divide-and-conquer algorithm

Solving Recurrence for Binary Search

\[
T(n) = T\left(\frac{n}{2}\right) + 1 \quad T(1) = 1
\]
▶ Not in a form we can immediately solve, but can massage it!
▶ Let \( n = 2^k \): \( T(2^k) = T(2^{k-1}) + 1 \)
▶ Now, let \( \alpha_k = T(2^k) \): \( \alpha_k = \alpha_{k-1} + 1 \) \( \alpha_0 = 1 \)
▶ What’s the solution for this recurrence?
▶ Since \( n = 2^k \), this implies \( T(n) = \log_2 n + 1 \)
▶ Hence, complexity of binary search: \( \Theta(\log n) \)

Example II: Merge Sort

▶ Problem: Sort elements in array
▶ Merge sort solution:
1. Recursively sort left half of array
2. Recursively sort right half of array
3. Merge the two sorted arrays
How to Merge Two Sorted Arrays?

- **Input:** Two sorted arrays \( A_1, A_2 \)
- **Output:** New sorted array that includes all elements in \( A_1, A_2 \)
- **Idea:** Pointers to current elements in \( A_1, A_2 \) (initially first)
- **Copy smaller element to output array and advance pointer**
- If combined size of \( A_1, A_2 \) is \( n \), merging takes \( 4n \) steps

Recurrence Relation for Merge Sort

- What is worst-case complexity of Merge Sort?
- Let \( T(n) \) be \( \# \) operations performed to sort array of length \( n \)
- What is a recurrence relation for \( T(n) \)?
- As before, let \( n = 2^k \):

Solving Recurrence Relation

- \( a_k = 2 \cdot a_{k-1} + 4 \cdot 2^k \) \( a_0 = 1 \)
- **Particular solution form:**
- **Particular solution:**
- **Solution for homogeneous recurrence:**
- **Solve for \( \alpha \):** \( \alpha \cdot 2^0 + 0 \cdot 2^1 = 1 \Rightarrow \alpha = 1 \)
- **Solution:**
- **Plug in \( k = \log_2 n \):**
- **Hence, algorithm is \( \Theta(n \cdot \log n) \)**

The Master Theorem

Consider the recurrence \( T(n) = a \cdot T(\frac{n}{b}) + f(n) \) where \( a, c \geq 1 \), \( d \geq 0 \), and \( b > 1 \). Then:

1. **If \( c \cdot b^d \) \( \leq a \)**, then \( T(n) = \Theta(n^d) \)
2. **If \( c \cdot b^d \) \( = a \)**, then \( T(n) = \Theta(n \cdot \log(n)) \)
3. **If \( c \cdot b^d \) \( > a \)**, then \( T(n) = \Theta(n^{\log_b(a)}) \)

Revisiting Examples

- **Example 1:** Recurrence for binary search: \( T(n) = T(\frac{n}{2}) + 1 \)
  - Here, \( a = 1, b = 2, d = 0 \), Hence \( a = b^d \)
  - By Case 2 of Master Thm, \( T(n) = \Theta(n^{0} \log n) = \Theta(\log n) \)
- **Example 2:** Recurrence for merge sort: \( T(n) = 2 \cdot T(\frac{n}{2}) + 4n \)
  - Here, \( a = 2, b = 2, d = 1 \), Hence \( a = b^d \)
  - By Case 2 of Master Thm, \( T(n) = \Theta(n \cdot \log n) \)
Why is the Master Theorem True?

Consider the recurrence $T(n) = a \cdot T(\frac{n}{b}) + c \cdot n^d$

- At every level of recursion, # subproblems multiplied by $a$
- But size of subproblem divided by $b$

Total Cost

$T(n) = \Theta(n^{\log_b a}) + \sum_{i=0}^{\log_b n-1} a^i \cdot c \cdot (\frac{n}{b^i})^d$

Can be rewritten as:

$T(n) = \Theta(n^{\log_b a}) + \sum_{i=0}^{\log_b n-1} a^i \cdot c \cdot (\frac{n}{b^i})^d$

Proof of Master Theorem, cont.

- Total amount of work:

$T(n) = \Theta(n^{\log_b a}) + \sum_{i=0}^{\log_b n-1} c \cdot (\frac{n}{b^i})^d$
Proof of Master Theorem, cont.

\[ T(n) = \Theta(n^{\log_b a}) + \sum_{i=0}^{\log_b n - 1} c \cdot \left( \frac{a}{b^d} \right)^i \cdot n^d \]

- **Case 3:** \( a > b^d \). In this case, \( n^{\log_b a} > n^d \).
- Use closed formula for geometric series to expand summation:
  \[ c \cdot n^d \cdot \frac{1 - \left( \frac{a}{b^d} \right)^{\log_b n - 1}}{1 - \frac{a}{b^d}} \]
- This can be rewritten to \( c' \left( a^{\log_b n} - n^d \right) \) for some constant \( c' \)
- Since, \( a^{\log_b n} = n^{\log_b a} \), \( T(n) \) is \( \Theta(n^{\log_b a}) \)

Final Logistics

- Final exam: 9 am to noon on Tuesday, December 15.
- Final is cumulative!
- Allowed to bring up to 10 cheat sheets
- Good luck on finals and have a wonderful winter break!