Algorithms: Sorting

Amotz Bar-Noy

CUNY
The Sorting Problem

Model:
- **Keys**: Entities from a well ordered domain.
  - Usually numbers.
- **Comparisons**: Between two keys $K_1$ and $K_2$
  - $K_1 < K_2$? $K_1 \leq K_2$? $K_1 = K_2$?

Task:
- **Input**: An unsorted array of $n \geq 1$ keys $A[1], A[2], \ldots, A[n]$.

Optimization Goal
- Minimize the number of comparisons between keys.
Introduction

Complexity of the Sorting Problem

**Lower bound**
- \( \Omega(n \log(n)) \) comparisons are required by any algorithm.

**Upper bounds**
- \( O(n^2) \) comparisons with "simple" algorithms.
- \( O(n \log(n)) \) comparisons with more "sophisticated" algorithms.

**Tight bound**
- \( \Theta(n \log(n)) \) overall complexity.

**Remark**
- Bounds are for both the worst case complexity and the average case complexity.
Some Sorting Algorithms

Simple algorithms
- **Bubble-Sort**: $\Theta(n^2)$ worst & average case.
- **Insertion-Sort**: $\Theta(n^2)$ worst & average case.

Efficient deterministic sorting algorithms
- **Merge-Sort**: $\Theta(n \log(n))$ worst & average case.
- **Heap-Sort**: $\Theta(n \log(n))$ worst & average case.
- **Balanced-Tree-Sort**: $\Theta(n \log(n))$ worst & average case.

Efficient randomized sorting algorithms
- **Quick-Sort**: $\Theta(n \log(n))$ average case; $\Theta(n^2)$ worst case.
- **Binary-Tree-Sort**: $\Theta(n \log(n))$ average case; $\Theta(n^2)$ worst case.
Bubble Sort

Input
- An unsorted array of \( n \) keys \( A[1], A[2], \ldots, A[n] \).

Ideas
- Find the minimum \( n - 1 \) times.
- Compare and exchange only adjacent keys.

Pseudocode
- \textbf{Bubble-Sort}(A[1], \ldots, A[n])
  - for \( i = 1 \) to \( n - 1 \)
    - for \( j = n \) downto \( i + 1 \)
        - then \( A[j] \leftrightarrow A[j - 1] \)
Example

Initial array: [8, 21, 1, 3, 2, 13, 5]

After round 1: [1, 8, 21, 2, 3, 5, 13]

After round 2: [1, 2, 8, 21, 3, 5, 13]

After round 3: [1, 2, 3, 8, 21, 5, 13]

After round 4: [1, 2, 3, 5, 8, 21, 13]

After round 5: [1, 2, 3, 5, 8, 13, 21]

After round 6: [1, 2, 3, 5, 8, 13, 21] (* the array is sorted *)
Correctness and Complexity

Correctness
- By induction, for $1 \leq i \leq n - 1$, after round $i$:
  - $A[i] \leq A[j]$ for all $i < j \leq n$

Complexity
- For $1 \leq i \leq n - 1$, in round $i$: exactly $n - i$ comparisons.
- The total number of comparisons is always
  \[
  (n - 1) + (n - 2) + \cdots + 1 = \frac{n(n - 1)}{2} = \Theta(n^2)
  \]
Merge-Sort

Input
- An unsorted array of \( n \) keys \( A[1], A[2], \ldots, A[n] \).

Divide and Conquer
- For \( n \geq 2 \) and \( q = \left\lfloor \frac{n+1}{2} \right\rfloor \), recursively **sort** the sub-arrays \( A[1..q] \) and \( A[q + 1..n] \).
- **Merge** the sub-arrays \( A[1..q] \) and \( A[q + 1..n] \) into a sorted array \( A[1..n] \).
### The Merge Procedure

**Global array**
- $A[1], A[2], \ldots, A[n]$

**Procedure**
- **Merge**($\ell, q, r$) (*$1 \leq \ell \leq q < r \leq n$*)
  - *Copy* the sorted subarray $A[\ell..q]$ to an array $L[\ell..q]$.
  - *Copy* the sorted subarray $A[(q + 1)..r]$ to an array $R[(q + 1)..r]$.
  - *Merge* the two sorted sub-arrays $L$ and $R$ into a sorted sub-array $A[\ell] \leq \cdots \leq A[r]$.

**Complexity**
- Number of comparisons is at most $(r - \ell)$.
The Recursive Merge-Sort Procedure

Initial recursive call

- **Merge-Sort**(1, n).

Recursive procedure

- **Merge-Sort**(ℓ, r)
  
  - if \( r > ℓ \) then
    
    - \( q = \lfloor \frac{ℓ+r}{2} \rfloor \)  
    
    - (** \( ℓ \leq q < q + 1 \leq r \) **)
    
    - **Merge-Sort**(ℓ, q)
    
    - **Merge-Sort**(q + 1, r)
    
    - **Merge**(ℓ, q, r)
Example

8, 21, 1, 34, 3, 2, 13, 5

8, 21, 1, 34

8, 21

8, 21

1, 34

1, 34

1, 8, 21, 34

1, 8, 21, 34

2, 3

2, 3

2, 3, 5, 13

2, 3, 5, 13

1, 2, 3, 5, 8, 13, 21, 34

1, 2, 3, 5, 8, 13, 21, 34
Correctness

Proof sketch

- Proof by induction on $r - \ell$.
- Case $r = \ell$ the array is sorted trivially.
- Case $\ell \leq q < r$, the induction hypothesis is true:
  - For sub-array $A[\ell..q]$ since $q - \ell < r - \ell$.
  - For sub-array $A[(q + 1)\.r]$ since $r - (q + 1) < r - \ell$.
- The correctness of procedure Merge justifies the inductive step.
Complexity

Notation
- \( T(n) \) - upper bound on the number of comparisons for an array with \( n \) keys.

Recursive formula
- \( T(1) = 0 \)
- \( T(n) \leq T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + (n - 1) \)

Master Theorem
- \( T(n) = 2T(n/2) + \Theta(n) \)
- \( a = 2, b = 2, \) and \( d = 1. \)
- \( \log_b(a) = 1 = d. \)
- \( \implies T(n) = \Theta(n^d \log(n)) = \Theta(n \log(n)). \)
Complexity for $n = 2^k$

Bottom-Up evaluation

\[
\begin{align*}
T(1) &= 0 \\
T(2) &\leq 2 \cdot T(1) + (2 - 1) \\&\leq 1 \\
T(4) &\leq 2 \cdot T(2) + (4 - 1) \\&\leq 5 \\
T(8) &\leq 2 \cdot T(4) + (8 - 1) \\&\leq 17 \\
T(16) &\leq 2 \cdot T(8) + (16 - 1) \\&\leq 49 \\
T(32) &\leq 2 \cdot T(16) + (32 - 1) \\&\leq 129
\end{align*}
\]

