CH-231-A

Algorithms and Data Structures

ADS

Lecture 17

Dr. Kinga Lipskoch

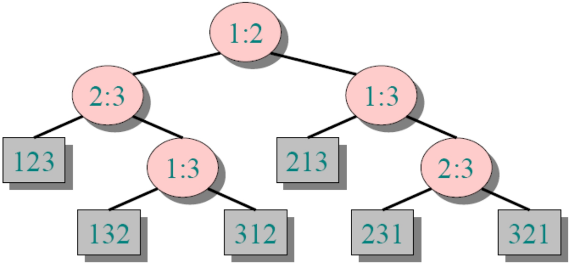
Spring 2022

Comparison Sorts

- All sorting algorithms we have seen so far are comparison sorts.
- A comparison sort only uses comparisons to determine the relative order of elements.
- The best worst-case running time we encountered for comparison sorting was $O(n \lg n)$.
- Is $O(n \lg n)$ the best we can do?

Decision Tree (1)

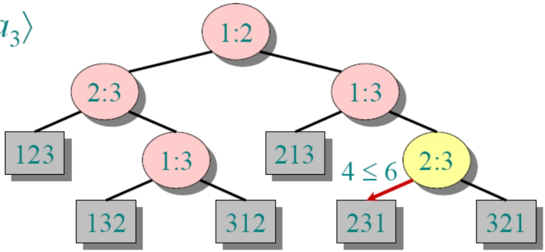
- Sort $< a_1, a_2, ..., a_n >$
- Each internal node is labeled $i : j$ for $i, j \in \{1, 2, ..., n\}$.
- Left subtree shows subsequent comparisons if $a_i \leq a_j$.
- Right subtree shows subsequent comparisons if $a_i \geq a_j$.



Decision Tree (2)

Example:
Each leaf contains a permutation $< \pi(1), \pi(2), \ldots, \pi(n) >$ indicating the order $a_{\pi(1)} \leq a_{\pi(2)} \leq \ldots \leq a_{\pi(n)}$.

Sort $\langle a_1, a_2, a_3 \rangle = \langle 9, 4, 6 \rangle$:



Decision Tree Model

A decision tree can model the execution of any comparison sort:

- One tree for each input size n.
- View the algorithm as splitting whenever it compares two elements.
- The tree contains the comparisons along all possible instruction traces.
- The running time of the algorithm = the length of the path taken.
- Worst-case running time = height of tree.

Decision Tree Sorting

Theorem:

Any decision tree that can sort n elements must have height $\Omega(n \lg n)$.

Proof:

The tree must contain $\geq n!$ leaves,
since there are $n!$ possible permutations.
A height-h binary tree has $\leq 2^h$ leaves.
Thus, $n! \leq 2^h$.
Then,

$$
\begin{aligned}
h &\geq \lg(n!) \\
&\geq \lg((n/e)^n) \\
&= n \lg n - n \lg e \\
&= \Omega(n \lg n).
\end{aligned}
$$

Used Stirling's formula: $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$ when $n \to \infty$.

Lower Bound for Comparison Sorting

- The lower bound for comparison sorting $\Omega(n \lg n)$.
- Heap Sort and Merge Sort are asymptotically optimal comparison sorting algorithms.

Non-Comparison Sorting?

- Is it possible to avoid comparisons between elements?
- Yes, if we can make assumptions on the input data.
- E.g., trivial case:
  - Input: $A[1...n]$, where $A[j] \in \{1, 2, ..., n\}$, and $A[i] \neq A[j]$ for all $i \neq j$
  - Output: $B[1...n]$

Counting Sort: Problem Statement

- Input: $A[1...n]$, where $A[j] \in \{1, 2, ..., k\}$.
- Output: $B[1...n]$, which is a sorted version of $A[1...n]$.
- Auxiliary storage: $C[1...k]$.

Counting Sort

```
for i := 1 to k do
  C[i] := 0
for j := 1 to n do
  C[A[j]] := C[A[j]] + 1
  // C[i] = |{key = i}|
for i := 2 to k do
  C[i] := C[i] + C[i - 1]
  // C[i] = |{key <= i }|
for j := n downto 1 do
  B[C[A[j]]] = A[j]
  C[A[j]] = C[A[j]] - 1
```

Counting Sort: Example (1)

Loop 1:

	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	0	0	0	0

B:					

for $i \leftarrow 1$ **to** k
  do $C[i] \leftarrow 0$

Counting Sort: Example (2)

Loop 2:

	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	1	0	2	2

B:					

$$\textbf{for } j \leftarrow 1 \textbf{ to } n$$
$$\quad \textbf{do } C[A[j]] \leftarrow C[A[j]] + 1 \qquad \triangleright\ C[i] = |\{\text{key} = i\}|$$

Counting Sort: Example (3)

Loop 3:

	1	2	3	4	5
A:	4	1	3	4	3
B:					

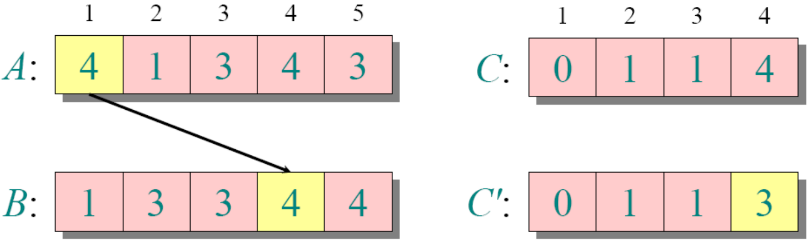
	1	2	3	4
C:	1	0	2	2
C':	1	1	3	5

for $i \leftarrow 2$ **to** k

  do $C[i] \leftarrow C[i] + C[i-1]$ $\triangleright\ C[i] = |\{\text{key} \leq i\}|$

Counting Sort: Example (4)

Loop 4:



for $j \leftarrow n$ **downto** 1

  do $B[C[A[j]]] \leftarrow \mathrm{A}[j]$

    $C[A[j]] \leftarrow C[A[j]] - 1$

Counting Sort: Asymptotic Analysis (1)

$\Theta(k)$

```
for i := 1 to k
    do C[i] := 0
```

$\Theta(n)$

```
for j := 1 to n
    do C[A[j]] := C[A[j]] + 1
```

$\Theta(k)$

```
for i := 2 to k
    do C[i] := C[i] + C[i-1]
```

$\Theta(n)$

```
for j := n downto 1
    do B[C[A[j]]] ← A[j]
       C[A[j]] ← C[A[j]] - 1
```

$\Theta(n+k)$

Counting Sort: Asymptotic Analysis (2)

- If $k = O(n)$, then Counting Sort takes $\Theta(n)$ time.
- Comparison sorting takes $\Omega(n \lg n)$ time.
- Counting Sort is not a comparison sort, not a single comparison between elements occurs.

Stable Sorting

- **Definition**:
  Stable sorting algorithms maintain the relative order of records with equal keys (i.e., values).
- Thus, a sorting algorithm is stable, if whenever there are two records R and S with the same key and with R appearing before S in the original list, R will appear before S in the sorted list.
- Is Counting Sort stable?