Recitation Notes: sorting

1 Merge Sort

- **Idea:** Divide array in half, recursively sort each half, merge the two sorted halves.
- **Pseudocode:**
  1. If $|A| \leq 1$, return A.
  2. Split A into $L = A[1 \ldots \lfloor n/2 \rfloor]$ and $R = A[\lfloor n/2 \rfloor + 1 \ldots n]$.
  3. Recursively sort L and R.
  4. Merge L and R by repeatedly taking the smaller front element (if there is a tie, take the element from L). This requires at most $n-1$ comparisons.
- **Time complexity:** $T(n) = O(n \log n)$ (apply Master Theorem to the recurrence $T(n) = 2\,T(n/2) + O(n)$, or note that there are $O(\log n)$ recursion levels, each taking $O(n)$ time).
- **Key properties:**
  - Deterministic, $O(n \log n)$ worst-case (no bad inputs).
  - Stable sort (equal elements preserve original order), since we take the left-hand element during tiebreaking.
  - Requires $O(n)$ extra space for the merge step.

2 Quick Sort

- **Idea:** Pick a pivot p, partition into elements $< p$ and $> p$, recursively sort each side.
- **Pseudocode:**
  1. If $|A| \leq 1$, return A.
  2. Pick arbitrary pivot element p from A.
  3. $LESS \leftarrow \{a_i \in A : a_i < p\}$, $\quad GREATER \leftarrow \{a_i \in A : a_i > p\}$.
  4. Return Quicksort($LESS$) + $\{p\}$ + Quicksort($GREATER$).
- **Time complexity:**
  - **In general:** worst-case $O(n^2)$ occurs when pivot is always the min or max.
  - **If we pick pivot uniformly at random:** expected $O(n \log n)$ time.

  - **If we pick median as pivot:** worst-case $O(n \log n)$ time. Median element can be found in $O(n)$, though the median-finding algorithm is tricky.

- **Key properties:**
  - Worst-case $O(n^2)$ time, but expected $O(n \log n)$ time, and practically very fast due to low constant factor.
  - Not stable in the standard in-place implementation.
  - In-place (no extra array needed).

3 Merge Sort vs. Quick Sort

- Both are divide-and-conquer with $O(n \log n)$ typical performance.
- Merge Sort: worst-case $O(n \log n)$, but uses $O(n)$ extra space.
- Quicksort: $O(n^2)$ worst case (deterministic), but $O(1)$ extra space in-place. Randomized Quicksort gives expected $O(n \log n)$ with no extra space. Also, deterministic $O(n \log n)$ Quicksort is possible if you pick the pivot intelligently.

4 Recitation Problem

Problem 1: sorting via flop operations

On an array $A[1 \dots n]$, define a flop operation on the range (l, r) to be swapping the minimum and maximum elements in the range $A[l \dots r]$. Prove that any permutation on $1 \dots n$ can be converted to any other permutation on $1 \dots n$ using $O(n \log^2 n)$ flop operations.

Proof. Note that the operations are reversible. So it suffices to show that any permutation can be converted to $1 \dots n$ in this many operations.

We try to emulate merge sort.

Assume $A[1 \dots k]$ and $A[(k+1) \dots n]$ are already sorted.

We show that these two lists can be merged using $O(n \log n)$ flop operations, via another divide and conquer.

Let the median of the overall list be m. Let its position in the first andn second half be i and $k+j$ respectively:

$$A[1] < A[2] < \dots < A[i] \leq m < A[i+1] < \dots A[k]$$
$$A[k+1] < A[k+2] < \dots < A[k+j] \leq m < A[k+j+1] < \dots A[n]$$

Note that $k - i = j$ (up to an off by 1), so it suffices to swap these two arrays' positions.

To do this, note that if we reverse the order of $A[i+1 \ldots k]$ and $A[k+1 \ldots k+j]$, we actually get a list that's decreasing. Formally

$$A[k+1] < A[k+2] < \ldots < A[k+j] < m < A[i+1] < A[i+2] < \ldots < A[k]$$

is equivalent to

$$A[k] > A[k-1] > \ldots > A[i+1] > m > A[k+j] > A[k+j-1] > \ldots A[k+1]$$

so we take this list, reverse it again, and end up with $A[k+1] \ldots A[k+j]$ before $A[l+1] \ldots A[k]$.

The last detail missing is reversing a sorted list. But this is easy: calling flop on first and last exchanges them, same with second and second last, etc...

□