LECTURE 7

Today

- $\Omega(n \log n)$ lower bound on sorting
- $O(n)$ algorithm for sorting!
- Conclusion: Importance of modelling

Review

- Have seen many algorithms for sorting
  - INSERTION SORT, MERGE SORT, QUICK SORT,
    - RANDOMIZED QUICK SORT ...
- Running times range from $\Theta(n \log n)$ to $\Theta(n^2)$. 

Can we get even faster algorithms?

Today: Two answers to this question.

**NO:** if algorithm only allowed to compare two array elements in one step.

**YES:** if algorithm allowed to use array elements in other ways (look at specific bits, use them as pointers...), and array elements are small!
**Comparison-Based Algorithm**

**Definition:** An algorithm that sorts an array $A[1..n]$ is a comparison-based algorithm if the only accesses it makes to elements of $A$ is to compare them (a copy, exchange).

- Insertion Sort
- Merge Sort
- Quick Sort
- Randomized QuickSort

Are all of these algorithms comparison-based?
Decision Tree Representation of Algorithm

- An algorithm to sort can be represented by a TREE of comparisons.
- At leaf of TREE output represented by PERMUTATION.
- Depth of TREE is upper bounded by running time of algorithm.
To sort $a_1, a_2, a_3$, an algorithm may first compare $a_1 < a_2$? Then compare $a_2 < a_3$? Finally (if not already known) compare $a_1 < a_3$.

Example

```
1:2

2:3

1, 2, 3

2:3

1:3

2, 1, 3

3, 2, 1

1, 3, 2

3, 1, 2

2, 3, 1
```
Worst-case: Makes 3 comparisons.

Could we have done better?

No! Any algorithm to sort 3 integers needs to have at least 6 leaves.

But tree is binary.

Depth 2 tree has at most 4 leaves.

So every tree sorting 3 integers must have some path of depth 3.

Food for thought: Does answer change if comparison is 3-way \((<, =, >)\)?
Generalizing to \( n \) elements.

- Every algorithm for sorting \( n \) elements must have \( n! \) leaves.

- Must have depth \( \log(n!) = ? \)

\[
\left( \text{use} \left( \frac{n}{2} \right) \right)^{n/2} \leq n! \leq n^n
\]

\[
\Rightarrow \quad \frac{n}{2} \log \frac{n}{2} \leq \log(n!) \leq n \log n
\]

\[
\Rightarrow \quad \log n! = \Theta(n \log n)
\]

**Theorem:** Comparison-based sorting algorithms require \( \Omega(n \log n) \) time.
Counting Sort

- Sorts $n$ elements in range $[1, \ldots, k]$ in time $\Theta(n+k)$
- Uses array elements as array indices.
- Not allowed in comparison-based algorithms.

Counting Sort

1. $1^{st}$ Pass: Compute array $C[1..k]$, $C[j] = \# i \ s.t. \ A[i] = j$.
2. $2^{nd}$ Pass: Compute array $D[1..k]$, $D[j] = \# i \ s.t. \ A[i] \leq j$.
3. $3^{rd}$ Pass: Use $D$ to sort $A$ into $B$.
Pseudo Code for

Counting Sort \( (A[1..n]) \);

1. For \( j = 1 \) to \( k \) do
   \[ C[j] \leftarrow 0; \]

2. For \( i = 1 \) to \( n \) do
   \[ \text{Increment } C[A[i]]; \]

3. \[ D[1] \leftarrow C[1]; \]
   For \( j = 2 \) to \( k \) do
   \[ D[j] \leftarrow D[j-1] + C[j]; \]

4. For \( i = n \) down to \( 1 \) do
   \[ B[D[A[i]]] \leftarrow A[i]; \]
   \[ \text{Decrement } D[A[i]] \]
Example

\[ A = \begin{bmatrix} 1 & 1 & 2 & 3 & 1 & 5 & 2 \end{bmatrix} \]

\[ C = \begin{bmatrix} 3 & 2 & 1 & 0 & 1 \end{bmatrix} \]

\[ D = \begin{bmatrix} 3 & 5 & 6 & 6 & 7 \end{bmatrix} \]

\[ B = \begin{bmatrix} 1 & 1 & 1 & 1 & 2 & 2 & 3 & 5 \end{bmatrix} \]
Notes:

- In our implementation, it is **stable**.

**Definition:** Algorithm is **stable** if it does not invert elements with equal key.

- If \( k = o(n \log n) \), runs in \( o(n \log n) \) time. In particular, runs in \( \Theta(n) \) time if \( k = \Omega(n) \).

- But is this the best we can do? Yes ... with **Radix Sort / Bucket Sort**.
Radix Sort

- \( \Theta(cn) \) time algorithm to sort integers in the range \( \{1, 2, \ldots, n^c\} \).

- Idea: Think of array elements as \( c \)-digit elements where digit in \( \{1, \ldots, n\} \).

- Key Question: How to sort digits by digit?
  
  Naive Idea: Sort most significant digit first...

  Correct Idea: Sort from least significant digit to most significant digit using STABLE sorting algorithm.
Pseudo Code

Notation: \( A[i].k = \text{\( k^{th} \) digit of } A[i] \)

\( k=1 \) least significant

\( k=c \) most significant

\text{StableSort} (A, k): \text{Sorts } A \text{ in place with key } = A[i].k

\text{RadixSort} (A, n, c)

\text{for } k=1 \text{ to } c \text{ do }

\text{StableSort} (A, k);

\text{Return} (A);
Example:

```
329  720  720  329
457  355  329  355
657  436  436  436
839  457  839  457
436  657  355  657
720  329  457  720
839  657  839  839
```


- $k < l \Rightarrow$ doesn't matter
- $k = l \Rightarrow$ relative ordering correct
- $k > l \Rightarrow$ keys agree: STABLE $\Rightarrow$ ordering does not change
Notes:

- Invented by Herman Hollerith (prototyped and punched cards for computers also).
- Speeded up the 1890 US Census from ~10 years to ~6 weeks.
- Radix Sorting machines still exist and are used daily by.....USPs!