Guess

\[T(n) \leq n \log_2 n - (n - 1).\]
Guessing by Unfolding the Recursion

**Top-Down evaluation**

\[
T(2^k) \leq 2T(2^{k-1}) + (2^k - 1)
\]

\[
= 2T(2^{k-1}) + (1 \cdot 2^k - 1)
\]

\[
\leq 2(2T(2^{k-2}) + (2^{k-1} - 1)) + (2^k - 1)
\]

\[
= 4T(2^{k-2}) + (2 \cdot 2^k - 3)
\]

\[
\leq 4(2T(2^{k-3}) + (2^{k-2} - 1)) + (2 \cdot 2^k - 3)
\]

\[
= 8T(2^{k-3}) + (3 \cdot 2^k - 7)
\]

\[
\vdots
\]

\[
= 2^i T(2^{k-i}) + (i \cdot 2^k - (2^i - 1))
\]

\[
\vdots
\]

\[
= 2^k T(2^0) + (k \cdot 2^k - (2^k - 1))
\]

\[
= n \log_2 n - (n - 1)
\]
Proof By Induction for $n = 2^k$

**Theorem**
- $T(n) \leq n \log_2 n - (n - 1)$

**Induction base**
- $n = 1$: $T(1) \leq 0 = 1 \cdot 0 - 0 = 1 \log_2 1 - (1 - 1)$

**Induction hypothesis**
- $T(n/2) \leq (n/2) \log_2(n/2) - (n/2 - 1)$

$$T(n/2) = (n/2)(\log_2 n - 1) - (n/2 - 1)$$

$$= (n/2) \log_2 n - (n - 1)$$

**Inductive step**
- $T(n) \leq 2T(n/2) + (n - 1)$

$$\leq 2((n/2) \log_2 n - (n - 1)) + (n - 1)$$

$$= n \log_2 n - (n - 1)$$
Complexity for $n \neq 2^k$

Bottom-Up evaluation

\[
\begin{align*}
T(1) &= 0 \\
T(2) &\leq T(1) + T(1) + (2 - 1) \leq 1 \\
T(3) &\leq T(2) + T(1) + (3 - 1) \leq 3 \\
T(4) &\leq T(2) + T(2) + (4 - 1) \leq 5 \\
T(5) &\leq T(3) + T(2) + (5 - 1) \leq 8 \\
T(6) &\leq T(3) + T(3) + (6 - 1) \leq 11 \\
T(7) &\leq T(4) + T(3) + (7 - 1) \leq 14 \\
T(8) &\leq T(4) + T(4) + (8 - 1) \leq 17 \\
T(9) &\leq T(5) + T(4) + (9 - 1) \leq 21
\end{align*}
\]
Complexity for $n \neq 2^k$

**Guess**

- $T(n) \leq n \lceil \log_2 n \rceil - (2^{\lceil \log_2 n \rceil} - 1)$

**Verification**

- $T(1) = 0 = 1 \lceil \log_2 1 \rceil - (2^{\lceil \log_2 1 \rceil} - 1)$
- $T(2) \leq 1 = 2 \lceil \log_2 2 \rceil - (2^{\lceil \log_2 2 \rceil} - 1)$
- $T(3) \leq 3 = 3 \lceil \log_2 3 \rceil - (2^{\lceil \log_2 3 \rceil} - 1)$
- $T(4) \leq 5 = 4 \lceil \log_2 4 \rceil - (2^{\lceil \log_2 4 \rceil} - 1)$
- $T(5) \leq 8 = 5 \lceil \log_2 5 \rceil - (2^{\lceil \log_2 5 \rceil} - 1)$
- $T(6) \leq 11 = 6 \lceil \log_2 6 \rceil - (2^{\lceil \log_2 6 \rceil} - 1)$
- $T(7) \leq 14 = 7 \lceil \log_2 7 \rceil - (2^{\lceil \log_2 7 \rceil} - 1)$
- $T(8) \leq 17 = 8 \lceil \log_2 8 \rceil - (2^{\lceil \log_2 8 \rceil} - 1)$
- $T(9) \leq 21 = 9 \lceil \log_2 9 \rceil - (2^{\lceil \log_2 9 \rceil} - 1)$
Ceilings of Logarithms

Observations

\[ \lceil \log_2 (k + 1) \rceil = \lceil \log_2 k \rceil \text{ for } k \neq 2^h. \]

\[ \lceil \log_2 (k + 1) \rceil = \lceil \log_2 k \rceil + 1 \text{ for } k = 2^h. \]

\[ \lceil \log_2 (2k) \rceil = \lceil \log_2 k \rceil + 1. \]

\[ \lceil \log_2 (2k + 1) \rceil = \lceil \log_2 k \rceil + 1 \text{ for } k \neq 2^h. \]

\[ \lceil \log_2 (2k + 1) \rceil = \lceil \log_2 k \rceil + 2 \text{ for } k = 2^h. \]
Proof for $n = 2k$

**Theorem**

\[ T(n) \leq n \lceil \log_2 n \rceil - (2^{\lceil \log_2 n \rceil} - 1) \]

**Inductive step**

\[
T(n) \leq 2T(k) + (n - 1) \\
\leq 2(k \lceil \log_2 k \rceil - (2^{\lceil \log_2 k \rceil} - 1)) + (n - 1) \\
= 2k \lceil \log_2 k \rceil + n - 2 \cdot 2^{\lceil \log_2 k \rceil} + 2 - 1 \\
= n(\lceil \log_2 k \rceil + 1) - (2^{\lceil \log_2 k \rceil + 1} - 1) \\
= n \lceil \log_2 n \rceil - (2^{\lceil \log_2 n \rceil} - 1) 
\]
Proof for \( n = 2k + 1 \) and \( k \neq 2^h \)

**Theorem**

\[ T(n) \leq n \lceil \log_2 n \rceil - (2^\lceil \log_2 n \rceil - 1) \]

**Inductive step**

\[
T(n) \leq T(k + 1) + T(k) + (n - 1)
\]
\[
\leq ((k + 1) \lceil \log_2 (k + 1) \rceil - (2^\lceil \log_2 (k + 1) \rceil - 1))
\]
\[
+ (k \lceil \log_2 k \rceil - (2^\lceil \log_2 k \rceil - 1)) + (n - 1)
\]
\[
= (2k + 1) \lceil \log_2 k \rceil + n - 2 \cdot 2^\lceil \log_2 k \rceil + 1
\]
\[
= n(\lceil \log_2 k \rceil + 1) - (2^{\lceil \log_2 k \rceil + 1} - 1)
\]
\[
= n \lceil \log_2 n \rceil - (2^\lceil \log_2 n \rceil - 1)
\]

Amotz Bar-Noy (CUNY)

Algorithms: Sorting
Proof for $n = 2k + 1$ and $k = 2^h$

**Theorem**

$T(n) \leq n \lceil \log_2 n \rceil - (2^{\lceil \log_2 n \rceil} - 1)$

**Inductive step**

$T(n) \leq T(k + 1) + T(k) + (n - 1)$

$\leq ((k + 1) \lceil \log_2 (k + 1) \rceil - (2^{\lceil \log_2 (k+1) \rceil} - 1))$

$+ (k \lceil \log_2 k \rceil - (2^{\lceil \log_2 k \rceil} - 1)) + (n - 1)$

$= (k + 1)(h + 1) - (2k - 1) + kh - (k - 1) + 2k$

$= (2k + 1)h + 3$

$= n(h + 2) - (2n - 3)$

$= n \lceil \log_2 n \rceil - (2^{\lceil \log_2 n \rceil} - 1)$

**Observation**

$2^{\lceil \log_2 n \rceil} = 2^{\lceil \log_2 (2k+1) \rceil} = 2^{h+2} = 4k = 2n - 2$
Quick-Sort

Input
- An unsorted array of \( n \) keys \( A[1], A[2], \ldots, A[n] \).

