CSE 102 10-25-22

Merge Sort

Sort an array A.

notation: $A[p \ldots q]$ is the subarray

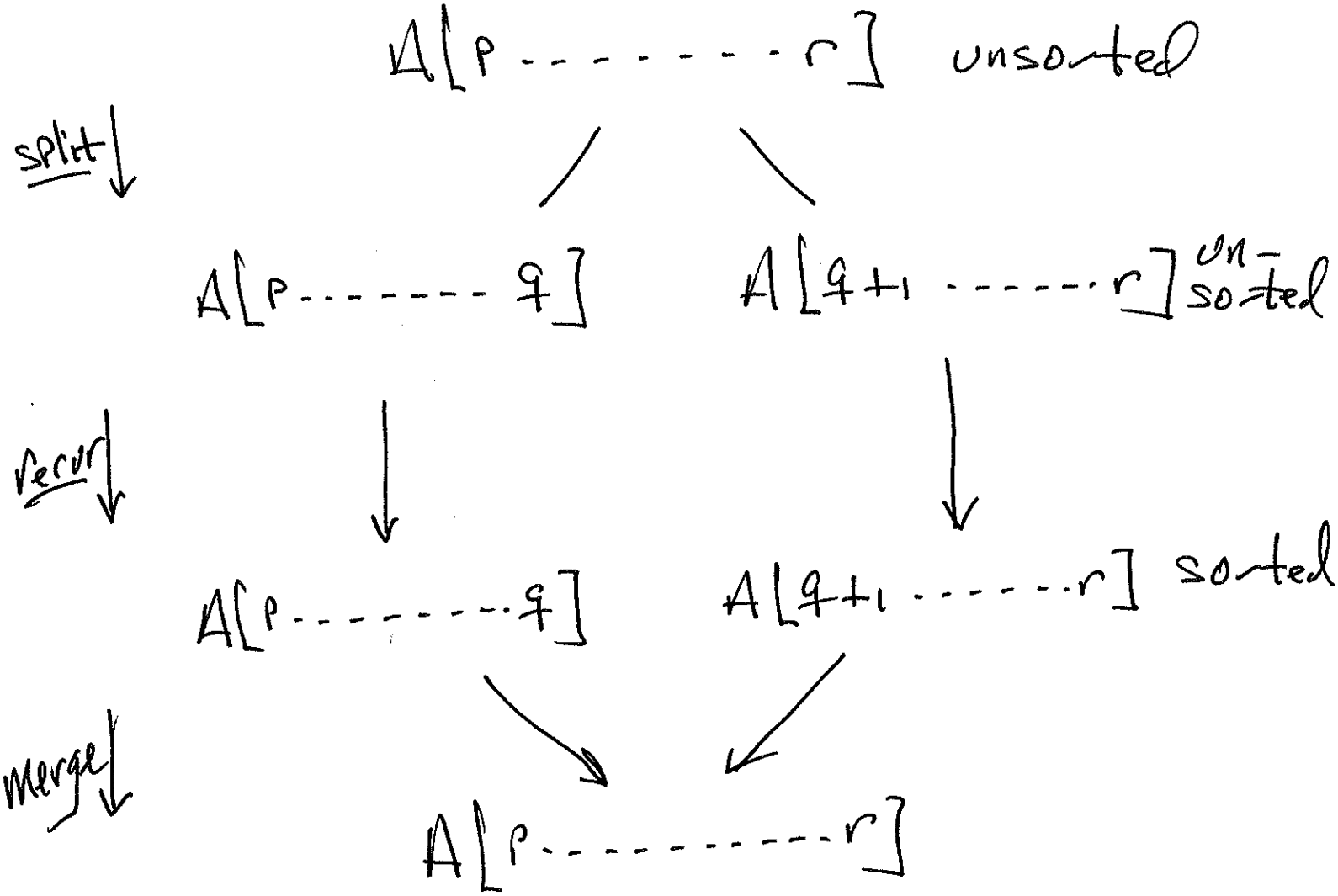
if $p \le q$: $(A[p], A[p+1], \ldots, A[q])$

if $p > q$: ϕ empty array

$$\text{length}(A[p \ldots q]) = q - p + 1$$

$$\text{length}(\underbrace{A[1 \ldots\ldots n]}_{\text{full array}}) = n - 1 + 1 = n$$

To sort $A[p \ldots\ldots r]$



Top level call : MergeSort(A, 1, n)

MergeSort(A, p, r) Pre: $p \le r$

		Cost
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)	$n-1$

Theorem:

Assuming the correctness of Merge() MergeSort(A, p, r) correctly sorts the subarray A[p----r].

