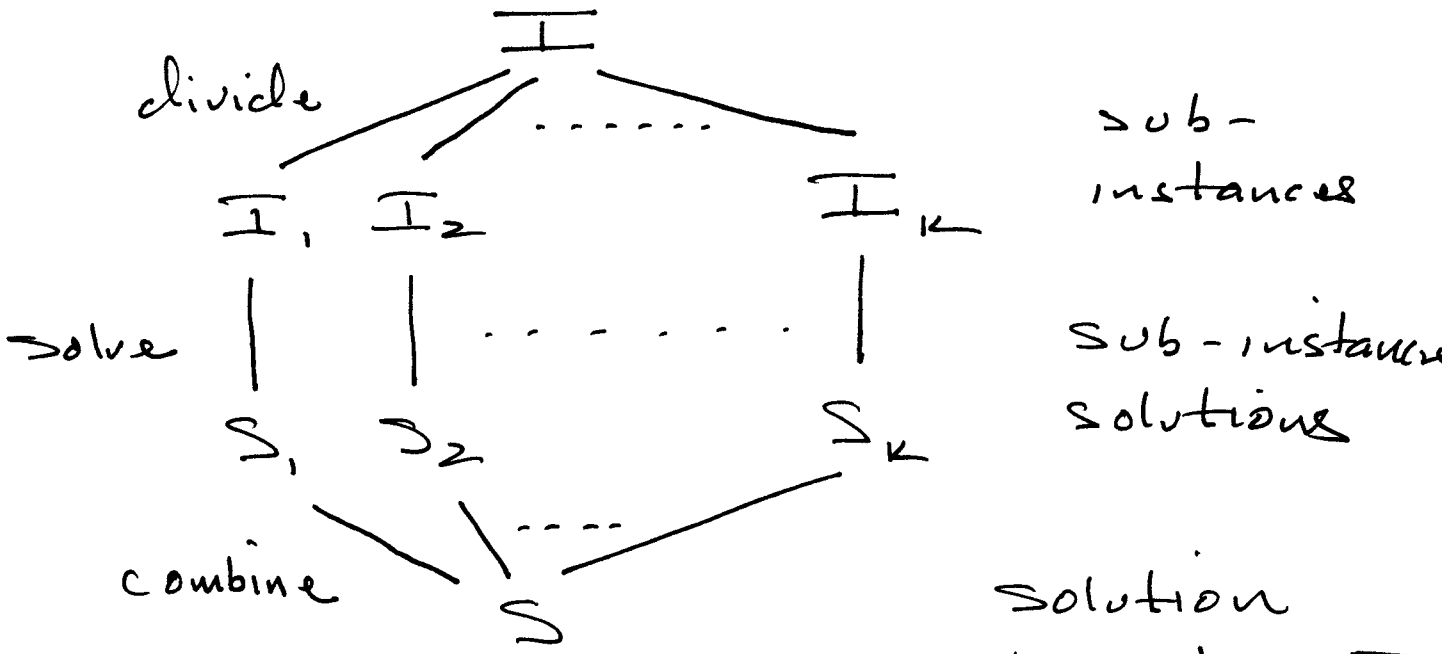
CSE 201 2-2-21

Divide & Conquer algorithms

- design
- analysis
- correctness

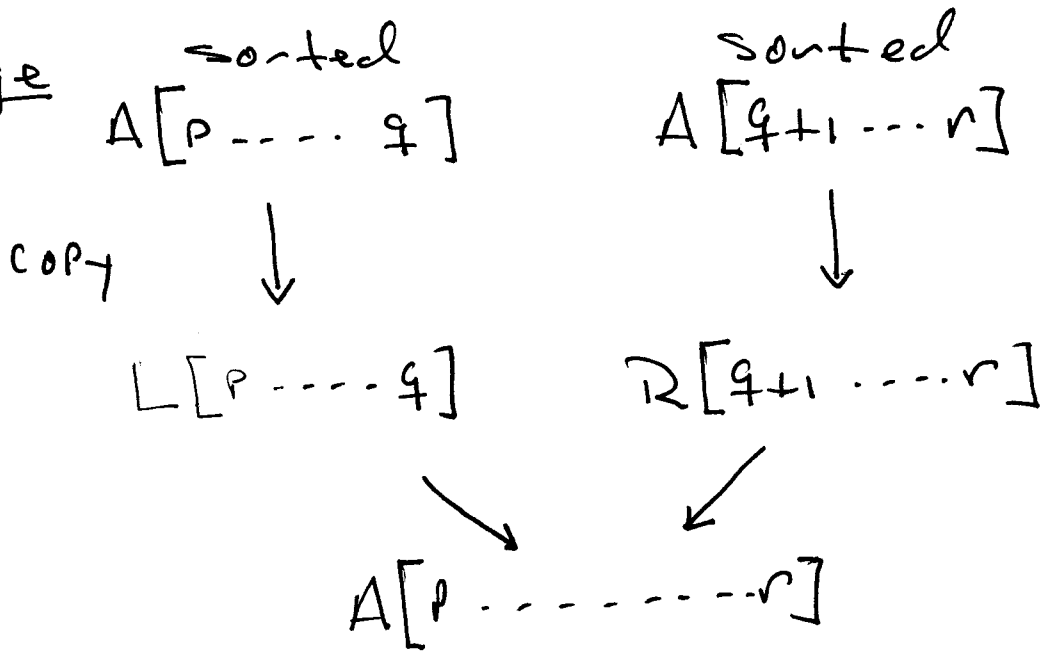
Problem instance: I



Recall MergeSort

MergeSort(A, p, r)	cost $\binom{p=1}{r=n}$
1. if $p < r$	0
2. $q = \lfloor \frac{p+r}{2} \rfloor$	0
3. MergeSort(A, p, q)	$T(\lceil \frac{n}{2} \rceil)$
4. MergeSort(A, q+1, r)	$T(\lfloor \frac{n}{2} \rfloor)$
5. Merge(A, p, q, r)	$n-1$

Merge



best case # of comparisons = $q - p + 1$

worst case # of comparisons = $(r-p+1) - 1 = r - p$

Let $T(n)$ denote worst case # of comparisons performed by MergeSort$(A, 1, n)$ where $n = \text{length}[A]$.

Exercise show that if $p=1, r=n$ $q = \lfloor \frac{1+n}{2} \rfloor$ then

$$\text{length}(A[p \cdots q]) = q - p + 1 = \lceil \tfrac{n}{2} \rceil$$

$$\text{length}(A[q+1 \cdots r]) = r - (q+1) + 1 = r - q = \lfloor \tfrac{n}{2} \rfloor \quad \blacksquare$$

so $T(n)$ must satisfy

$$T(n) = \begin{cases} 0 & n = 1 \\ T(\lceil \tfrac{n}{2} \rceil) + T(\lfloor \tfrac{n}{2} \rfloor) + (n-1) & n \geq 2 \end{cases}$$