Divide and Conquer
- For \( n \geq 2 \) and some \( 1 \leq q < n \), Partition the array \( A[1..n] \) into two sub-arrays \( A[1..q-1] \) and \( A[q+1..n] \) such that
  * All the keys in \( A[1..(q-1)] \) are smaller or equal to \( A[q] \).
  * \( A[q] \) is smaller or equal to all the keys in \( A[(q+1)\ldots n] \).
- Recursively, Sort the two sub-arrays \( A[1..q-1] \) and \( A[q+1..n] \) if they have at least 2 keys.
The Partition Procedure

Global array:
- \(A[1], A[2], \ldots, A[n]\).

Pivot partitioning
- Partition(\(\ell, r\)): (* \(\ell < r\) *)
  - A pivot key is compared with the rest of the keys.
  - Return a value \(\ell \leq q \leq r\) such that \(A[i] \leq A[q] \leq A[j]\) for any \(\ell \leq i \leq q - 1\) and \(q + 1 \leq j \leq r\).
  - The pivot key \(A[q]\) is the \((q - \ell + 1)\)-smallest key and the \((r - q + 1)\)-largest key in the sub-array \(A[\ell..r]\).

Complexity
- Number of comparisons is exactly \((r - \ell)\).
The Recursive Quick-Sort Procedure

Initial recursive call:
- \textbf{Quick-Sort}(1, n).

Recursive procedure
- \textbf{Quick-Sort}(\ell, r)
  
  if $r > \ell$ then
  
  $q = \text{Partition}(\ell, r)$
  
  \textbf{Quick-Sort}(\ell, q - 1)
  
  \textbf{Quick-Sort}(q + 1, r)
Example

34, 21, 1, 8, 3, 2, 13, 5

1, 3, 2

1, 2

1, 2

1

3

1, 2

2

3

3

5

5

5

5

34, 21, 8, 13

13

8

13

21

34

1, 2, 3, 5, 8, 13, 21, 34

1, 2, 3, 5, 8, 13, 21, 34
Correctness

Proof

- By induction on $r - \ell$.
- Case $r < \ell$: the array does not exist.
- Case $r = \ell$: the array is sorted trivially.
- Case $\ell < r$ and $\ell \leq q \leq r$ the induction hypothesis is true:
  - For sub-array $A[\ell..(q - 1)]$ since $(q - 1) - \ell < r - \ell$.
  - For sub-array $A[(q + 1)..r]$ since $r - (q + 1) < r - \ell$.
- The inductive step is correct since procedure Partition guarantees that all the keys in $A[\ell..(q - 1)]$ are smaller or equal to all the keys in $A[(q + 1)..r]$. 
**Complexity**

**Notation**
- $T(n)$ - upper bound on the number of comparisons for an array with $n$ keys.

**Recursive formula**
- $T(0) = T(1) = 0$.
- $T(n) \leq T(q - 1) + T(n - q) + (n - 1)$ for $1 \leq q \leq n$.

**Solutions to some special cases**
- **Best:** $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$.
- **Good:** $T(n) = T(n/10) + T(9n/10) + \Theta(n) = \Theta(n \log n)$.
- **Worst:** $T(n) = T(n - 1) + (n - 1) = \Theta(n^2)$. 
Randomized Pivot Selection

Assumption
- Ignore ceilings and floors.

Definition
- A **good pivot** is greater than at least $n/4$ keys and is smaller than at least $n/4$ keys.
- Half of the pivots are good pivots.

Pivot selection:
- Repeat selecting a pivot $A[i]$ for a random $i$ from the range $[1..n]$ until finding a good pivot.
Expected Number of Comparisons

Notation
- \( T(n) \): The expected number of comparisons for Quick-Sort with the randomized pivot selection on arrays of size \( n \).

The recursive formula
- \( \Theta(n) \) comparisons to perform one partition.
- \( \Theta(n) \) comparisons until a good partition is performed.
  - With probability at least \( \frac{1}{2} \) the random pivot is a good pivot.
  - Expected number of random selections to get a good pivot is 2.
  - The expectation of a sum is the sum of expectations.
- \( T(n) \leq T(\varepsilon n) + T((1 - \varepsilon)n) + \Theta(n) \) for \( \frac{1}{4} \leq \varepsilon \leq \frac{3n}{4} \).
- Worst case happens for \( \varepsilon = \frac{1}{4} \). Therefore,
  \[
  T(n) \leq T\left(\frac{3n}{4}\right) + T\left(\frac{n}{4}\right) + \Theta(n) = \Theta(n \log n)
  \]
Solving the Recursion

**Theorem**

- Assume $T(n) \leq T(3n/4) + T(n/4) + \alpha n$ for constant $\alpha > 0$.
- Then $T(n) \leq \beta n \log n$ for constant $\beta > 1.25\alpha$.

**Proof by induction**

\[
T(n) \leq T\left(\frac{3n}{4}\right) + T\left(\frac{n}{4}\right) + \alpha n
\]

\[
\leq \beta \frac{3n}{4} \log_2 \left(\frac{3n}{4}\right) + \beta \frac{n}{4} \log_2 \left(\frac{n}{4}\right) + \alpha n
\]

\[
= \beta \left(\frac{3n}{4} \log_2 \left(\frac{3n}{4}\right) + \frac{n}{4} \log_2 \left(\frac{n}{4}\right)\right) + \alpha n
\]

\[
= \beta \left(\frac{3n}{4} \log_2 n + \frac{n}{4} \log_2 n\right) + \beta \left(\frac{3n}{4} \log_2 \left(\frac{3}{4}\right) + \frac{n}{4} \log_2 \left(\frac{1}{4}\right)\right) + \alpha n
\]

\[
= \beta n \log_2 n - \beta \left(\frac{3}{4} \log_2 \left(\frac{4}{3}\right) + \frac{1}{4} \log_2 4\right) n + \alpha n
\]

\[
= \beta n \log_2 n + \left(\alpha - \beta \frac{1}{2} - \frac{3\beta}{4} \log_2 \left(\frac{4}{3}\right)\right) n
\]
Solving the Recursion

Theorem
- Assume $T(n) \leq T(3n/4) + T(n/4) + \alpha n$ for constant $\alpha > 0$.
- Then $T(n) \leq \beta n \log n$ for constant $\beta > 1.25\alpha$.

