Lecture 2 Divide and Conquer I

Thursday, August 31, 2017 2:38 PM

- Merge Sort
  - how to analyze the running time?
  - usually: runtime of a d&c algorithm follows a recursion
  - Let $T(n)$ be the time it takes for merge-sort to sort n numbers.

MergeSort(a[])
1. IF Length(a) < 2 THEN RETURN a.
2. Partition a[] evenly into two arrays b[], c[].
3. b[] = MergeSort(b[])
4. c[] = MergeSort(c[])
5. RETURN Merge(b[], c[]).

$$\boxed{T(n) = \underbrace{T\left(\tfrac{n}{2}\right)}_{\text{step 3}} + \underbrace{T\left(\tfrac{n}{2}\right)}_{\text{step 4}} + \underbrace{\overset{\text{constant}}{A} \cdot n}_{\text{step 5}}}$$ recursion

$$T(1) = 1$$ base case.

- goal: solve the recursion
  $$T(n) = f(n)$$

- method 1: Guess and prove by induction. (better for rigorous proof)

Claim: For $n \geq 2$, $T(n) \leq (A+3) n \log_2 n$

$(T(n) = O(n \log n))$

Proof: By induction

Base Case: $n = 2, 3$, easy to show

Hypothesis: $T(n) \leq (A+3) n \log_2 n$ for $2 \leq n < k$

want: $T(k) \leq (A+3)\, k \log_2 k$

Proof: $T(k) = 2\,T\left(\frac{k}{2}\right) + A \cdot k$ (from recursion)

$$\leq \cancel{2} \cdot (A+3) \cdot \frac{k}{\cancel{2}} \cdot \log_2 \frac{k}{2} + A \cdot k \quad \text{(induction hypothesis)}$$

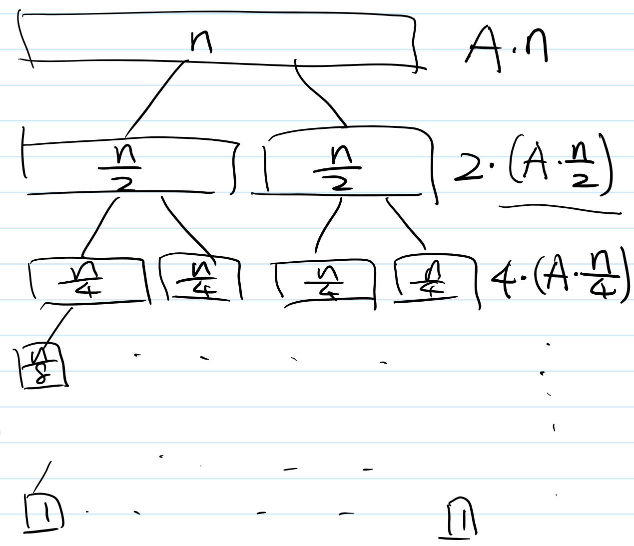
$$= (A+3) \cdot k \left(\log_2 k - 1\right) + A \cdot k$$

$$= (A+3) \cdot k \log_2 k - (A+3) \cdot k + A \cdot k$$

$$\leq (A+3) \cdot k \log_2 k$$

$\square$

- method: recursion tree.
  - expand recursion as a tree
  - count: time spent on merging for each layer.
  - take the sum of these costs.



$$T(n) = \sum_{i=1}^{\#\text{layers}} \text{merging cost for layer } i$$

$$= \sum_{i=1}^{\log_2 n} (\#\text{nodes in layer } i) \times (\text{cost per node})$$

$$= \sum_{i=1}^{\log_2 n} 2^{i-1} \times \left(A \frac{n}{2^{i-1}}\right)$$

$$= \sum_{i=1}^{\log_2 n} \underline{A \cdot n} = A \cdot n \cdot \log_2 n$$

$$T(n) = \sum_{i=1}^{\#\text{layers}} \text{merging cost} + \overbrace{\text{base cost} \times \#\text{base cases}}$$

$$1 \qquad\qquad n$$

How to get the formula for recursion tree method?

$$T(n) = 2T(\tfrac{n}{2}) + \underbrace{A \cdot n}_{\text{merge cost for layer 1}}$$

$$= 4T(\tfrac{n}{4}) + \underbrace{2 \cdot (A \cdot \tfrac{n}{2})}_{\text{merge cost for layer 2}} + A \cdot n \qquad \left(T(\tfrac{n}{2}) = 2T(\tfrac{n}{4}) + A \cdot \tfrac{n}{2}\right)$$

$$= 8T(\tfrac{n}{8}) + \underbrace{4 \cdot (A \cdot \tfrac{n}{4})}_{\text{merge cost for layer 3}} + 2 \cdot (A \cdot \tfrac{n}{2}) + A \cdot n \qquad \left(T(\tfrac{n}{4}) = 2T(\tfrac{n}{8}) + A \cdot \tfrac{n}{2}\right)$$

$$= \ldots \qquad (\text{repeat} \approx \log_2 n \text{ times})$$

$$= \underbrace{n \cdot T(1)}_{\text{base cost}} + \underbrace{\underbrace{\tfrac{n}{2}(A \cdot 2)}_{\text{merge cost for last layer}} + \underbrace{\tfrac{n}{4} \cdot (A \cdot 4)}_{\text{merge cost for 2nd to last}} + \cdots + 4 \cdot (A \cdot \tfrac{n}{4}) + 2 \cdot (A \cdot \tfrac{n}{2}) + A \cdot n}_{\sum_{i=1}^{\#\text{layers}} \text{merge cost at layer } i}$$

$$= n \cdot 1 + \underbrace{A \cdot n + A \cdot n + \cdots + A \cdot n + A \cdot n + A \cdot n}_{\log_2 n \text{ terms}}$$

$$= n + A n \log_2 n$$

can be ignored in asymptotic analysis.