Data selection. Lower complexity bound for sorting

Lecturer: Georgy Gimel’farb

COMPSCI 220 Algorithms and Data Structures
1. Data selection: Quickselect
2. Lower complexity bound for sorting
3. The worst-case complexity bound
4. The average-case complexity bound
5. Lower sorting complexity under additional constraints
Data Selection vs. Data Sorting

- **Selection**: finding only the \( k \)th smallest element, called the element of **rank** \( k \), or the \( k \)th **order statistic** in a list of \( n \) items.

- Main question: can selection be done faster without sorting?

Quickselect: the average \( \Theta(n) \) and worst-case \( \Theta(n^2) \) complexity

1. If \( n = 0 \) or 1, return “not found” or the list item, respectively.
2. Otherwise, choose one of the list items as a pivot, \( p \), and partition the list into disjoint “head” and “tail” sublists with \( j \) and \( n - j - 1 \) items, respectively, separated by \( p \) at position with index \( j^a \).
3. Return the result of quickselect on the head if \( k < j \); the element \( p \) if \( k = j \), or the result of quickselect on the tail otherwise.

\(^a\)All head (tail) items are less (greater) than the pivot \( p \) and precede (follow) it.
Theorem 2.33: The average-case time complexity of quickselect is linear, or \( \Theta(n) \).

Proof. Up to \( cn \) operations to partition the list into the head and tail sublists of size \( j \) and \( n - 1 - j \), respectively, where \( 0 \leq j \leq n - 1 \).

- As in quicksort, each final pivot index \( j \) with equal probability \( \frac{1}{n} \).
- Average time \( T(n) \) to select the \( k^{th} \) smallest item out of \( n \) items:
  \[
  T(n) = \frac{1}{n} \sum_{j=0}^{n-1} \frac{T(j) + T(n-j-1)}{2} + cn = \frac{1}{n} \sum_{j=0}^{n-1} T(j) + cn.
  \]
- Therefore, \( nT(n) = \sum_{j=0}^{n-1} T(j) + cn^2 \).
  - \( nT(n) - (n - 1)T(n - 1) = T(n - 1) + c(2n - 1) \), or
  - \( T(n) \approx T(n - 1) + c' \), so that \( T(n) \in \Theta(n) \).
Implementation of Quickselect

**algorithm** quickSelect

*finds $k^{th}$ smallest element in the subarray $a[l..r]$*

**Input:** array $a[0..n-1]$; array indices $l, r$; integer $k$

**begin**

if $l \leq r$ then

$i \leftarrow \text{pivot}(a, l, r)$

$j \leftarrow \text{partition}(a, l, r, i)$

$q \leftarrow j - l + 1$

if $k = q$ then return $a[j]$

else if $k < q$ then return quickSelect($a, l, j - 1, k$)

else return quickSelect($a, j + 1, r, k - q$)

end if

else return “not found”

end
Sorting by Pairwise Comparisons: Decision Tree

Representing any sorting of $n$ items by pairwise comparisons with a binary decision tree having $n!$ leaves (internal nodes: comparisons).

Decision tree for $n = 3$
Sorting by Pairwise Comparisons: Decision Tree

- Each leaf $ijk$:
  a sorted array $a_i, a_j, a_k$ obtained from the initial list $a_1, a_2, a_3$.

- Each internal node:
  a pairwise comparison $i:j$ between the elements $a_i$ and $a_j$.
    - Two downward arcs: two possible results: $a_i \geq a_j$ or $a_i < a_j$.

- Any of $n!$ permutations of $n$ arbitrary items $a_1, \ldots, a_n$ may be met after sorting: so the decision tree must have at least $n!$ leaves.