Proof continue
- To prove that $T(n) \leq \beta n \log_2 n$ the coefficient of $n$ must be non-positive:
  \[ \alpha \leq \frac{\beta}{2} + \frac{3\beta}{4} \log_2 \left(\frac{4}{3}\right) \]
- Equivalently:
  \[ \beta > \frac{1}{0.5 + 0.75 \log_2(1.333)} \alpha \approx 1.233\alpha \]

Q.E.D.
- Because by assumption $\beta \geq 1.25\alpha$. 

Amotz Bar-Noy (CUNY)
Solving the Recursion

Theorem
- Assume $T(n) \leq T(3n/4) + T(n/4) + \alpha n$ for constant $\alpha > 0$.
- Then $T(n) \leq \beta n \log n$ for constant $\beta > 1.25\alpha$.

Corollary
- $T(n) \leq 2.5n \log_2(n)$.

Proof
- The expected value of $\alpha$ is 2 because the expected number of selections until a good pivot is found is 2 and each selection makes $n – 1$ comparisons.
Another Method to Solve $T(n)$

Pivot selection
- The pivot is $A[i]$ for a random $i$ in the range $[1..n]$.

Arrays with $n$ distinct keys
- For $n \geq 2$, with probability $1/n$ the value of $q$ is $i$ for any $i \in \{1, 2, \ldots, n\}$.

Arrays with arbitrary keys
- The partition may generate smaller arrays for the recursive calls.
- Therefore having non-distinct keys can only help the analysis.

Counting comparisons
- For a random selected $1 \leq q \leq n$, there are $(T(q - 1) + T(n - q))$ comparisons in the recursive calls.
- Procedure **Partition** makes exactly $n - 1$ comparisons.
Another Method to Solve $T(n)$

**Theorem**
- $T(n) = \Theta(n \log(n))$.

**Proof**

- **Initial values:** $T(0) = T(1) = 0$.
- **Recursive formula:**

\[
T(n) = (n - 1) + \frac{1}{n} \sum_{q=1}^{n} (T(q - 1) + T(n - q))
\]

\[
= (n - 1) + \frac{1}{n} \sum_{q=1}^{n} T(q - 1) + \frac{1}{n} \sum_{q=1}^{n} T(n - q)
\]

\[
= (n - 1) + \frac{2}{n} \sum_{q=2}^{n-1} T(q)
\]
Bounding Summations by Integrals

For a **monotonic non-decreasing** function $f(x)$:

$$\int_{\ell-1}^{u} f(x)dx \leq \sum_{i=\ell}^{u} f(i) \leq \int_{\ell}^{u+1} f(x)dx$$

For a **monotonic non-increasing** function $f(x)$:

$$\int_{\ell}^{u+1} f(x)dx \leq \sum_{i=\ell}^{u} f(i) \leq \int_{\ell-1}^{u} f(x)dx$$
Bounding Summations by Integrals

Figure 3.1 Approximation of $\sum_{k=m}^{n} f(k)$ by integrals. The area of each rectangle is shown within the rectangle, and the total rectangle area represents the value of the summation. The integral is represented by the shaded area under the curve. By comparing areas in (a), we get $\int_{m-1}^{n} f(x) \, dx \leq \sum_{k=m}^{n} f(k)$, and then by shifting the rectangles one unit to the right, we get $\sum_{k=m}^{n} f(k) \leq \int_{m}^{n+1} f(x) \, dx$ in (b).
Harmonic Numbers

Definition

\[ H(n) = \sum_{i=1}^{n} \frac{1}{i} = 1 + \sum_{i=2}^{n} \frac{1}{i}. \]

Facts

- \( f(x) = \frac{1}{x} \) is a monotonic non-increasing function.
- \( \int \frac{dx}{x} = \ln(x). \)

Bounds

- Upper Bound:
  \[ H(n) \leq 1 + \int_{1}^{n} \frac{dx}{x} = 1 + \ln(n) - \ln(1) = 1 + \ln(n). \]
- Lower bound:
  \[ H(n) \geq \int_{1}^{n+1} \frac{dx}{x} = \ln(n+1) - \ln(1) = \ln(n+1). \]
- Exact bound:
  \[ H_n = \ln(n) + \gamma + \frac{1}{(2n)} - \frac{1}{(12n^2)} + \epsilon_n/(120n^4) \]
  \[ \text{for } 0 < \epsilon_n < 1 \text{ and } \gamma \approx 0.5772 \text{ (Euler’s constant)}. \]
Harmonic Numbers

Bounding Summations by Integrals

\[ \sum_{i=1}^{n} \frac{1}{i} = \ln n + O(1) \]

\[ \ln (n+1) = \int_1^{n+1} \frac{1}{x} \, dx \leq \sum_{i=1}^{n} \frac{1}{i} \leq 1 + \int_1^{n} \frac{1}{x} \, dx = (\ln n) + 1 \]

Recall: \( \ln 1 = 0 \)
Bounding the Sum $\sum_{q=2}^{n-1} q \ln(q)$

**Facts**
- $f(x) = x \ln(x)$ is a **monotonic non-decreasing** function.
- $\int x \ln(x) \, dx = \frac{x^2 \ln x}{2} - \frac{x^2}{4}$.

**The upper bound**

$$\sum_{q=2}^{n-1} q \ln(q) \leq \int_{2}^{n} x \ln(x) \, dx$$

$$= \left( \frac{n^2 \ln(n)}{2} - \frac{n^2}{4} \right) - \left( \frac{2^2 \ln(2)}{2} - \frac{2^2}{4} \right)$$

$$\leq \frac{1}{2} n^2 \ln(n) - \frac{1}{4} n^2$$
Solving the Recursive Formula

**Theorem**

\[ T(n) \leq 2n \ln(n). \]

**Proof by induction**

\[
T(n) = (n - 1) + \frac{2}{n} \sum_{q=2}^{n-1} T(q) \\
\leq (n - 1) + \frac{4}{n} \sum_{q=2}^{n-1} q \ln(q) \\
\leq (n - 1) + \frac{4}{n} \left( \frac{1}{2} n^2 \ln(n) - \frac{1}{4} n^2 \right) \\
\leq 2n \ln(n)
\]

**Corollary**

\[ \ln(2) = 0.69 \ldots < 0.7 \quad \Rightarrow \quad T(n) \leq 1.4n \log_2(n). \]
A Third Method to Solve $T(n)$

Proof sketch

1. $T(n) = (n - 1) + \frac{2}{n} \sum_{q=2}^{n-1} T(q)$
2. $T(n - 1) = (n - 2) + \frac{2}{n-1} \sum_{q=2}^{n-2} T(q)$
3. $nT(n) - (n - 1) T(n - 1) = (2n - 2) + 2T(n - 1)$
   * $2n - 2 = n(n - 1) - (n - 1)(n - 2)$
   * $2T(n - 1) = 2 \sum_{q=2}^{n-1} T(q) - 2 \sum_{q=2}^{n-2} T(q)$
4. $nT(n) - (n + 1) T(n - 1) = 2n - 2$
5. $\frac{T(n)}{n+1} - \frac{T(n-1)}{n} = \frac{2n-2}{n(n+1)}$
   * After both sides of the equation are divided by $n(n + 1)$
A Third Method to Solve $T(n)$