Proof

Induction on $m = r - p + 1$.

I. if $m = 1$, then $r - p + 1 = 1 \therefore r = p$.

$\therefore$ condition on line 1 is false, so 2-5 are skipped. Indeed a sub-array of length 1 is already sorted.

IId. let $m > 1$. Assume MergeSort() correctly sorts any subarray of length less than m. we must show mergeSort() correctly sorts

$$A[p \ldots r]$$

where $m = r - p + 1$.

observe $m > 1$ implies $r - p + 1 > 1$,

$\therefore r - p > 0 \quad \therefore r > p$, so cond. on line 1 is true. Thus lines 2-5 are executed. also

$$p < r \Rightarrow p < \frac{p+r}{2} < r \Rightarrow p \leq \left\lfloor \frac{p+r}{2} \right\rfloor < r$$

$$\Rightarrow p \leq q < r$$

Thus

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

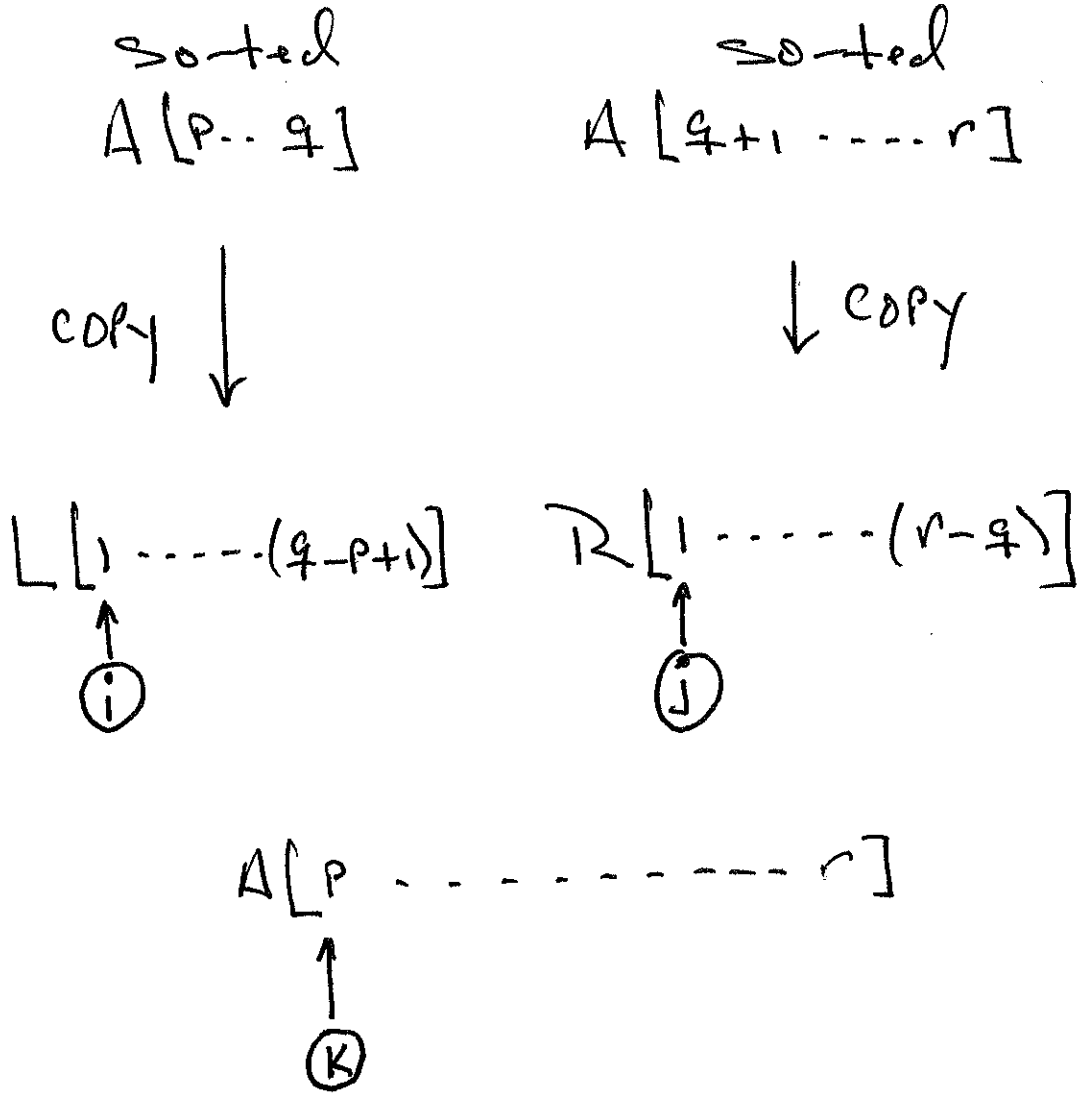
Also

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

The ind. hyp. implies that lines 3 and 4 result in $A[p..q]$ & $A[q+1..r]$ being sorted.

