Lecture 14: Quicksort

Agenda:

- Quicksort
  - AC running time (KC)
  - Space requirement
  - Improvement
- Two useful trees in algorithm analysis
  - Recursion tree
  - Decision tree

Reading:

- Textbook pages 153 – 168

Quicksort AC running time:

- Recurrence

  $$T(n) = \begin{cases} 0, & \text{when } n = 0, 1 \\ T(n_1) + T(n-1-n_1) + (n-1), & \text{when } n \geq 2 \end{cases}$$

- Average case: "What is the probability for the left subarray to have size n_1?"

  Average case: always ask "average over what input distribution?"

  - Unless stated otherwise, assume each possible input equiprobable

    Uniform distribution

  - Here, each of the ____ possible inputs equiprobable

  - Key observation: equiprobable inputs imply for each key, rank among keys so far is equiprobable

    So, n_1 can be $0, 1, 2, \ldots, n-2, n-1$, with the same probability $\frac{1}{n}$

-
$$\begin{aligned} T(n) &= \tfrac{1}{n}(T(0) + T(n-1)) \\ &\quad + \tfrac{1}{n}(T(1) + T(n-2)) \\ &\quad + \ldots \\ &\quad + \tfrac{1}{n}(T(n-2) + T(1)) \\ &\quad + \tfrac{1}{n}(T(n-1) + T(0)) \\ &\quad + (n-1) \\ &= \tfrac{2}{n}\sum_{i=0}^{n-1} T(i) + (n-1) \end{aligned}$$

Solving $T(n)$:

- $T(n) = \frac{2}{n}\sum_{i=0}^{n-1} T(i) + (n-1)$
- Therefore,
  - $n \times T(n) = 2\sum_{i=0}^{n-1} T(i) + n(n-1)$
  - $(n-1) \times T(n-1) = 2\sum_{i=0}^{n-2} T(i) + (n-1)(n-2)$
- Therefore,

  $$n \times T(n) - (n-1) \times T(n-1) = 2T(n-1) + 2(n-1)$$

  Rearrange it:

  $$nT(n) = (n+1)T(n-1) + 2(n-1)$$

  Or

  $$\begin{aligned} \frac{T(n)}{n+1} &= \frac{T(n-1)}{n} + \frac{2(n-1)}{n(n+1)} \\ &= \frac{T(n-1)}{n} + \frac{2}{n+1} - 2(\frac{1}{n} - \frac{1}{n+1}) \\ &= \frac{T(n-1)}{n} + \frac{4}{n+1} - \frac{2}{n} \end{aligned}$$

  which gives you (iterated substitution)

  $$\frac{T(n)}{n+1} = \sum_{i=1}^{n} \frac{2}{i+1} + \left(\frac{2}{n+1} - 2\right)$$

Solving $T(n)$ (cont'd):

- Recall that $\sum_{i=1}^{n} \frac{1}{i} = H_n = \ln n + \gamma$ — the Harmonic number where $\gamma \approx 0.577\cdots$

- So, from
$$\frac{T(n)}{n+1} = \sum_{i=1}^{n} \frac{2}{i+1} + \left(\frac{2}{n+1} - 2\right)$$
we have
$$\begin{aligned} T(n) &= 2(n+1)H_{n+1} - (4n+2) \\ &\approx 2(n+1)(\ln(n+1)+\gamma) - (4n+2) \\ &\in \Theta(n \log n) \end{aligned}$$

- Conclusion:
Quicksort AC running time in $\Theta(n \log n)$.

Quicksort space requirement:

- Not an in-space sorting algorithm, because
  - extra space required for all subproblems on the stack
  - in the worst case, there can be $\Theta(n)$ subproblems on stack

Quicksort improvements:

- Split key selection, instead of $A[n]$
  - use $A[\frac{n+1}{2}]$
  - use median of $A[1], A[\frac{n+1}{2}], A[n]$
  - randomized: randomly choose one from $A[1..n]$
    * say $A[j]$
    * swap $A[j] \leftrightarrow A[n]$
    * normal Quicksort (using $A[n]$ as the split key)
- Small sublists:
  - Use insertion sort
  - Can determine the best crossover size is about 20

    can you?

Sorting Algorithms So Far: Running Time Comparison

Alg.	BC	WC	AC
InsertionSort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$
SelectionSort			
BubbleSort			
MergeSort	$\Theta(n \log n)$	$\Theta(n \log n)$	?
HeapSort	$\Theta(n \log n)$	$\Theta(n \log n)$	?
QuickSort	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n \log n)$

- How to get these running times?
- Identify the BC/WC/AC cases for them.