**Proof sketch continue**

- $S(n) = \frac{T(n)}{n+1}$ and $S(1) = 0$
- $S(n) = S(n - 1) + \frac{2(n-1)}{n(n+1)}$
- $S(n) = \sum_{i=2}^{n} \frac{2(i-1)}{i(i+1)}$
- $S(n) < \sum_{i=2}^{n} \frac{2}{i+1} = 2 \sum_{i=1}^{n-1} \frac{1}{i} = 2H(n - 1)$
- $S(n) < 2(1 + \ln(n))$
- $T(n) = (n + 1)S(n) \leq 2(n + 1) \ln(n) = 2n \ln(n) + 2n + 2 \ln(n) + 2$
- $T(n) = \Theta(n \ln(n))$
A Fourth Method to Solve $T(n)$

**Notations**

- Let the order among the $n$ keys in the array be $K_1 \leq K_2 \leq \cdots \leq K_n$.
- For $1 \leq i < j \leq n$, let $p_{ij}$ be the probability that $K_i$ and $K_j$ are compared during the run of Quick-Sort.

**Lemma**

$$p_{ij} = \frac{2}{j - i + 1}$$

**Corollaries**

- $p_{ij} = 1$ for $j = i + 1$.
- $p_{1n} = 2/n$ for $i = 1$ and $j = n$. 
Proof sketch

- $K_i$ and $K_j$ are compared only if they both belong to the same sub-array of some recursive call. Assume that this is the case.

- If the pivot is $K_h \notin \{K_i, \ldots, K_j\}$, then both $K_i$ and $K_j$ remain in the same sub-array of the next recursive calls without being compared.

- If one of $K_{i+1}, \ldots, K_{j-1}$ is selected as a pivot, then $K_i$ and $K_j$ would never be compared because in future recursive calls each belongs to another sub-array.

- If either $K_i$ or $K_j$ is selected as a pivot, then $K_i$ is compared with $K_j$.

- The lemma follows, since there are $j - i + 1$ keys in the set $\{K_i, \ldots, K_j\}$ and only two of them force comparison between $K_i$ and $K_j$. 
A Fourth Method to Solve $T(n)$

**Expected number of comparisons**

- Let $T(n)$ be the expected number of comparisons.
- The expected of a sum is the sum of expectations, therefore

$$T(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} p_{ij}$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j - i + 1}$$

$$= \sum_{i=1}^{n-1} \sum_{h=2}^{n-i+1} \frac{2}{h}$$

$$< \sum_{i=1}^{n-1} \sum_{h=2}^{n} \frac{2}{h}$$

$$< \sum_{i=1}^{n-1} 2 \ln(n)$$

$$< 2n \ln(n)$$

(*replace $j - i + 1$ by $h$*)

(*use the bound for the harmonic numbers*)
Quick-Sort vs. Merge-Sort

**Complexity Remarks**

- Merge-Sort performs $\Theta(n \log n)$ comparisons in the worst case and in the average case.

- Quick-Sort performs $\Omega(n^2)$ comparisons in the worst case and $O(n \log n)$ comparisons in the average case.

- Class analysis shows that Merge-Sort performs less comparisons in the worst case than Quick-Sort performs in the average case.

- More accurate analysis shows that both sorting algorithms perform about the same number of comparisons.

- However, the overall time complexity of Quick-Sort is better than the overall time complexity of Merge-Sort.
Which Sorting Algorithm is the Fastest?

A summary

- What’s the fastest way to alphabetize your bookshelf?
  [https://www.youtube.com/watch?v=WaNLJf8xzC4](https://www.youtube.com/watch?v=WaNLJf8xzC4)

Quick Sort vs. Bubble-Sort and Merge-Sort

- Bubble-Sort vs. Quick-Sort:
  [https://www.youtube.com/watch?v=aXXWXz5rF64&vl=en](https://www.youtube.com/watch?v=aXXWXz5rF64&vl=en)

- Merge-Sort vs. Quick-Sort:
  [https://www.youtube.com/watch?v=es2T6KY45cA](https://www.youtube.com/watch?v=es2T6KY45cA)

Competition among many sorting algorithms

- Visualization and Comparison of Sorting Algorithms
  [https://www.youtube.com/watch?v=ZZuD6iUe3Pc](https://www.youtube.com/watch?v=ZZuD6iUe3Pc)

- Color Visualization of Sorting Algorithms
  [https://www.youtube.com/watch?v=14oa9QBT5Js](https://www.youtube.com/watch?v=14oa9QBT5Js)

- Visualization of 24 Sorting Algorithms In 2 Minutes
  [https://www.youtube.com/watch?v=BeoCbJPuvSE](https://www.youtube.com/watch?v=BeoCbJPuvSE)

- 15 Sorting Algorithms in 6 Minutes
  [https://www.youtube.com/watch?v=kPRA0W1kECg](https://www.youtube.com/watch?v=kPRA0W1kECg)
A Lower Bound for the Number of Comparisons

The adversary model

- **The algorithm goal:** Find a permutation of 1, \ldots, n.
  - There are $n! = n(n-1)(n-2) \cdots 2 \cdot 1$ permutations.

- **The adversary goal:**
  - Force any algorithm to have an $\Omega(n \log n)$ worst case complexity.
  - For any algorithm, select a permutation that is found by the algorithm with $\Omega(n \log n)$ comparisons.

Equivalency

- A lower bound for finding a permutation implies a lower bound for sorting arrays of keys.

- Why? Because in the comparison model after the array is sorted only the relative order among the keys are known.
The Adversary Strategy

Data structure
- Maintain a set $S_k$ of all the candidate permutations that are consistent with the first $k$ comparisons.
- Initially, $S_0$ is the set of all the $n!$ permutations.
- At the end, $S_h$ contains exactly one permutation.

Answering rules
- Let the $k^{th}$ comparison be $(A[i] : A[j])$:
$n = 4$

**Number of comparisons**

- Initially, there are $4! = 24$ candidate permutations.
- The adversary strategy forces any algorithm to perform at least $\lceil \log_2(24) \rceil = 5$ comparisons:
  - After 1 comparison, there are at least 12 candidates.
  - After 2 comparisons, there are at least 6 candidates.
  - After 3 comparisons, there are at least 3 candidates.
  - After 4 comparisons, there are at least 2 candidates.
  - After 5 comparisons, the permutation is found.
$n = 5$

Number of comparisons

- Initially, there are $5! = 120$ candidate permutations.
- The adversary strategy forces any algorithm to perform at least $\lceil \log_2(120) \rceil = 7$ comparisons:
  - After 1 comparison, there are at least 60 candidates.
  - After 2 comparisons, there are at least 30 candidates.
  - After 3 comparisons, there are at least 15 candidates.
  - After 4 comparisons, there are at least 8 candidates.
  - After 5 comparisons, there are at least 4 candidates.
  - After 6 comparisons, there are at least 2 candidates.
  - After 7 comparisons, the permutation is found.
$n = 6$

**Number of comparisons**

- Initially, there are $6! = 720$ candidate permutations.
- The adversary strategy forces any algorithm to perform at least $\lceil \log_2(720) \rceil = 10$ comparisons:
  - After 1 comparison, there are at least 360 candidates.
  - After 2 comparisons, there are at least 180 candidates.
  - After 3 comparisons, there are at least 90 candidates.
  - After 4 comparisons, there are at least 45 candidates.
  - After 5 comparisons, there are at least 23 candidates.
  - After 6 comparisons, there are at least 12 candidates.
  - After 7 comparisons, there are at least 6 candidates.
  - After 8 comparisons, there are at least 3 candidates.
  - After 9 comparisons, there are at least 2 candidates.
  - After 10 comparisons, the permutation is found.
$n = 4$

**Notations**

- Assume the numbers $\{1, 2, 3, 4\}$ are stored at the variables $\{x, y, z, w\}$.

