Sorting in Linear Time

Alexandre David
B2-206
Linear Sort? But…

- Best algorithms so far perform in $\Theta(n \lg n)$. But they are **comparison** sorts.
  - We do not inspect the value or use other information.
  - Only comparisons between keys.
- Comparison sorts need at least $\Omega(n \lg n)$. Previous algorithms were **optimal**!
Optimality

- How to prove that $n \log n$ is the lower bound for all possible comparison sort algorithms?
- Use decision-tree.
  - Binary trees representing comparisons.
  - All possible permutations represented.
  - $n!$ permutations, thus $n!$ leaves.
  - Sorting algorithms find an ordering, i.e., a path.
Decision Tree

“The tree represents the comparisons done by a sorting algorithm.”

Node element i:j

\[ \leq \quad > \]


Comparisons between all (needed) pairs.

2:3 \leq 1:3 > 2:3 

\[ \langle 1,2,3 \rangle \]

\[ \langle 1,3,2 \rangle \]

\[ \langle 3,1,2 \rangle \]

\[ \langle 2,1,3 \rangle \]

\[ \langle 3,2,1 \rangle \]

\[ \langle 2,3,1 \rangle \]
The point: Lower bound on the heights of all decision trees = lower bound of running time of any comparison sort algorithm.

Because to find one ordering one must go on a path from the root to a leaf.

Every sort algorithm has a different decision tree ⇒ bound all of them!

Bound on the height.
Leaves = permutations ↦ \( n! = \text{leaves} \quad \Rightarrow \quad h \geq \lg(n!) = \Omega(n \lg n) \)
Optimality

Conclusion:

- Any correct algorithm must go through $\Omega(n \lg n)$ to produce any ordering.
- We have sorting algorithms that have a bound of $O(n \lg n)$.
- These (comparison) sorting algorithm are optimal!
- You can’t do better. If you do better, then your algorithm cannot generate all the permutations and is incorrect.
Counting Sort

- Assume that the input is made of integers with a small range (k):
  - $0 \leq a_1..n \leq k$.
  - When $k = O(n)$, counting-sort runs in $\Theta(n)$.

- Idea: For every x
  - count how many elements are $\leq x$, say $t$,
  - put x at position $t$, the right position.
Counting Sort

\[\text{Count\_sort}(A,B,k): \quad \text{// } B = \text{output} \]
\[\text{for } i = 0 \text{ to } k \text{ do } C[i] = 0 \quad \text{// initialize} \]
\[\text{for } i = 1 \text{ to } \text{length}(A) \text{ do } C[A[i]]++ \quad \text{// count } x_i \]
\[\text{for } i = 1 \text{ to } k \text{ do } C[i] += C[i-1] \quad \text{// count } \leq x_i \]
\[\text{for } i = \text{length}(A) \text{ downto } 1 \text{ do} \]
\[\quad B[C[A[i]]] = A[i] \quad \text{// write } x \text{ at } t \]
\[\quad C[A[i]]-- \quad \text{// update counter} \]
\[\text{done} \]

Needs extra memory with k elements: C[0..k] for counting.
Counting Sort

Count_sort(A,B,k):
for i = 0 to k do C[i] = 0
for i = 1 to length(A) do C[A[i]]++
for i = 1 to k do C[i] += C[i-1]
for i = length(A) downto 1 do
  B[C[A[i]]] = A[i]
  C[A[i]]--
done

A: 2 5 3 0 2 3 0 3

C:
0 1 2 3 4 5
0 0 0 0 0 0
2 0 2 3 0 1
2 2 4 7 7 8
2 2 4 7 7 8
2 2 4 6 7 8
1 2 4 6 7 8
1 2 4 5 7 8

B: 0 3 3
Counting Sort

- Running time \( T(n) = \Theta(n+k) \).
  - For \( k = O(n) \), \( T(n) = \Theta(n) \).
- There is no comparison.
- The sort is **stable** (order kept for \( a_i = a_j \)).
- The sort is **not** in-place.
- **Problem**: Range of numbers translates into the size of the working array (counters).

```plaintext
Count_sort(A,B,k):
for i = 0 to k do C[i] = 0
for i = 1 to n do C[A[i]]++
for i = 1 to k do C[i] += C[i-1]
for i = n downto 1 do
  B[C[A[i]]] = A[i]
  C[A[i]]--
done
```
Radix Sort

- Sort on the digits of the numbers.
  - Least significant digits first.
  - Use a **stable** sort (like counting sort).

Radix_sort\((A, d)\):

\[
\text{for } i = 1 \text{ to } d \text{ do}
\]

  \[
  \text{stable_sort } A \text{ with keys=digit } i
  \]

\[
\text{done}
\]

- Sort n-digits numbers with each digit taking k values (base k), running time is \(T(n)=\Theta(d(n+k))\).
Radix Sort
Bucket Sort

- Assume the input is uniformly distributed.
- Assume values in \([0,1)\).

\[
\text{Bucket\_sort}(A):
\begin{align*}
& n = \text{length}(A) \\
& \text{for } i = 1 \text{ to } n \text{ do insert } A[i] \text{ into list } B[n*A[i]] \\
& \text{for } i = 0 \text{ to } n-1 \text{ do insertion\_sort list } B[i] \\
& \text{concatenate lists } B[0], B[1], \ldots, B[n-1]
\end{align*}
\]

- Running time: \( T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2) \)
- Expected running time: \( \Theta(n) \).
Bucket Sort

- Expected time:
  - Use $E[n_i^2] = 2 - 1/n$.
    See book for technicality.

$$E[T(n)] = \Theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2]) = \Theta(n)$$

- Idea: $n$ elements distributed uniformly in $n$ entries $\Rightarrow$ 1 element per entry in average. But not $O(1)$ for sorting...
Bucket Sort

A: 0.78
   0.17
   0.39
   0.26
   0.72
   0.94
   0.21
   0.12
   0.23
   0.68

Hash table

B:

0 /
1 -> 0.12 -> 0.17 /
2 -> 0.21 -> 0.23 -> 0.26 /
3 -> 0.39 /
4 /
5 /
6 -> 0.68 /
7 -> 0.72 -> 0.78 /
8 /
9 -> 0.94 /