Lectures 13/14: Solving Recurrences

COMS10007 - Algorithms

Dr. Christian Konrad

18.03.2019 and 19.03.2019

Divide-and-conquer Algorithms

Algorithmic Design Principle: Divide-and-conquer

1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem
2. **Conquer** the subproblems by solving them recursively (if subproblems have constant size, solve them *directly*)
3. **Combine** the solutions to the subproblems into the solution for the original problem

Examples

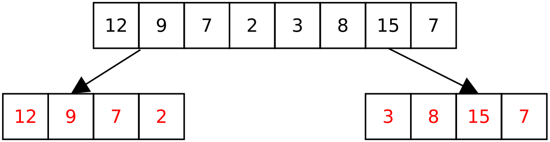
Quicksort, mergesort, maximum subarray algorithm, binary search, FAST-PEAK-FINDING, ...

Example: Merge sort

Recall: Merge Sort

Recall: Merge Sort

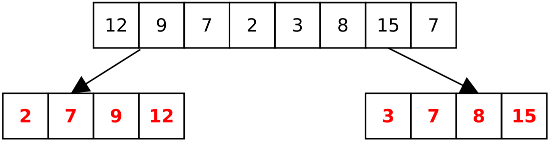
1. **Divide**
   Split input array A of length n into subarrays $A_1 = A[0, \lfloor n/2 \rfloor]$ and $A_2 = A[\lfloor n/2 \rfloor + 1, n-1]$



Example: Merge sort

Recall: Merge Sort

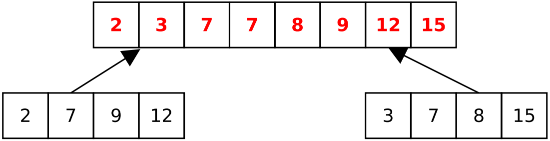
1. **Divide** $A \to A_1$ and A_2
2. **Conquer**
   Sort A_1 and A_2 recursively using the same algorithm



Example: Merge sort

Recall: Merge Sort

1. **Divide** $A \rightarrow A_1$ and A_2
2. **Conquer** Solve A_1 and A_2
3. **Combine**
   Combine sorted subarrays A_1 and A_2 and obtain sorted array A



Runtime: (assuming that n is a power of 2)

$$\begin{aligned} T(1) &= O(1) \\ T(n) &= 2T(n/2) + O(n) \end{aligned}$$

How to solve Recurrences?

Recurrences

- Divide-and-Conquer algorithms naturally lead to recurrences
- How can we *solve* them? Often only interested in asymptotic upper bounds

Methods for solving recurrences

- Substitution method
  guess solution, verify, induction
- Recursion-tree method (previously seen for merge sort and maximum subarray problem)
  may have plenty of awkward details, provides good guess that can be verified with substitution method
- Master theorem
  very powerful, cannot always be applied

The Substitution Method

1. Guess the form of the solution
2. Use mathematical induction to find the constants and show that the solution works
3. Method provides an upper bound on the recurrence

Example (suppose n is always a power of two)

$$\begin{aligned} T(1) &= O(1) \\ T(n) &= 2T(n/2) + O(n) \end{aligned}$$

The Substitution Method

1. Guess the form of the solution
2. Use mathematical induction to find the constants and show that the solution works
3. Method provides an upper bound on the recurrence

Example (suppose n is always a power of two)

$$\begin{aligned} T(1) &= c_1 \\ T(n) &= 2T(n/2) + c_2 n \end{aligned}$$

Eliminate O-notation in recurrence

The Substitution Method

1. Guess the form of the solution
2. Use mathematical induction to find the constants and show that the solution works
3. Method provides an upper bound on the recurrence

Example (suppose n is always a power of two)

$$\begin{aligned} T(1) &= c_1 \\ T(n) &= 2T(n/2) + c_2 n \end{aligned}$$

Eliminate O-notation in recurrence

Step 1. Guess good upper bound

$$T(n) \leq Cn \log n$$

The Substitution Method (2)

Step 2. Substitute into the Recurrence

- Assume that our guess $T(n) \le Cn \log n$ is correct for every $n' < n$
- Corresponds to induction step of a proof by induction

$$\begin{aligned} T(n) &= 2T(n/2) + c_2 n \ \le \ 2C\frac{n}{2}\log(\frac{n}{2}) + c_2 n \\ &= Cn\,(\log(n) - \log(2)) + c_2 n \\ &= Cn \log n - Cn + c_2 n \ \le \ Cn \log n \ , \end{aligned}$$

if we chose $C \ge c_2$. ✓

Verify the Base Case

$$T(1) \le C \cdot 1 \log(1) = 0 \not\ge c_1 \quad ✗$$

The base case is a problem...