- A permutation is represented by a 4-letter word composed of all the variables $x, y, z, w$:
  
  - $x = 2, y = 3, z = 1, w = 4$ implies the permutation $zxyw$.
  - The permutation $wyxz$ implies $x = 3, y = 2, z = 4, w = 1$.
  - If $y < z$ then $wyzx$ could be a candidate permutation and $zyxw$ cannot be a candidate permutation.
### Lower Bound

**The Adversary Technique**

\[ n = 4: S_0 \]

\[
\begin{array}{cccc}
xyzw & yxzw & zxyw & wxyz \\
xywz & yxwz & zxwy & wxzy \\
xzyw & yzwx & zyxw & wyxz \\
xzwy & yzwx & zywx & wyzx \\
xwyz & ywxz & zwxy & wzxy \\
xwzy & ywzx & zwyx & wzyx \\
\end{array}
\]

\[ \star |S_0| = 24 = 4! \]
$n = 4$: $S_1$ after $x < y$ is True

$|S_1| = 12$ for every comparison and every answer.
$n = 4$: the Comparison is $x < z$

<table>
<thead>
<tr>
<th>$S_2$ if $x &lt; z$ is true</th>
<th>$S_2$ if $z &lt; x$ is true</th>
</tr>
</thead>
<tbody>
<tr>
<td>$xyzw$</td>
<td>$zxyw$</td>
</tr>
<tr>
<td>$xywz$</td>
<td>$zxwy$</td>
</tr>
<tr>
<td>$xzyw$</td>
<td>$****$</td>
</tr>
<tr>
<td>$xzwy$</td>
<td>$****$</td>
</tr>
<tr>
<td>$xwyz$</td>
<td>$****$</td>
</tr>
<tr>
<td>$xwzy$</td>
<td>$****$</td>
</tr>
</tbody>
</table>

$|S_2| = 8$ since the adversary answers $x < z$ true.
$n = 4$: $S_2$ after $z < w$ is True

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$xyzw$</td>
<td>* ***</td>
<td>$zxyw$</td>
<td>* ***</td>
<td></td>
</tr>
<tr>
<td>* ***</td>
<td>* ***</td>
<td>$zxwy$</td>
<td>* ***</td>
<td></td>
</tr>
<tr>
<td>$xzyw$</td>
<td>* ***</td>
<td>* ***</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$xzwy$</td>
<td>* ***</td>
<td>* ***</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* ***</td>
<td>* ***</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>* ***</td>
<td>* ***</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$|S_2| = 6$ also if $w < z$ is true.
Lower Bound for the Number of Comparisons

**Theorem**
- Suppose an algorithm finds the permutation with $h$ comparisons.
- Then $h = \Omega(n \log n)$.

**Proof outline**
- $S_{k-1} = S \cup S' \Rightarrow |S_k| \geq \frac{|S_{k-1}|}{2}$.
- At least $\log_2(n!)$ comparisons are required to reduce the size of $S_0$ from $n!$ to 1 which is the size of $S_h$.
- The theorem follows since $\log_2(n!) = \Omega(n \log n)$. 
**Bounds on** \( \log_2(n!) \)

**A lower bound**

- \( n! \geq n(n - 1) \cdots \left\lceil \frac{n}{2} \right\rceil \geq \left( \left\lceil \frac{n}{2} \right\rceil \right)^\left\lceil \frac{n}{2} \right\rceil \geq \left( \frac{n}{2} \right)^{\frac{n}{2}}. \)
- \( \log_2(n!) \geq \log_2 \left( \left( \frac{n}{2} \right)^{\frac{n}{2}} \right) = \frac{n}{2} \log_2 \left( \frac{n}{2} \right) = \Omega(n \log n). \)

**An upper bound**

- \( n! \leq n^n. \)
- \( \log_2(n!) \leq \log_2(n^n) = n \log_2(n) = O(n \log n). \)

**Stirling’s approximation**

- \( n! = \sqrt{2\pi n} \left( \frac{n}{e} \right)^n (1 + \Theta \left( \frac{1}{n} \right)). \)
- \( \log_2(n!) = \Theta(n \log n). \)
Comparison Tree Algorithm to Sort 3 Keys

The Comparison Tree Technique

Amotz Bar-Noy (CUNY)
Rooted Binary Trees

Definitions

- **Binary trees**: Each internal node has 1 or 2 children.
- **Full binary trees**: Each internal node has exactly 2 children.
  - A full binary tree with \( k \) leaves has exactly \( k - 1 \) internal nodes.
- **Heights in trees**:
  - **Leaf height**: Length of path from the leaf to the root.
  - **Root height**: 0.
  - **Tree height**: The maximum height of one of the leaves.
- **Balanced binary trees**: Leaves heights are \( h \) or \( h + 1 \) \((h \geq 1)\).
Proposition

- Any comparison-based sorting algorithm can be represented by a full binary tree with \( n! \) leaves.

Tree construction

- The root represents the first comparison.
- Any internal node represents a comparison:
  - If the answer is YES continue with the left child.
  - If the answer is NO continue with the right child.
- A leaf represents a permutation.
Full Binary Trees: Height and Average Height

Notations

- \( T \): a full binary tree with \( k \) leaves.
- \( h(\ell) \): height of leaf \( \ell \).
- \( h(T) = \max_\ell \{ h(\ell) \} \): height of \( T \).
- \( \hat{h}(T) = (1/k) \sum_\ell h(\ell) \): average height of \( T \) over all \( k \) leaves.

Lemma I

- \( h(T) \geq \lceil \log_2 k \rceil \).

Lemma II

- \( \hat{h}(T) \geq \log_2 k \).
Height and Average Height

Example I

- **Balanced tree:** \( h(T) = 2 \) and \( \hat{h}(T) = 2 \).
- **Non-balanced tree:** \( h(T) = 3 \) and \( \hat{h}(T) = \frac{9}{4} = 2.25 \).
Height and Average Height

Example II

- **Balanced tree:** $h(T) = 3$ and $\hat{h}(T) = 16/6 = 2.666\ldots$
- **Non-balanced tree:** $h(T) = 4$ and $\hat{h}(T) = 17/6 = 2.833\ldots$
Proofs of the Lemmas

Lemma I

- \( h(T) \geq \lceil \log_2 k \rceil \).