  For example, what is the best case array for QuickSort when $n = 15$?
- How to modify HeapSort to have best case running time in $\Theta(n)$?

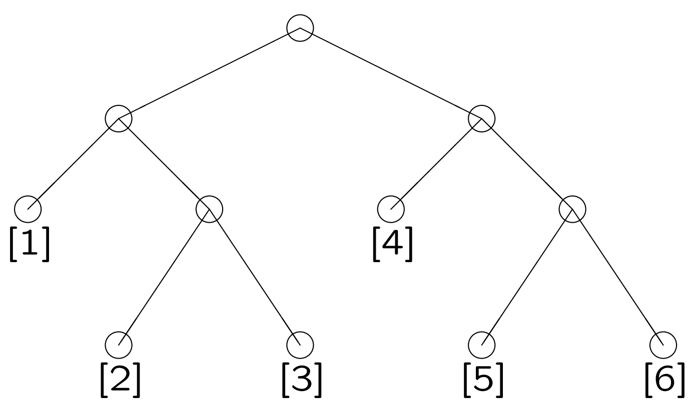
Two useful trees in algorithm analysis:

- Recursion tree
  - node $\longleftrightarrow$ recursion call
  - describes algorithm execution for one particular input by showing all calls made
  - one algorithm execution $\longleftrightarrow$ all nodes (a tree)
  - useful in analysis: sum number of operations over all nodes
- Decision tree
  - node $\longleftrightarrow$ algorithm decision
  - describes algorithm execution for all possible inputs by showing all possible algorithm decisions
  - one algorithm execution $\longleftrightarrow$ one root-to-leaf path
  - useful in analysis: sum number of operations over nodes on one path

Recursion tree example:

- Merge sort pseudocode

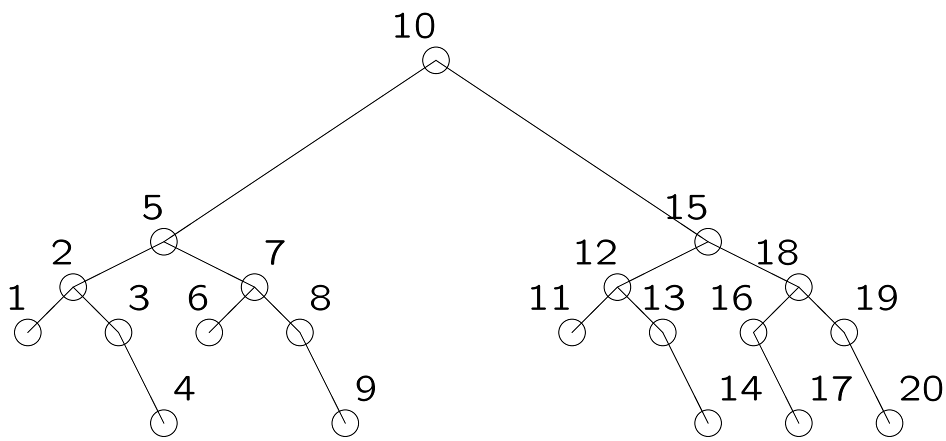
```
Merge(A; lo, mid, hi)    **p 29
       **pre-condition:  lo ≤ mid ≤ hi
       **pre-condition:  A[lo, mid] and A[mid + 1, hi] sorted
       **post-condition:  A[lo, hi] sorted


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)
```



- For different input instance, the number of operations at each node could be different.

Binary search decision tree:

- Assume input keys in array $A[1..20]$
- Tree node $\longleftrightarrow$ "3-way key comparison $<,=,>$?
- Node label $A[j]$
- WC number of KC: 5 (in general $1 + \lfloor \lg n \rfloor$)



- AC number of KC:

  Ask input distribution?

  - target in the array, each location equiprobable:
    $\frac{1}{20} \times (2^0 \times 1 + 2^1 \times 2 + 2^2 \times 3 + 2^3 \times 4 + 5 \times 5) = 3.7$
  - target not in the array, each gap equiprobable:
    $\frac{1}{21} \times (11 \times 4 + 10 \times 5) = 4.5$
  - Both distribution:
    $T(n = 2^k - 1) = \lfloor \lg n \rfloor + \frac{1}{2}$

Have you understood the lecture contents?

well	ok	not-at-all	topic
☐	☐	☐	quicksort AC running time
☐	☐	☐	quicksort space requirement
☐	☐	☐	quicksort improvements
☐	☐	☐	randomized quicksort
☐	☐	☐	recursion tree
☐	☐	☐	decision tree
☐	☐	☐	difference between them