- The path length from the root to a leaf is equal to the total number of comparisons for getting the sorted list at the leaf.
  - The longest path (the tree height) is equal to the worst-case number of comparisons.
  - Example: 3 items are sorted with no more than 3 comparisons, because the height of the tree for $n=3$ is equal to 3.
Sorting by Pairwise Comparisons: Decision Tree

\[ a = (35, 10, 17) \]

**Example:**
\[ a_1 = 35, a_2 = 10, a_3 = 17 \]

1. **Comparison 1 : 2**
   \((35 > 10) \rightarrow \) left branch \( a_1 > a_2 \)

2. **Comparison 2 : 3**
   \((10 < 17) \rightarrow \) right branch \( a_2 < a_3 \)

3. **Comparison 1 : 3**:
   \((35 > 17) \rightarrow \) left branch \( a_1 > a_3 \)

4. **Sorted array** 231 →
   \( a_2 = 10, a_3 = 17, a_1 = 35 \)
The Worst-case Complexity Bound

**Lemma:** A decision tree of height $h$ has at most $2^h$ leaves.

**Proof:** by mathematical induction.

- **Base cases:** A tree of height 0 has at most $2^0$ leaves (i.e. one leaf).
- **Hypothesis:** Let any tree of height $h - 1$ have at most $2^{h-1}$ leaves.
- **Induction:**
  - Any tree of height $h$ consists of a root and two subtrees of height at most $h - 1$ each.
  - The number of leaves in the whole decision tree of height $h$ is equal to the total number of leaves in its subtrees, that is, at most $2^{h-1} + 2^{h-1} = 2^h$. □
The Worst-case Complexity Bound

**Theorem 2.35** (Textbook): Every pairwise-comparison-based sorting algorithm takes $\Omega(n \log n)$ time in the worst case.

**Proof:**

- Each binary tree, as shown in Slide 9, has at most $2^h$ leaves.
- The least height $h$ such that $2^h \geq n!$ has the lower bound $h \geq \lg(n!)$.
- By the Stirling’s approximation, $n! \approx n^n e^{-n} \sqrt{2\pi n}$ as $n \to \infty$.
- Therefore, asymptotically, $\lg(n!) \approx n \lg n - 1.44n$, or $\lg(n!) \in \Omega(n \log n)$.

Therefore, heapsort and mergesort have the asymptotically optimal worst-case time complexity for comparison-based sorting.
The Average-case Complexity Bound

**Theorem 2.36** (Textbook): Every pairwise-comparison-based sorting algorithm takes \( \Omega(n \log n) \) time in the average case.

*Proof:* Let \( H(k) \) be the sum of all heights of \( k \) leaves in a balanced decision tree with equal numbers, \( \frac{k}{2} \), of leaves on the left and right subtrees.

- Such a tree has the smallest height, i.e., in any other decision tree, the sum of heights cannot be smaller than \( H(k) \).
- \( H(k) = 2H \left( \frac{k}{2} \right) + k \) as the link to the root adds one to each height, so that \( H(k) = k \lg k \).
- When \( k = n! \) (the number of permutations of an array of \( n \) keys), \( H(n!) = n! \lg(n!) \).
- Given equiprobable permutations, the average height of a leaf is \( H_{\text{avg}}(n!) = \frac{1}{n!} H(n!) = \lg(n!) \approx n \lg n - 1.44n \).
- Thus, the lower bound of the average-case complexity of sorting \( n \) items by pairwise comparisons is \( \Omega(n \log n) \).
Counting Sort – Exercise 2.7.2 (Textbook)

Input: an integer array $a_n = (a_1, \ldots, a_n)$; each $a_i \in Q = \{0, \ldots, Q - 1\}$.

- Make a counting array $t_Q$ and set $t_q \leftarrow 0$ for $q \in Q$.
- Scan through $a_n$ to accumulate in the counters $t_q$; $q \in Q$, how many times each item $q$ is found: if $a_i = q$, then $t[q] \leftarrow t[q] + 1$.
- Loop through $0 \leq q \leq Q - 1$ and output $t_q$ copies of $q$ at each step.

Linear worst- and average-case time complexity, $\Theta(n)$ when $Q$ is fixed.

- $Q + n$ elementary operations to first set $t_Q$ to zero; count then how many times $t_q$ each item $q$ is found in $a_n$, and successively output the sorted array $a_n$ by repeating $t_q$ times each entry $q$.

Theorems 2.35 and 2.36 do not hold under additional data constraints!