Proof I sketch

- Any tree with the shortest height among all full trees with \( k \) leaves must be a balanced full binary tree.
- \( h(T) \geq \lceil \log_2 k \rceil \) in a balanced full binary tree \( T \) with \( k \) leaves.

Lemma II

- \( \hat{h}(T) \geq \lfloor \log_2 k \rfloor \).

Proof II sketch

- Any tree with the lowest average height among all full trees with \( k \) leaves must be one of the balanced full binary trees.
- \( \hat{h}(T) \geq \log_2 k \) in a balanced full binary tree \( T \) with \( k \) leaves.
In rounds, transform a non-balanced tree $T$ to a more balanced tree $T'$. Since $T$ is non-balanced, there must be two leaves $A$ and $B$ of height $x$ and one leaf $C$ of height $y$ for $y \leq x - 2$. In $T'$ make $C$ the parent of $A$ and $B$. $D$ the parent of $A$ and $B$ in $T$ becomes a leaf in $T'$. The leaf $C$ from $T$ is not a leaf in $T'$. All the other leaves of $T$ including $A$ and $B$ remain leaves in $T'$. 
Proofs of the Lemmas

Comparing the heights of $T$ and $T'$

- $h'(A) = h'(B) \leq h'(D) \leq x - 1 < x = h(A) = h(B)$.
- Therefore, $h'(T')$ the maximum height in $T'$ can only be smaller than $h(T)$ the maximum height in $T$.

Comparing the average heights of $T$ and $T'$

- $h(A) = h(B) = x$ and $h(C) = y$. Therefore these three leaves contribute $2x + y$ to the sum of all heights in $T$.
- $h'(A) = h'(B) = y + 1$ and $h'(D) = x - 1$. Therefore these three leaves contribute $2(y + 1) + (x - 1)$ to the sum of all heights in $T'$.
- $y + 1 < x$ implies that $2(y + 1) + (x - 1) = y + x + (y + 1) < y + 2x$.
- Since the heights of all other leaves remain the same, it follows that the sum of all heights in $T$ is larger than the sum of all heights in $T'$.
- Hence, the average height in $T'$ is smaller than the average height in $T$. 
A Deterministic Lower Bound

Key ideas

- Any deterministic sorting algorithm that sorts $n$ keys can be represented by a comparison tree with $n!$ leaves.

- The height of the comparison tree is the worst case number of comparisons performed by the algorithm.

- Lemma I implies that any deterministic sorting algorithm must perform at least $\lceil \log_2(n!) \rceil = \Omega(n \log n)$ comparisons in the worst case.
A Randomized Lower Bound

Theorem (Yao’ Principle)

- The expected cost of a randomized algorithm on the worst-case input is no better than the expected cost for a worst-case probability distribution on the inputs of the deterministic algorithm that performs best against that distribution.

Corollary to the Theorem

- The lowest average height of a comparison tree is a lower bound to the expected number of comparisons performed by any randomized algorithm.

Corollary to the Corollary

- Lemma II implies that any randomized sorting algorithm must perform at least \( \log_2(n!) = \Omega(n \log n) \) comparisons on average.
Sort in Linear Time

Idea
- Sort without comparisons by using memory locations.

Complexity
- Often $o(n \log n)$ and even $O(n)$ for sorting an array of $n$ keys.

A contradiction?
- A different model.
- A bounded range for the keys.
Bucket Sort

Input
- Keys belong to a **bounded** domain of size $k$:
  - Without loss of generality the keys are $1, 2, \ldots, k$.

Output

Idea
- For each key between 1 and $k$, **count** the number of times it appears in $A$ and then **rearrange** $A$.

Complexity
- $\Theta(n + k)$.
- $O(n)$ for $k = O(n)$.
- $o(n \log(n))$ for $k = o(n \log(n))$
Bucket Sort

Implementation

Bucket-Sort($A[1], \ldots, A[n]$)

for $i = 1$ to $k$ do
    $B[i] = 0$

for $j = 1$ to $n$ do
    $j = 0$

for $i = 1$ to $k$ do
    while $B[i] > 0$ do
        $j = j + 1$
        $A[j] = i$
        $B[i] = B[i] - 1$

(* prepare $k$ empty buckets *)

(* fill the buckets *)

(* spill all the buckets *)

(* spill bucket $i$ *)

Complexity

$\Theta(k) + \Theta(n) + \Theta(n + k) = \Theta(n + k)$. 
Stable Sorting Algorithms

Definition

- A sorting algorithm is **stable**:
  - If keys with the same values appear in the output array in the **same order** as they do in the input array.
  - If \( A[i] \) is placed in \( A[i'] \) and \( A[j] \) is placed in \( A[j'] \) for \( i < j \), then \( A[i] = A[j] \) implies that \( i' < j' \).

Motivation

- Important when **satellite** data are carried with the keys.

Observations

- There exists a **stable** implementation for most of the sorting algorithms.
- In particular, there exists a **stable** implementation of **Bucket-Sort** which is crucial for **Radix-Sort**.
Tuples as Keys

Definition

For positive integers $d$, $h$:

- A **key** is a tuple $⟨d_1, \ldots, d_d⟩$ of $d$ **digits**.
- Range of digits: $[0..(h − 1)]$.
- $d_1$ is the **most significant** digit.
- $d_d$ is the **least significant** digit.

Number of tuples

- Range of keys: $[0..(h^d − 1)]$.
- $h = 10$ In the base 10 number system. As a result, there are $10^d$ integers with at most $d$ digits from 0 to 99...9.
Lexicographic Order of Tuples

Definition

\[ \langle d_1, \ldots, d_d \rangle < \langle d'_1, \ldots, d'_d \rangle \text{ if} \]

* \( (d_1 < d'_1) \): (*1999... < 2111... *)

* \( (d_1 = d'_1 \text{ and } d_2 < d'_2) \): (*12999... < 13111... *)

* \( (\forall 1 \leq i < j < d \ d_i = d'_i \text{ and } d_j < d'_j) \): (*12...91999... < 12...92111... *)

* \( (\forall 1 \leq i < d \ d_i = d'_i \text{ and } d_d < d'_d) \): (*12...88 < 12...89 *)
Lexicographic Sort of Tuples

Algorithm

Lexicographic-Sort\((A[1], \ldots, A[n])\)

for \( i = 1 \) to \( d \) do

Sort \( A \) by digit \( i \)

Correctness

- By definition of lexicographic order.

Complexity

- \( \Theta(d(n + h)) \) using Bucket-Sort.

Implementation

- A complicated memory handling is required. Must avoid losing the already sorted prefixes of length \( i - 1 \) when sorting by digit \( i \).
Radix Sort of Tuples

Algorithm

Radix-Sort\((A[1], \ldots, A[n])\)

for \(i = d\) downto 1 do

Stable-Sort \(A\) by digit \(i\)

Correctness

- By induction: The suffixes of length \(s\) are sorted after \(s\) applications of the Stable-Sort.
- At the end, the suffixes of length \(d\) are the tuples themselves.