By our assumption on Merge(), line 5 results in $A[p \ldots r]$ being sorted. ∎

what about Merge(A, p, q, r) ?



Exercise:

write pseudo-code for Merge(A, p, q, r).

(or see sec. 2.3 of CLRS.)

To sort $A[1 \cdots n]$, call

MergeSort(A, 1, n)

Runtime of MergeSort()

- Pick a basic operation and count # of times it is executed on input of size n.

- in sorting algorithms:

  basic op. = comparison of array 2 elements

- we count # comparisons in

  - best case
* - worst case ← mostly this
  - average case ← occasionally.

Let $T(n)$ = worst case # of comparisons by M.S. on $A[1 \ldots n]$.

Let $p=1, r=n$, so $q=\lfloor \frac{n+1}{2} \rfloor$

Exercise: show that

- $\text{length}(A[p..q]) = q-p+1 = \lceil \frac{n}{2} \rceil$
- $\text{length}(A[q+1...r]) = r-q = \lfloor \frac{n}{2} \rfloor$

Note:

Worst case # of comparisons by Merge(A, p, q, r) is

$$\text{len}(A[p...r]) - 1 = (r-p+1) - 1$$
$$= r-p$$

since $p=1, r=n$, $\boxed{\#\text{ comp} = n-1}$

so $T(n)$ satisfies

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

simplify for master thm:

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

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

By case 2:

$$\boxed{T(n) = \Theta(n \log n)}$$

Suppose n = exact pow of 2

so $\lfloor \frac{n}{2} \rfloor = \lceil \frac{n}{2} \rceil = \frac{n}{2}$ at <u>all</u> levels of recursion.

Simplify to

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

Iteration method

$$\begin{aligned} T(n) &= (n-1) + 2T\left(\frac{n}{2}\right) \\ &= (n-1) + 2\left(\left(\frac{n}{2}-1\right) + 2T\left(\frac{n/2}{2}\right)\right) \\ &= (n-1) + (n-2) + 2^2 T\left(\frac{n}{2^2}\right) \\ &= (n-1) + (n-2) + 2^2\left(\left(\frac{n}{2^2}-1\right) + 2T\left(\frac{n/2^2}{2}\right)\right) \\ &= (n-1) + (n-2) + (n-2^2) + 2^3 T\left(\frac{n}{2^3}\right) \end{aligned}$$

$$\vdots$$

$$= \sum_{i=0}^{K-1} (n - 2^i) + 2^K T\left(\frac{n}{2^K}\right)$$

$$= \sum_{i=0}^{K-1} n - \sum_{i=0}^{K-1} 2^i + 2^K T\left(\frac{n}{2^K}\right)$$

$$= Kn - \frac{2^K - 1}{2 - 1} + 2^K T\left(\frac{n}{2^K}\right)$$

$$= Kn - 2^K + 1 + 2^K \cdot T\left(\frac{n}{2^K}\right)$$

stop when $\frac{n}{2^K} = 1 \Rightarrow n = 2^K \Rightarrow \boxed{K = \lg(n)}$

$\therefore \boxed{T(n) = n\lg(n) - n + 1}$ exact soln in special case n = Pow. of 2,

Idea: instead of splitting $A[p \ldots r]$ into 2 sub-arrays, try 3.

Recursion:

$$T(n) = 3T\left(\frac{n}{3}\right) + \Theta(n)$$

By master-thm:

compare: n to $n^{\log_3 3} = n$

still $\boxed{T(n) = \Theta(n \log n)}$

Exercise

Write an algorithm (based on Merge sort) that just checks to see if array is sorted.

$A[p \ldots r]$

- Prove its correctness by ind.
- analyse runtime (count # comparisons.)

answer: $\Theta(n)$

Iterative Version

$A_1 \; A_2 \; A_3 \; A_4 \; \ldots \; A_{n-1} \; A_n$

comp $= n-1 = \Theta(n)$.

distinct elements

Exercise

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

- $i < j$
- and $A_i > A_j$

write an algorithm (Based on mergeSort) that counts # inversions.

note: # Pairs (i,j) with $i < j$ is $\binom{n}{2} = \frac{n(n-1)}{2}$.

- write algorithm
- prove correctness
- analyze runtime

2 Poss. runtimes!

— $\Theta(n^2)$

— $\Theta(n \log n)$ better.