CS 340 - lecture 21 - Oct 26

partitioning in Quicksort:

3.1. Swap pivot into list[last]

3.2. i = first; j = last - 1;

3.3. while (i ≤ j)

```
{
    while ((i ≤ last) and (list[i] < pivot)) {i++};
    while ((j ≥ first) and (list[j] ≥ pivot)) {j--};
    if (i < j)  swap (list[i], list[j]);
}
```

3.4. swap (list[i], list[last])

Example 37: 6, 1, 4, 9, 0, 3, 5, 2, 7, 8

median-of-three: pivot = 6

pivot: 6

8, 1, 4, 9, 0, 3, 5, 2, 7, 6

i does not move

2, 1, 4, 9, 0, 3, 5, 8, 7, 6

2, 1, 4, 5, 0, 3, 9, 8, 7, 6

3.4. swap pivot back:

will not change from now on

2, 1, 4, 5, 0, 3, 6, 8, 7, 9

sort recursively, pivot 3 | sort recursively, pivot 8

Usually, one cuts off Quicksort when the lists get short, e.g., $N \leq 20$, and then runs Insertion Sort (more efficient).

3.6. A lower bound

Heapsort, Mergesort: $T_{worst}(N) = \Theta(N \log N)$

Can we do better? NO!

Theorem 12. Any sorting algorithm that uses only comparisons between list elements requires $\Omega(N \log N)$ comparisons to sort a list of N elements, in the worst case.

"Proof". Let A be a sorting algorithm that uses only comparisons between list elements.

There are exactly $N!$ potential orderings of list elements produced by A on inputs of (duplicate-free) list of length N.

e.g., if $N=3$, list=[l0, l1, l2], there are $3! = 6$ potential sorting results:

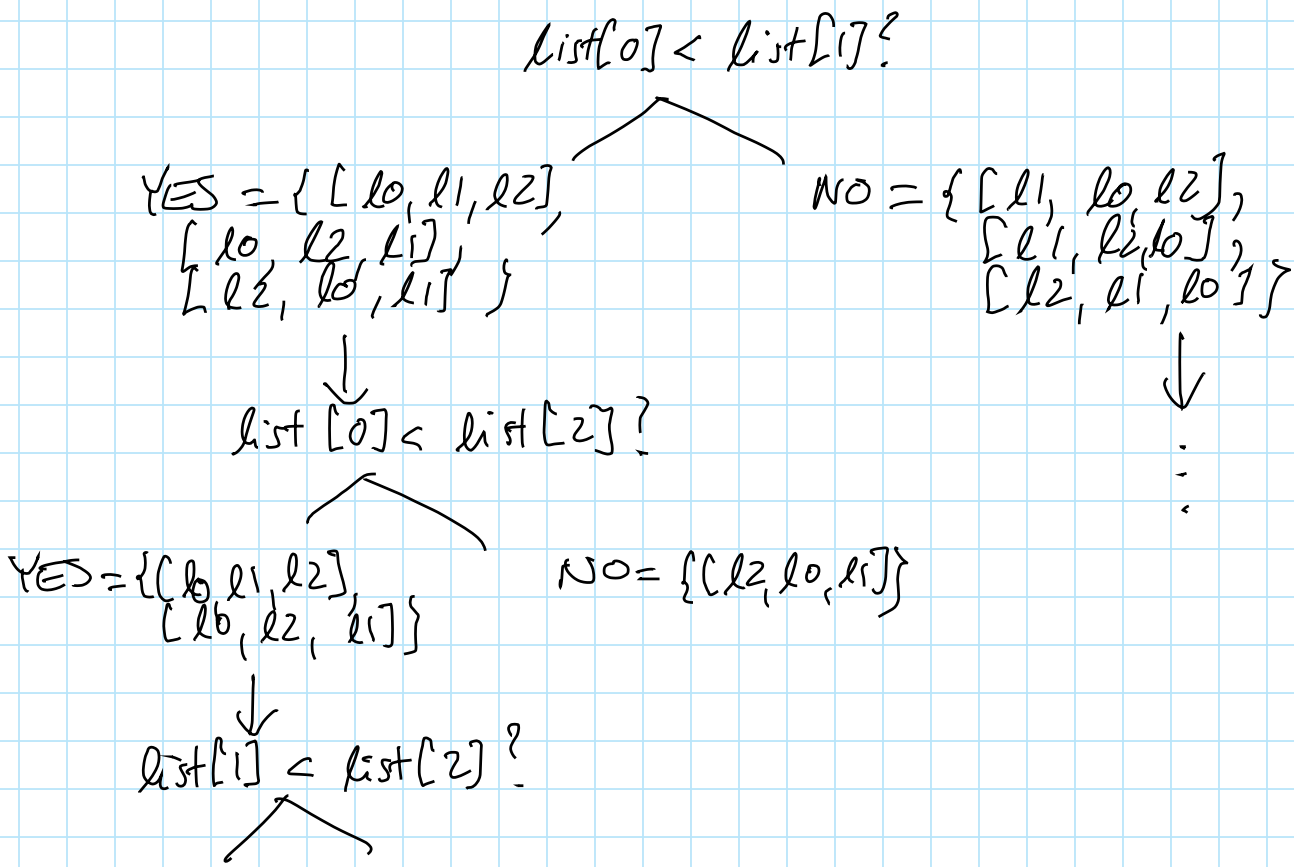
[l0, l1, l2], [l0, l2, l1], [l1, l0, l2], [l1, l2, l0], [l2, l0, l1], [l2, l1, l0]

Every comparison may rule out some of these orderings.

For any comparison "list[i] < list[j]" made by A, let

YES be the set of current candidate orderings in which li is left of lj

NO " " " lj " " li



For each comparison made, at least one of the sets YES and NO contains at least half of the current candidates (YES ∪ NO).

Consider the case of an input list L whose correct ordering is always contained in the larger of the two sets, for all comparisons made by A on L.

after ... comparisons	# candidate orderings
0	$N!$
1	$\geq N!/2$
2	$\geq N!/4$
3	$\geq N!/8$
⋮	⋮

$\Rightarrow$ A needs at least $\lceil \log(N!) \rceil$ comparisons on L

$\lceil \log(N!) \rceil = \Omega(N \log N)$. □

$\Rightarrow$ Asymptotically, Mergesort and Heapsort have the best possible worst case running time!

one can also prove:

Theorem 13. Any sorting alg. that uses only comparisons between list elements requires $\Omega(N \log N)$ comparisons on average to sort a list of N elements.

$\Rightarrow$ Asymptotically, Quicksort, Mergesort, and Heapsort have the best possible avg. case running time.

3.7 External Sorting

what if input does not fit into main memory?

accessing data dominates cost!

$\Rightarrow$ algorithms studied so far would be inefficient.

$\Rightarrow$ different algorithms needed

choice depends on storage device

e.g. tape: access data sequentially

(as opposed to disk)

$\Rightarrow$ SEE HANDOUT