MAT2540, Classwork4, Spring2026

5.4 Recursive Algorithms (Conti.)

11. A Pseudocode to merge two ordered lists.

Pseudocode:

procedure *merge*(L_1, L_2 : sorted lists)
$L = \{\}$ (L is an empty list)
while ($L_1 \neq \phi$ and $L_2 \neq \phi$)
remove smaller of first elements of L_1 and L_2 from its list; put it at the right end of L
if ($L_1 = \phi$ or $L_2 = \phi$) **then** remove all elements from the nonempty list and append them to L
return L {output is the merged list with elements in increasing order}

12. A Recursive Merge Sort.

Merge Sort: A sorting algorithm to put the terms of the list in increasing order.

In general, a merge sort includes two steps:

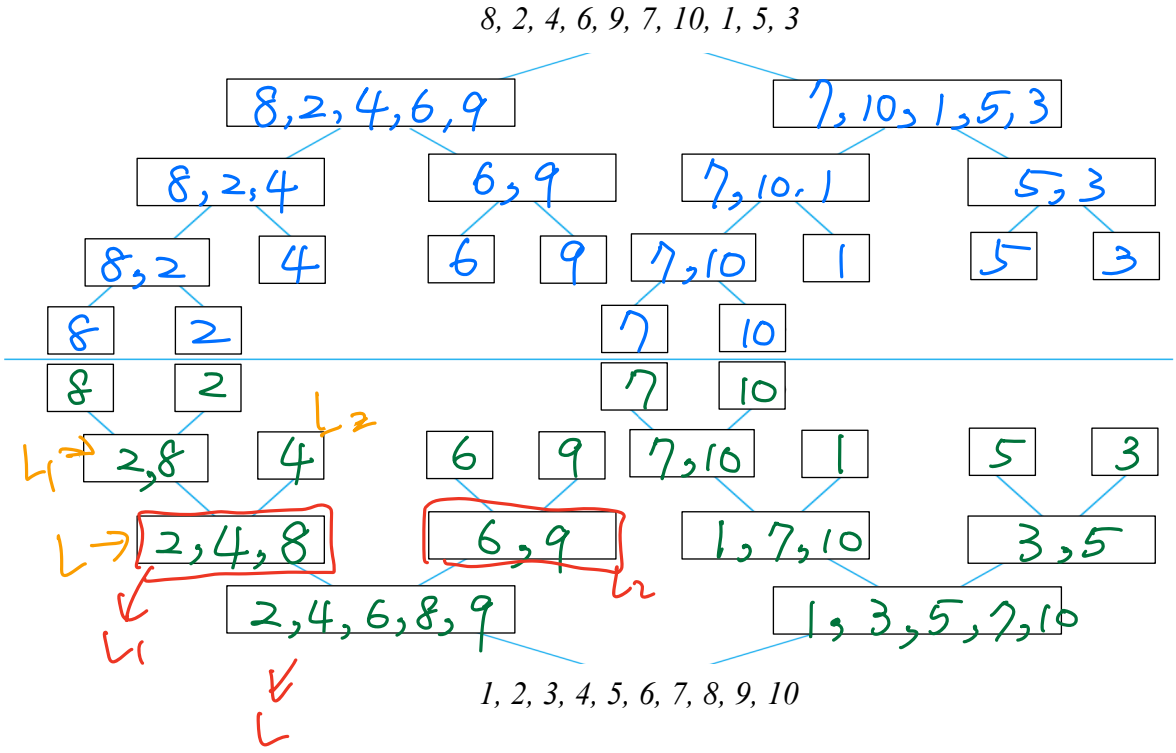
(1) It proceeds by iteratively splitting lists into two sublists of equal length (or where one sublist has one more element than the other) until each sublist contains one element.

(2) The procedure continues by successively merging pairs of lists, where both lists are in increasing order, into a larger list with elements in increasing order, until the original list is put into increasing order.

Here we use an example to explain the algorithm:

Use the merge sort to put the terms of the list 8, 2, 4, 6, 9, 7, 10, 1, 5, 3 in increasing order.

Algorithm:



Pseudocode:

floor function $\lfloor \frac{n}{2} \rfloor = \begin{cases} \frac{n}{2} & \text{if } n \text{ is even} \\ \frac{n-1}{2} & \text{if } n \text{ is odd} \end{cases}$

procedure *mergesort*$(L = \{a_1, \cdots, a_n\})$

if $n > 1$ **then**

$m :=$ $\lfloor \frac{n}{2} \rfloor$ (find the index to split L in half)

$L_1 := \{a_1, a_2, \cdots, a_m\}$, $L_2 := \{a_{m+1}, a_{m+2}, \cdots, a_n\}$

$L := merge(mergesort(L_1), mergesort(L_2))$

{ L is now sorted into elements in nondecreasing order}

13. How many comparison operations do we need when merging $\{2, 3, 5, 6\}$ and $\{1, 4\}$ into one list?

4 comparisons

$a=4$ $b=2$ $\Rightarrow a+b-1=5$

14. Two sorted lists with a elements and b elements can be merged into a sorted list using no more than $a+b-1$ comparisons.

15. *Theorem*. The number of comparisons needed to merge sort a list with n elements is $O(n \cdot \log\ n)$.

To analyze the complexity of the merge sort of a list with n elements, we assume $n = 2^m$ for simplicity.

$m > 0$, m is an integer.

The first stage of the splitting procedure:

Level		# of elements in each list	# of list(s)
0		2^m	$2^0 = 1$
1		$\frac{2^m}{2} = 2^{m-1}$	$2^1 = 2$
2		$\frac{2^m}{4} = \frac{2^{m-1}}{2} = 2^{m-2}$	$2^2 = 4$
⋮	⋮	⋮	⋮
k	⋯	$\frac{2^m}{2^k} = 2^{m-k}$	2^k
⋮	⋮	⋮	⋮
m	⋯	$\frac{2^m}{2^m} = 1$	2^m

The second step of merging by combining pairs of lists:

From level m to level $m-1$, there are 2^{m-1} comparisons.

In level k, there are 2^{m-k} elements in each list and it needs at most $2^{m-k} + 2^{m-k} - 1$ comparisons to merge a pair of them, there are totally 2^{k-1} pairs.

Therefore, from level k to level $k-1$, it needs at most $2^{k-1}(2^{m-k} + 2^{m-k} - 1)$ comparisons.

Thus, merging from level m to level 0, it needs

$$\sum_{k=1}^{m} 2^{k-1}(2^{m-k} + 2^{m-k} - 1) = \sum_{k=1}^{m} 2^m - \sum_{k=1}^{m} 2^{k-1} = m(2^m - (2^m - 1)) = n\log(n) - n + 1$$

Here, we have $n = 2^m$ implies $m = \log(n)$.