Analysis of Algorithms

SORT

The Sorting Problem

Keys: Entities from a well ordered domain.

Comparison: For 2 keys K_1 and K_2

- either $K_1 < K_2$
- or $K_2 < K_1$
- or $K_1 = K_2$.

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

Output: A sorted array $A[1] \leq A[2] \leq \cdots \leq A[n]$.

Goal: Minimize number of comparisons between keys.

Complexity of the Sorting Problem

- ⋆ $\Omega(n \log n)$ comparisons lower bound.
- ⋆ $O(n^2)$ comparisons simple upper bounds.
- ⋆ $O(n \log n)$ comparisons upper bound.
- ⋆ $\Theta(n \log n)$ overall complexity.

- Bounds are for both worst case and average case complexity.

Simple Sorting Algorithms and Their Time Complexity

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

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

Selection-Sort: $\Theta(n^2)$ worst & average case.

Efficient Sorting Algorithms and Their Time Complexity

Merge-Sort: $\Theta(n \log n)$ worst & average case.

Quick-Sort: $\Theta(n \log n)$ average case.
$\Theta(n^2)$ worst case.

Heap-Sort: $\Theta(n \log n)$ worst & average case.

Binary-Tree-Sort: $\Theta(n \log n)$ average case.
$\Theta(n^2)$ worst case.

Balanced-Tree-Sort: $\Theta(n \log n)$ worst & average case.

Bubble Sort

Ideas:

- ⋆ Find the minimum $n-1$ times.
- ⋆ Compare and exchange only adjacent keys.

Implementation:

Bubble-Sort$(A[1], \ldots, A[n])$
  for $i = 1$ to $n-1$
    for $j = n$ downto $i+1$
      if $A[j] < A[j-1]$ (* comparison *)
        then $A[j] \leftrightarrow A[j-1]$

Bubble Sort – Correctness

⋆ By induction, for $1 \leq i \leq n-1$, after round i:

- $A[1] \leq A[2] \leq \cdots \leq A[i]$.
- $A[i] \leq A[j]$ for all $i < j \leq n$.

⋆ After $n-1$ rounds: $A[1] \leq A[2] \leq \cdots \leq A[n]$.

Bubble Sort – Complexity

- ⋆ For $1 \le i \le n-1$, in round i: exactly $n-i$ comparisons.

- ⋆ The total number of comparisons is always

$$(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}=\Theta(n^2)\ .$$

Insertion Sort

Ideas:

- ⋆ Insert the current key into a sorted prefix of the array.
- ⋆ Repeat this insertion $n-1$ times.
- ⋆ Compare and exchange only adjacent keys.

Implementation:

Insertion-Sort$(A[1],\ldots,A[n])$
  for $i = 2$ to n
    $j = i$
    while $(j > 1)$ and $(A[j] > A[j-1])$ (* comparison *)
      $A[j] \leftrightarrow A[j-1]$
      $j = j - 1$

Insertion Sort – Correctness

- By induction, for $2 \leq i \leq n$, after round i:
  - $A[1] \leq A[2] \leq \cdots \leq A[i]$.
- After $n - 1$ rounds:
  - $A[1] \leq A[2] \leq \cdots \leq A[n]$.

Insertion Sort – Worst Case Complexity

Upper bound on the number of comparisons:

- $\star$ $1 + 2 + \cdots + (n-1) = \frac{n(n-1)}{2}$.
- $\star$ The input array is sorted in a decreasing order.

Lower bound to the number of comparisons:

- $\star$ $1 + 1 + \cdots + 1 = n - 1$.
- $\star$ The input array is sorted in a non-decreasing order.

Complexity: $\Theta(n^2)$ comparisons in the worst case.

Insertion Sort with Binary Search

- ⋆ With binary search Insertion sort can be implemented with $\Theta(n \log n)$ comparisons.

- ⋆ But with $\Theta(n^2)$ operations:
  - – When the input array is sorted in a decreasing order, there are $\Theta(n^2)$ array assignments operations.

- ⋆ In any case, $\Theta(n^2)$ worst case time complexity.

Insertion Sort – Average Case Complexity

Assumption: A uniform distribution over all n permutations.