Master Theorem:

simplify: $T(n) = 2T\left(\frac{n}{2}\right) + n$

compare: n to $n^{\log_2(2)} = n^1 = n$

Case 2: $\boxed{T(n) = \Theta(n \log n)}$ ← Merge sort cost

Insertion sort cost = $\boxed{\Theta(n^2)}$

(see chap. 1). observe

$$\frac{n \log n}{n^2} = \frac{\log n}{n} \longrightarrow 0 \quad \text{as } n \to \infty$$

$\therefore$ $n \log n = o(n^2)$

Exact solution in case n is an exact power of 2 : $n = 2^k$

recurrence becomes

$$T(n) = \begin{cases} 0 & n = 1 \\ 2T\left(\frac{n}{2}\right) + (n-1) & n \geq 2, \; n = 2^k \text{ (some } k\text{)} \end{cases}$$

iteration method

$$\begin{aligned} T(n) &= (n-1) + 2T\left(\frac{n}{2}\right) \\ &= (n-1) + 2\left(\left(\frac{n}{2}-1\right) + 2T\left(\frac{n}{2^2}\right)\right) \\ &= (n-1) + (n-2) + 2^2 T\left(\frac{n}{2^2}\right) \\ &= (n-1) + (n-2) + 2^2\left(\left(\frac{n}{2^2}-1\right) + 2T\left(\frac{n}{2^3}\right)\right) \\ &= (n-1) + (n-2) + (n-2^2) + 2^3 T\left(\frac{n}{2^3}\right) \\ &\vdots \\ &= \sum_{i=0}^{k-1} (n-2^i) + 2^k T\left(\frac{n}{2^k}\right) \end{aligned}$$

