COMP-250 - Lecture 16

MergeSort - Proof of correctness, running time ↓ recurrences

array a - n elements

Algorithm MergeSort (a, p, q)

Output: array a is sorted between p and q

// Call with MergeSort (a, 0, n-1)

Divide-and-conquer-style algorithms

Big problem → split it in pieces → solve → "glue"

if (p < q) // have something to sort

	int $m \leftarrow \lfloor \frac{p+q}{2} \rfloor$	
$T(n/2)$	mergeSort (a, p, m)	$\forall i,j \in \{p \dots m\}\ a[i] \le a[j]$ if $i \le j$
$T(n/2)$	mergeSort (a, m+1, q)	$\forall i,j \in \{m+1 \dots q\}\ a[i] \le a[j]$ if $i \le j$
$T_{merge}(n)$	merge (a, p, m, q)	$\forall i,j \in \{p \dots q\}, a[i] \le a[j]$ if $i \le j$

Correctness: proof by induction by length of array

Suppose merge is correct

T(n) - running time of MergeSort on array of size n

Base case: array of size 1

MergeSort(a[], 0, 0) → no action done ☺

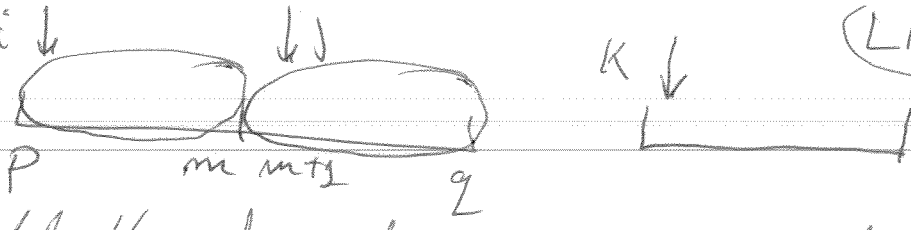
Induction step: Suppose rec. calls worked (on array of size $n/2$)

Array of size n; using induction, correctness of merge $\Rightarrow \forall i \le j,\ i, j \in \{0, \ldots n-1\}$

$a[i] \le a[j]$ ☺

Put your faith in the recursion

Merge algorithm



tmp array to hold the elements → not in-place

selection sort was in-place (no extra array was allocated)

Memory vs time tradeoff

SelSort vs MergeSort

slower → less mem

faster → more mem

Algorithm merge (a, p, m, q)

Array tmp of size q-p+1 → size m

int i ← p, j ← m+1, k ← 0

while (i ≤ m or j ≤ q)

if j = q+1 or a[i] ≤ a[j] then

tmp[k] ← a[i]

k++; i++;

if i = m+1 or a[i] > a[j] then

tmp[k] ← a[j]

k++ j++;

for → copy from tmp to a

tmp - each element is "filled" one by one O(n)

Loop invariant

tmp array, for all l, f ≤ k

l ≤ f

tmp[l] ≤ tmp[f]

(tmp[k] is value in between these already in tmp and those left in a

for i, j in the array a

a[i] is min of a[i], a[i+1] ... a[m]

(assuming portion p ... m sorted)

a[j] is min of a[j] a[j+1] ... a[q]

(m+1 ... q sorted)

a[p] ... a[i] → a[i] is max

a[m+1] .. a[j] → a[j] is max

tmp[l] tmp[f] tmp[k]

tmp[l] ≤ tmp[k]

tmp[f] ≤ tmp[k]

tmp[l] ≤ tmp[f] because tmp written left→right

tmp[k] = min(a[i], a[j])

a[p] ... a[i-1] ≤ tmp[k]

a[m+1] .. a[j-1] ≤ tmp[k]

Running time for Merge Sort:

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

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

↑ running time of alg as fn of running time of "pieces"

recurrence

Merge is $O(n) \Rightarrow T_{merge}(n) = c \cdot n$

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

$$= 2\left[2T\left(\frac{n}{2^2}\right) + c\frac{n}{2}\right] + cn =$$

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

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

$$= 2^2\left[2T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2}\right] + 2cn =$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 3cn$$

Until when can we unroll?

$$2^k T\left(\frac{n}{2^k}\right) \to T(1) \quad + k \cdot c \cdot n$$

For what K is this true?

$$\frac{n}{2^K} \simeq 1 \qquad n \simeq 2^K \Rightarrow K \simeq \log_2 n$$

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

$$= 2^{\log_2 n} \cdot T(1) + c \cdot n \cdot \log_2 n$$

$$= n \cdot c_1 + c\, n \log_2 n \in O(n \log_2 n)$$

O eats up smaller terms

Max recursion

$$T(n) = T(n-1) + c =$$

$$= T(n-2) + c + c = T(n-2) + 2c =$$

$$= T(n-3) + c + 2c = T(n-3) + 3c$$

$$= T(1) + n \cdot c$$

Proving that the sol to rec. is what we calculated
$\Rightarrow$ proof by induction.