The Substitution Method (3)

Recall: $T(1) = c_1$ and $T(n) = 2T(n/2) + c_2 n$
Our guess: $T(n) \leq Cn \log n$ (induction step holds for $C \geq c_2$)

Solution: Choose a different base case! $n = 2$

$$\begin{aligned} T(2) &= 2T(1) + 2c_2 = 2c_1 + 2c_2 = 2(c_2 + c_1) \\ C2 \log 2 &= 2C \end{aligned}$$

Hence, for every $C \geq c_2 + c_1$, our guess holds for $n = 2$:

$$T(2) \leq C2 \log 2 \ .$$

Result

- We proved $T(n) \leq Cn \log n$, for every $n \geq 2$, when choosing $C \geq c_1 + c_2$
- **Observe:** This implies $T(n) \in O(n \log n)$ (important)

Asymptotic notation allows us to chose arbitrary base-case

A Strange Problem

Example: Give an upper bound on the recurrence

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1 \end{aligned}$$

Step 1: Guess correct solution $T(n) \le f(n) := Cn$

Step 2: Verify the solution

$$T(n) \le C\lceil n/2 \rceil + C\lfloor n/2 \rfloor + 1 = Cn + 1 \not\le f(n) \text{ ✗}$$

- We need a different guess
- Let's try: $f_1(n) := Cn + 1$ and $f_2(n) := Cn - 1$

$$\begin{aligned} f_1 : T(n) &\le C\lceil n/2 \rceil + 1 + C\lfloor n/2 \rfloor + 1 + 1 = Cn + 3 \not\le f_1(n) \text{ ✗} \\ f_2 : T(n) &\le C\lceil n/2 \rceil - 1 + C\lfloor n/2 \rfloor - 1 + 1 = Cn - 1 = f_2(n) \checkmark \end{aligned}$$

(holds for every positive C)

A Strange Problem (2)

Verify Base Case for f_2

- We have: $T(1) = 1$ and $f_2(1) = C - 1 \geq T(1)$ for $C \geq 2$
- We thus set the constant C in f_2 to $C = 2$
- Then $f_2(n) = 2n - 1 \geq T(n)$ for every $n \geq 1$
- This implies that $T(n) \in O(n)$

Comments

- Guessing right can be difficult and requires experience
- However, recursion tree method can provide a good guess!

Recursion Tree:

- Each node represents cost of single subproblem
- Recursive invocations become children of a node

Example

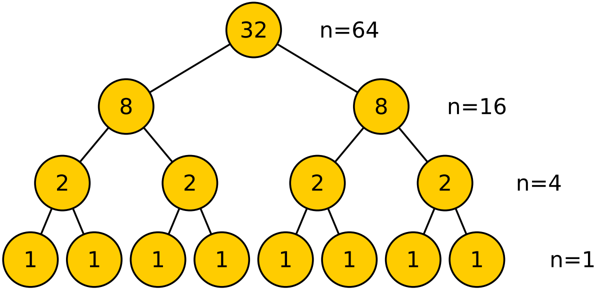
$$T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2$$

$$\begin{aligned} T(64) &= 2T(16) + 32 = 2(2T(4) + 8) + 32 \\ &= 2(2(2T(1) + 2) + 8) + 32 \\ &= 2(2(2 \cdot 1 + 2) + 8) + 32 = 64 \end{aligned}$$

Example

$$T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + \underbrace{n/2}_{\text{cost of subproblem}}$$

Recursion Tree for $n = 64$:

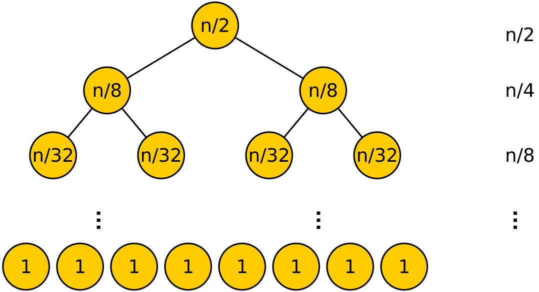


Sum of values assigned to nodes equals $T(64)$

Obtaining a Good Guess for Solution

$$T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2$$

Draw Recursion Tree for general n (Observe: we ignore $\lfloor . \rfloor$)



Sum of Nodes in Level i**:** $\frac{n}{2^l}$ (except the last level)

Obtaining a Good Guess for Solution (2)

Number of Levels: l

- We have $\frac{n}{4^{l-1}} \approx 1$
- $l = \log_4(n) + 1$

Cost on last Level: $=$ number of nodes on last level

$$\approx 2^{\log_4(n)} = 2^{\frac{\log n}{\log 4}} = 2^{\log(n)/2} = n^{\frac{1}{2}} = \sqrt{n} \ .$$

Our Guess:

$$\left(\sum_{i=1}^{\log_4(n)} \frac{n}{2^i} \right) + \sqrt{n} = \left(n \cdot \underbrace{\sum_{i=1}^{\log_4(n)} \frac{1}{2^i}}_{\text{geom. series}} \right) + \sqrt{n} = n \cdot O(1) + \sqrt{n} = O(n) \ .$$

Use substitution method to prove that guess is correct!

Verification via Substitution Method

$$T(1) = 1, \quad T(n) = 2T(\lfloor n/4 \rfloor) + n/2$$

Our Guess: $T(n) \leq c \cdot n$

Substitute into the Recurrence:

$$T(n) \;=\; 2T(\lfloor n/4 \rfloor) + n/2 \leq 2c\lfloor \frac{n}{4} \rfloor + \frac{n}{2} \leq n\frac{c+1}{2} \leq c \cdot n \;,$$

for every $c \geq 1$.

Verify the Base Case: $T(1) = 1 \leq c \cdot 1 = c$ for every $c \geq 1$.

Summary:

- We proved $T(n) \leq n$, for every $n \geq 1$
- Hence $T(n) \in O(n)$

Summary

Recursion Tree Method

- Assign contribution of subproblem to each node
- Sum up contributions using tree structure
- Allows us to be sloppy, since we only aim for a good guess
- Verify guess with subsitution method

Substitution Method

- Guess correct solution
- Verify guess using mathematical induction
- Guessing can be difficult