Lecture 4: Merge Sort & Asymptotic Notations

Agenda:

- Merge sort (analysis later)
  - a quick review
  - recursion
  - correctness
- Asymptotic notations

Reading:

- Textbook pages 28 – 61

Merge sort pseudocode

```
Merge(A; lo, mid, hi)    **p 29
       **pre-condition:  lo ≤ mid ≤ hi
       **pre-condition:  A[lo, mid] and A[mid + 1, hi] sorted
       **post-condition:  A[lo, hi] sorted

MergeSort(A; lo, hi)    **p 32
   if lo < hi then
       mid ← ⌊(lo + hi)/2⌋
       MergeSort(A; lo, mid)
       MergeSort(A; mid + 1, hi)
       Merge(A; lo, mid, hi)
```

Algorithm analysis issues (review):

- Issues:
  - correctness
  - resources used (time & space)
  - optimality
- Estimating resources used
  - input size n: time $T(n)$ & space $S(n)$
  - worst/best/average case (WC/BC/AC)
  - machine independent — computational model
- Model of computation
  - simple (architecture, instruction set)
  - reflective (typical machine, accurate estimates)
  - our choice — RAM (random access machine)
  - two versions: uniform cost & log cost
    - choose version depending on applications
    - astronomical numbers — log cost RAM
    - reasonable size numbers — uniform cost RAM

Algorithm running time analysis

— what have been counted?

E.g.,

- RAM instructions?
- log RAM instructions?
- Data moves?
- Data comparisons?
- Arithmetic operations?
  - additions? subtractions?
  - multiplications? divisions?
- Pentium IV clock cycles? etc.

Merge sort, the big idea — divide-and-conquer:

- Divide the whole list into 2 sublists of equal size;
- Recursively merge sort the 2 sublists;
- Combine the 2 sorted sublists into a sorted list.

  Make sure you do know how to combine !

- This is the idea of *recursion*.
  - a programming technique (not really a design technique)
  - recursion trees
  - recurrence relations

Example:

	1	2	3	4	5	6	7	8	9	10	11	12	13
A	[31	23	01	17	19	28	09	03	13	15	22	08	29]

$\boxed{1:13}$

$\boxed{1:7}$ $\boxed{8:13}$

$\boxed{1:4}$ $\boxed{5:7}$ $\boxed{8:10}$ $\boxed{11:13}$

$\boxed{1:2}$ $\boxed{3:4}$ $\boxed{5:6}$ $\boxed{7:7}$ $\boxed{8:9}$ $\boxed{10:10}$ $\boxed{11:12}$ $\boxed{13:13}$

$\boxed{31\ 23}$ $\boxed{01\ 17}$ $\boxed{19\ 28}$ $\boxed{09}$ $\boxed{03\ 13}$ $\boxed{15}$ $\boxed{22\ 08}$ $\boxed{29}$

$\boxed{23\ 31}$ $\boxed{01\ 17}$ $\boxed{19\ 28}$ $\boxed{09}$ $\boxed{03\ 13}$ $\boxed{15}$ $\boxed{08\ 22}$ $\boxed{29}$

$\boxed{01\ 17\ 23\ 31}$ $\boxed{09\ 19\ 28}$ $\boxed{03\ 13\ 15}$ $\boxed{08\ 22\ 29}$

$\boxed{01\ 09\ 17\ 19\ 23\ 28\ 31}$ $\boxed{03\ 08\ 13\ 15\ 22\ 29}$

$\boxed{01\ 03\ 08\ 09\ 13\ 15\ 17\ 19\ 22\ 23\ 29\ 31}$

Asymptotic notations, motivations:

- Analysis of algorithms becomes analysis of functions:
  - *e.g.*,
    $f(n)$ denotes the WC running time of insertion sort
    $g(n)$ denotes the WC running time of merge sort
  - $f(n) = c_1 n^2 + c_2 n + c_3$
    $g(n) = c_4 n \log n$
  - Which algorithm is preferred (runs faster)?
- To simplify algorithm analysis, want function notation which indicates *rate of growth* (a.k.a., *order* of complexity)

$O(f(n))$ — read as "big O of $f(n)$"

roughly, The set of functions which, as n gets large, grow no faster than a constant times $f(n)$.

precisely, (or mathematically) The set of functions $\{h(n) : N \to R\}$ such that for each $h(n)$, there are constants $c_0 \in R^+$ and $n_0 \in N$ such that $h(n) \le c_0 f(n)$ for all $n > n_0$.

examples: $h(n) = 3n^3 + 10n + 1000 \log n \in O(n^3)$

$h(n) = 3n^3 + 10n + 1000 \log n \in O(n^4)$

$$h(n) = \begin{cases} 5^n, & n \le 10^{120} \\ n^2, & n > 10^{120} \end{cases} \in O(n^2)$$

Have you understood the lecture contents?

well	ok	not-at-all	topic
☐	☐	☐	alg analysis in general
☐	☐	☐	divide-and-conquer: merge sort idea
☐	☐	☐	merge sort algorithm
☐	☐	☐	why asymptotic notations
☐	☐	☐	what $O(f(n))$ means