COMPSCI 330 Lecture 2 Divide and Conquer

Tuesday, August 30, 2016 3:58 PM

- Divide and Conquer: General Idea
- Merge Sort
- Solving Recursions for Running Time and Memory
- Counting Inversions

- General Idea
  1. divide the problem into independent subproblem
  2. solve the subproblems recursively
  3. merge the solutions
- Design: how to divide and merge
- Analysis: compute recursive formula
- Example: Merge Sort

B: | 1 | 8 | 5 | 7 | C: | 2 | 6 | 4 | 3 |

↓ ↓

| 1 | 5 | 7 | 8 | | 2 | 3 | 4 | 6 |

1 2 3 4 5 6 7 8

```
merge sort (A)
  if len(A) ≤ 1 return(A)
  split A into B and C
  mergesort(B)
  merge sort(C)
  merge(B, C)
```

Time of merging procedure $O(n) \leq C \cdot n$

- Running Time
  1. find a recursive formula
  2. solve the recursion

Let $T(n)$ be the running time for array of length n

$$T(1) = 0$$

$$T(n) \leq T(\lfloor \tfrac{n}{2} \rfloor) + T(\lceil \tfrac{n}{2} \rceil) + Cn$$

for simplicity $\boxed{T(n) \leq 2T(\tfrac{n}{2}) + Cn}$

Theorem: $T(n) \leq C \cdot n \log_2 n$ (*)

Proof: when $n=1$ $T(n) = 0$ $C \cdot n \log_2 n = 0$ ✓

assume (*) is true for $n = 1, 2, 3, \ldots, k$

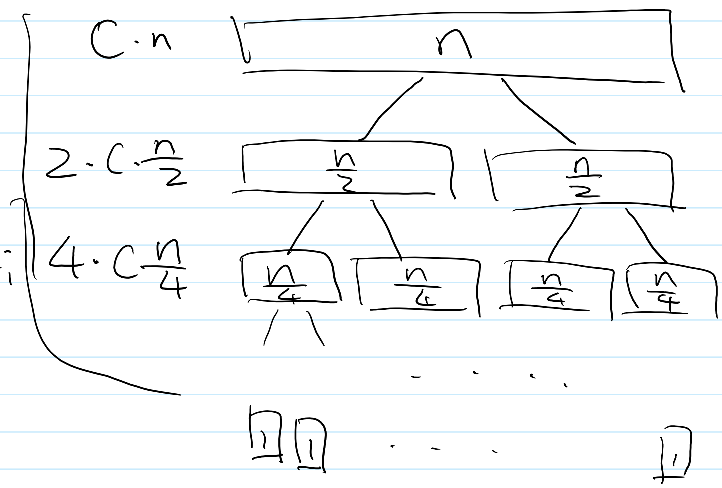
need to show (*) is still true when $n = k+1$

$$\begin{cases} T(k+1) \le 2T\left(\frac{k+1}{2}\right) + C(k+1) \\ T\left(\frac{k+1}{2}\right) \le C \cdot \frac{k+1}{2} \log_2 \frac{k+1}{2} \\ \qquad\qquad = C \cdot \frac{k+1}{2} \left(\log_2(k+1) - 1\right) \end{cases}$$

$$\Rightarrow T(k+1) \le \underbrace{2 \cdot C \cdot \frac{k+1}{2} \left(\log_2(k+1) - 1\right)}_{\text{cost of recursion}} + \underbrace{C(k+1)}_{\text{cost of merging}}$$

$$= C \cdot (k+1) \log_2(k+1) \qquad \square$$

- recursion tree
  - expand recursions as a tree
  - count: time spend on merging on each layer



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

$$= C \cdot n \cdot \#\text{layers}$$

$$= C \cdot n \log_2 n$$

- space
  - main difference: mergesort(B) mergesort(C) can use the same set memory

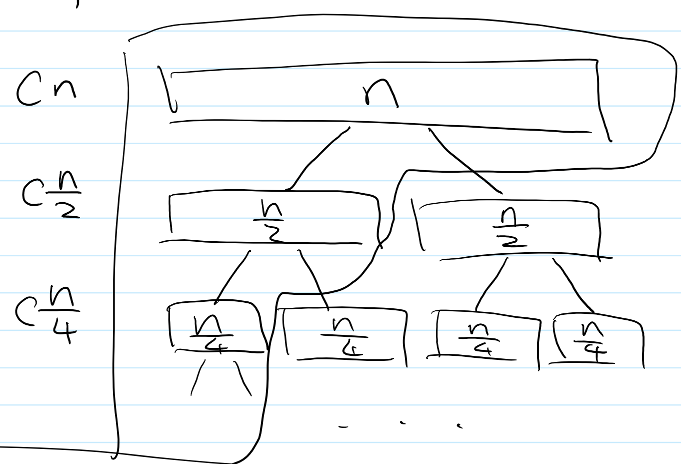
$$S(n) \le S\left(\frac{n}{2}\right) + Cn$$

$$S(n) \le C \cdot n + C\frac{n}{2} + C \cdot \frac{n}{4} + \cdots + C \cdot 1$$

$$\le C \cdot n \sum_{i=0}^{\infty} 2^{-i}$$

$$= 2C \cdot n$$

$$= O(n) \qquad \square$$



– Counting Inversions

Probem: Given array $A[1...n]$, count # inversions

inversion: (i,j) $1 \le i < j \le n$, $A[i] > A[j]$

example: $1, 2, 3, \dots, n$ $\#inv = 0$

$n, n-1, n-2, \dots, 1$ $\#inv = \frac{n(n-1)}{2}$

naive algorithm $\Theta(n^2)$

2 $\boxed{1 \quad 8 \quad 5 \quad 7}$ $\boxed{2 \quad 6 \quad 4 \quad 3}$ 3

attempt

```
Count(A)
  if len(A) ≤ 1 return 0
  split A into B, C
  count(B)
  count(C)
  merge(B, C)
```

result = 2 + 3 + # inversions between B and C

inv: any number in B > any number in C

- for each number in B, how many numbers in C are smaller

1 8 5 7

0 + 4 + 3 + 4 = 11

idea: easy to count the number of inversions if C is sorted

binary search $\boxed{2 \; 3 \; 4 \; 6}$

5

sort(C), do binary search

$O(n\log n)$ $O(n\log n)$

2 $\boxed{1 \; 5 \; 7 \; 8}$ $\boxed{2 \; 3 \; 4 \; 6}$ 3

1 2 3 4 5 6 7 8

in C that are smalle: 0 + 3 + 4 + 4 = 11

```
Countsort(A)
  if len(A) ≤ 1 return 0
  split A into B, C
  countsort(B)
  countsort(C)
  countmerge(B, C)
```