Complexity

\(\Theta(d(n + h))\) using Bucket-Sort same as Lexicographic-Sort.

Implementation

- Relatively easy due to the stability of the sorting procedure.
Example

4555  4432  3345  7942  6168  2173  1741  1629  9733  8258  2199
### Example

<table>
<thead>
<tr>
<th>4555</th>
<th>1741</th>
</tr>
</thead>
<tbody>
<tr>
<td>4432</td>
<td>4432</td>
</tr>
<tr>
<td>3345</td>
<td>7942</td>
</tr>
<tr>
<td>7942</td>
<td>2173</td>
</tr>
<tr>
<td>6168</td>
<td>9733</td>
</tr>
<tr>
<td>2173</td>
<td>4555</td>
</tr>
<tr>
<td>1741</td>
<td>3345</td>
</tr>
<tr>
<td>1629</td>
<td>6168</td>
</tr>
<tr>
<td>9733</td>
<td>8258</td>
</tr>
<tr>
<td>8258</td>
<td>1629</td>
</tr>
<tr>
<td>2199</td>
<td>2199</td>
</tr>
</tbody>
</table>
## Example

| 4555 | 1741 | 1629 |
| 4432 | 4432 | 4432 |
| 3345 | 7942 | 9733 |
| 7942 | 2173 | 1741 |
| 6168 | 9733 | 7942 |
| 2173 | 4555 | 3345 |
| 1741 | 3345 | 4555 |
| 1629 | 6168 | 8258 |
| 9733 | 8258 | 6168 |
| 8258 | 1629 | 2173 |
| 2199 | 2199 | 2199 |
Example

<table>
<thead>
<tr>
<th>4555</th>
<th>1741</th>
<th>1629</th>
<th>6168</th>
</tr>
</thead>
<tbody>
<tr>
<td>4432</td>
<td>4432</td>
<td>4432</td>
<td>2173</td>
</tr>
<tr>
<td>3345</td>
<td>7942</td>
<td>9733</td>
<td>2199</td>
</tr>
<tr>
<td>7942</td>
<td>2173</td>
<td>1741</td>
<td>8258</td>
</tr>
<tr>
<td>6168</td>
<td>9733</td>
<td>7942</td>
<td>3345</td>
</tr>
<tr>
<td>2173</td>
<td>4555</td>
<td>3345</td>
<td>4432</td>
</tr>
<tr>
<td>1741</td>
<td>3345</td>
<td>4555</td>
<td>4555</td>
</tr>
<tr>
<td>1629</td>
<td>6168</td>
<td>8258</td>
<td>1629</td>
</tr>
<tr>
<td>9733</td>
<td>8258</td>
<td>6168</td>
<td>9733</td>
</tr>
<tr>
<td>8258</td>
<td>1629</td>
<td>2173</td>
<td>1741</td>
</tr>
<tr>
<td>2199</td>
<td>2199</td>
<td>2199</td>
<td>7942</td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>4555</th>
<th>1741</th>
<th>1629</th>
<th>6168</th>
<th>1629</th>
</tr>
</thead>
<tbody>
<tr>
<td>4432</td>
<td>4432</td>
<td>4432</td>
<td>2173</td>
<td>1741</td>
</tr>
<tr>
<td>3345</td>
<td>7942</td>
<td>9733</td>
<td>2199</td>
<td>2173</td>
</tr>
<tr>
<td>7942</td>
<td>2173</td>
<td>1741</td>
<td>8258</td>
<td>2199</td>
</tr>
<tr>
<td>6168</td>
<td>9733</td>
<td>7942</td>
<td>3345</td>
<td>3345</td>
</tr>
<tr>
<td>2173</td>
<td>4555</td>
<td>3345</td>
<td>4432</td>
<td>4432</td>
</tr>
<tr>
<td>1741</td>
<td>3345</td>
<td>4555</td>
<td>4555</td>
<td>4555</td>
</tr>
<tr>
<td>1629</td>
<td>6168</td>
<td>8258</td>
<td>1629</td>
<td>6168</td>
</tr>
<tr>
<td>9733</td>
<td>8258</td>
<td>6168</td>
<td>9733</td>
<td>7942</td>
</tr>
<tr>
<td>8258</td>
<td>1629</td>
<td>2173</td>
<td>1741</td>
<td>8258</td>
</tr>
<tr>
<td>2199</td>
<td>2199</td>
<td>2199</td>
<td>7942</td>
<td>9733</td>
</tr>
</tbody>
</table>
Radix Sort

Correctness

- **Induction claim**: After sorting digit $i$, the suffixes of length $d - i + 1$ of all $n$ tuples are sorted.

- **Induction base for $i = d$**: By definition of sorting.

- **Induction hypothesis**: Claim is true for length $d - i$ suffixes.

- **Induction step**: Due to the stability of the digit sorting, when sorting by digit $i$, the existing order among the suffixes of length $d - i$ is preserved.
Radix Sort of Integers

**Setting**
- Keys: Tuples of $d$ digits each from the range $[0..9]$ ($h = 10$).
- Keys are integers from the range $[0..(10^d - 1)]$.

**Complexity**
- $\Theta(dn)$ complexity since $h = 10$ is constant.

**Radix-Sort vs. Bucket-Sort**
- Let $k = 10^d$ and assume $n = O(k)$.
- The complexity of **Bucket-Sort** is $\Theta(n + k) = \Theta(k)$.
- The complexity of **Radix-Sort** is $\Theta(dn) = \Theta(n \log(k))$.
- For $n \ll k$, **Radix-Sort** is much better than **Bucket-Sort**.
  - $k = 10^{12}$ (one billion) and $n = 1000$.
  - **Bucket-Sort** complexity is around one billion while **Radix-Sort** complexity is around 12000.
Solving Array Problems

Model
- The input is an array containing $n$ keys (usually numbers).
- Sometimes the input includes several arrays of the same or different sizes.

Goal
- **Efficiently** do something with the array(s) and/or find something that is based on some or all the numbers in the array(s).
- Determine if the complexity of the most efficient solution is $\Theta(1)$, $\Theta(\log(n))$, $\Theta(n)$, $\Theta(n \log(n))$, or $\Theta(n^2)$. 
Sorted Arrays Vs. Unsorted Arrays

**Sorted arrays**
- Can a binary-search like procedure solve the problem with complexity $\Theta(\log n)$ instead of a possible “trivial” solution that scans the array and examines all the $n$ numbers?
- Can the problem be solved by inspecting only $\Theta(1)$ numbers avoiding more involved search procedures?

**Unsorted arrays**
- Is it possible to solve the problem with complexity $\Theta(n)$ avoiding sorting the array?
- Will sorting the array yield a solution with complexity $\Theta(n \log n)$ instead of a possible “trivial” $\Theta(n^2)$ solution that examines all pairs of numbers?