Tail Recursion

Speaker: MARK
What is in-place algorithm?

- Algorithm that uses $O(1)$ extra space in addition to the original input

- How about Quicksort?
  - Quicksort has in-place partition
  - Then, Quicksort is in-place algorithm? NO !!
Quicksort

The Quicksort algorithm works as follows:

\[
\text{Quicksort}(A, p, r) \quad /* \text{to sort array } A[p..r] */
\]

1. if ( \( p \geq r \) ) return;
2. \( q = \text{Partition}(A, p, r) \);
3. \text{Quicksort}(A, p, p+q-1);
4. \text{Quicksort}(A, p+q+1, r);

In-place!

In-place?
Quicksort needs stack

1 3 7 8 2 6 4 5

after partition

1 3 2 4 5 6 7 8

Qsort(A,1,4)  Qsort(A,6,8)

STACK

Q(1,4), Q(6,8)
Q(1,8)
Quicksort needs stack (cont.)

1 3 2 4 5 6 7 8

after partition

Qsort(A,1,3)  Qsort(A,6,7)

STACK

Q(1,3),Q(6,7)
Q(1,4),Q(6,8)
Q(1,8)
Worst Case Space

STACK size = $O(n)$ entries

Can we use less space?
Method I

while (length(X_i) > n/2)
  partition again

until all length(X_i) < n/2
Method I (cont.)

STACK

\[ Q(X_3), Q(X_4), Q(X_7) \]

STACK size = \( O(\log n) \) entries

STACK

\[ Q(X_5), Q(X_6), Q(X_7) \]

Is it good enough?

No! Space of an entry may be as large as \( O(n) \)
Method II

if (there is X with length(X) < n/2) call Qsort(X)
else partition X into X' and X''

until all X are processed
Method II (cont.)

STACK size = $O(\log n)$ entries

Space of every entry is only $O(1)$
Conclusion

- The idea of Method II is tail recursion
  - First solves sub-problem with smaller size
  - Call recursion only when sub-problem is small enough

- Even with the improvement, Method II’s space complexity = input + \( O(\log n) \)
  - Still not in-place algorithm !!