halt when $\frac{n}{2^k} = 1$, i.e. $n = 2^k$,

i.e. $\boxed{k = \lg(n)}$. for this k: $T\left(\frac{n}{2^k}\right) = 0$

$$\therefore T(n) = \sum_{i=0}^{k-1} (n - 2^i)$$

$$= \sum_{i=0}^{k-1} n - \sum_{i=0}^{k-1} 2^i$$

$$= kn - \frac{2^k - 1}{2 - 1}$$

$$= n\lg n - 2^{\lg(n)} + 1$$

$$\boxed{T(n) = n\lg(n) - n + 1}$$

check: $T(1) = 0$

$$\text{Rhs} = 2T\left(\frac{n}{2}\right) + (n-1)$$

$$= 2\left(\frac{n}{2}\lg\left(\frac{n}{2}\right) - \frac{n}{2} + 1\right) + (n-1)$$

$$= n(\lg n - 1) - n + 2 + n - 1$$

$$= n\lg n - n - \cancel{n} + 2 + \cancel{n} - 1$$

$$= n\lg n - n + 1$$

$$= T(n) = \text{Lhs}.$$

Theorem

Assume correctness of Merge(A, P, q, r).
After MergeSort(A, P, r) is called,
the sub-array A[P...r] is sorted.

Proof: induction on $m = r - P + 1$.

I. If $m = 1$, then $r = P$ and A[P...r] is already sorted. Indeed, the algorithm does nothing in this case.

IId. $\forall m > 1 : P(1) \wedge \cdots \wedge P(m-1) \longrightarrow P(m)$

Let $m > 1$ arbitrary. Assume that MergeSort( ) correctly sorts any sub-array of length less than m. we must show MergeSort( ) correctly sorts any sub-array of length m.

Now $m > 1 \Rightarrow r - p + 1 > 1 \Rightarrow r > p$, so condition on line 1 is true, so line 2 is executed, setting

$$q = \left\lfloor \frac{r+p}{2} \right\rfloor$$

This implies $p \leq q < r$ (exercise)
Thus

$$\text{length}(A[p \cdots q]) = q - p + 1 < r - p + 1 = m$$

$$\text{length}(A[q+1 \cdots r]) = r - (q+1)+1$$
$$= r - q \leq r - p < r - p + 1 = m$$

By the induction hypothesis, both $A[p \cdots q]$ and $A[q+1 \cdots r]$ are sorted after lines 3 and 4. Our assumption on Merge(A, p, q, r) says the sub-array $A[p \cdots r]$ is sorted after line 5. ∎

Exercise

write an algorithm that just checks to see if an array is sorted, and returns true/false accordingly.

- Iterative: $(A_1, A_2, A_3, A_4, \cdots, A_{n-1}, A_n)$
  - A_1 vs A_2: comp
  - A_2 vs A_3: comp
  - A_3 vs A_4: comp
  - A_{n-1} vs A_n: comp

  ($n-1$ comparisons in worst case)

- Divide & Conquer: emulate MergeSort()

  call algorithm: isSorted(A, p, r)

Exercise

Given $A = (A_1, A_2, \ldots, A_n)$, a pair of indices (i, j) is called an *inversion* iff

$$i < j \quad \text{and} \quad A_i > A_j$$

i.e. A_i, A_j are out of order.

Write an algorithm, modeled on MergeSort(), that counts the # of inversions in its input Array.

Two ways to do this

Inversione (A, p, r)

1. if $p < r$
2. $\quad q = \lfloor \frac{p+r}{2} \rfloor$ $\quad$ # inv. (i,j)
3. $\quad a =$ Inversione (A, p, q) $\quad p \le i < j \le q$
4. $\quad b =$ Inversione $(A, q+1, r)$ $\quad q+1 \le i < j \le r$
5. $\quad c =$ Compare (A, p, q, r) $\quad p \le i \le q < j \le r$
6. $\quad$ return $(a+b+c)$

Compare (A, p, q, r)

1. count $= 0$
2. for $i = p$ to q
3. $\quad$ for $j = q+1$ to r
4. $\quad\quad$ if $A[i] > A[j]$
5. $\quad\quad\quad$ count ++
6. return count

Runtime: $T(n)$ = # comparisons.

$$T(n) = \begin{cases} 0 & n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + \lfloor \frac{n}{2} \rfloor \cdot \lceil \frac{n}{2} \rceil & n \geq 2 \end{cases}$$

Simplify

$$T(n) = 2T(\tfrac{n}{2}) + n^2$$

check that case 3 gives

$$T(n) = \Theta(n^2)$$

Real exercise

Figure out how to do this in time $\Theta(n \log n)$.