Round i for $2 \leq i \leq n$:

- ⋆ For $1 \leq k \leq i$, $A[i]$ is the k-largest among $A[1], \ldots, A[i]$ with probability $1/i$.
- ⋆ k comparisons if $A[i]$ is the k-largest, for $1 \leq k \leq i - 1$.
- ⋆ $i - 1$ comparisons if $A[i]$ is the smallest (i-largest).
- ⋆ Average number of comparisons in round i is

$$\frac{1}{i}\left(1 + 2 + \cdots + (i-1) + (i-1)\right) = \frac{(i+1)}{2} - \frac{1}{i}.$$

Insertion Sort – Average Case Complexity

Average number of comparisons $C(n)$:

$$
\begin{aligned}
C(n) &= \sum_{i=2}^{n} \frac{(i+1)}{2} - \sum_{i=2}^{n} \frac{1}{i} \\
&= \frac{1}{2} \sum_{i=3}^{n+1} i - \sum_{i=2}^{n} \frac{1}{i} \\
&= \frac{1}{2} \left(\frac{(n+1)(n+2)}{2} - 3 \right) - \sum_{i=2}^{n} \frac{1}{i} \\
&\approx \frac{n^2}{4} + \frac{3n}{4} - 1 - \ln n \\
&\approx \frac{n^2}{4} = \Theta(n^2)
\end{aligned}
$$

Merge-Sort

Divide and Conquer for $n \geq 2$:

- ⋆ Recursively sort the sub-arrays $A[1..q]$ and $A[q+1..n]$ for $q = \left\lfloor \frac{n+1}{2} \right\rfloor$.

- ⋆ Merge the sub-arrays $A[1..q]$ and $A[q+1..n]$ into a sorted array $A[1..n]$.

The Merge Procedure

Global array: $A[1], A[2], \ldots, A[n]$.

Merge(p, q, r) :

- ⋆ $1 \leq p \leq q < r \leq n$.
- ⋆ Merge the two sorted sub-arrays
  $A[p] \leq \cdots \leq A[q]$ and $A[q+1] \leq \cdots \leq A[r]$
  into a sorted sub-array $A[p] \leq \cdots \leq A[r]$.

Complexity: Number of comparisons is at most $(r - p)$.

The Recursive Merge-Sort Procedure

Initial recursive call: Merge-Sort$(1, n)$.

Recursive procedure:

```
Merge-Sort(p, r)
  if r > p then
     q = ⌊(p+r)/2⌋      (* p ≤ q < q+1 ≤ r *)
     Merge-Sort(p, q)
     Merge-Sort(q+1, r)
     Merge(p, q, r)
```

Merge-Sort – Correctness

⋆ By induction on $r - p$.

⋆ Case $r = p$ the array is sorted trivially.

⋆ Case $p \leq q < r$ the induction hypothesis holds:

– For sub-array $A[p..q]$ since $q - p < r - p$.

– For sub-array $A[q+1..r]$ since $r - (q+1) < r - p$.

⋆ The induction step is correct due to the correctness of procedure Merge.

MergeSort – Complexity

⋆ $T(n)$ - number of comparisons.

⋆ $T(1) = 0$.

⋆ $T(n) \leq T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + (n-1)$.

Master Theorem:

– $a = 2$, $b = 2$, and $d = 1$.

– $\log_b(a) = d \quad \Rightarrow \quad T(n) = \Theta(n \log n)$.

Merge-Sort – Complexity for $n = 2^k$

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

Guess: $T(n) \leq n \log_2 n - (n-1)$.

Guessing by Unfolding the Recursion

$$
\begin{aligned}
T(2^k) &\leq 2T(2^{k-1}) + (2^k - 1) \\
&= 2T(2^{k-1}) + (1 \cdot 2^k - 1) \\
&\leq 2(2T(2^{k-2}) + (2^{k-1} - 1)) + (2^k - 1) \\
&= 4T(2^{k-2}) + (2 \cdot 2^k - 3) \\
&\leq 4(2T(2^{k-3}) + (2^{k-2} - 1)) + (2 \cdot 2^k - 3) \\
&= 8T(2^{k-3}) + (3 \cdot 2^k - 7) \\
&\vdots \\
&= 2^i T(2^{k-i}) + (i \cdot 2^k - (2^i - 1)) \\
&\vdots \\
&= 2^k T(2^0) + (k \cdot 2^k - (2^k - 1)) \\
&= n \log_2 n - (n - 1)
\end{aligned}
$$

Proof By Induction for $n = 2^k$

Claim: $T(n) \leq n \log_2 n - (n-1)$.

Verify for $n = 1$: $1 \log_2 1 - (1-1) = 0 \geq T(1)$.

Verify for $n = 2$: $2 \log_2 2 - (2-1) = 1 \geq T(2)$.

Verify for $n = 4$: $4 \log_2 4 - (4-1) = 5 \geq T(4)$.

Proof By Induction for $n = 2^k$

Assume correctness for $n/2$:

$$\begin{aligned} T(n/2) &\leq (n/2)\log_2(n/2) - (n/2 - 1) \\ &= (n/2)(\log_2 n - 1) - (n/2 - 1) \\ &= (n/2)\log_2 n - (n - 1) \end{aligned}$$

Induction step for n:

$$\begin{aligned} T(n) &\leq 2T(n/2) + (n - 1) \\ &\leq 2\left((n/2)\log_2 n - (n - 1)\right) + (n - 1) \\ &= n\log_2 n - (n - 1)\ . \end{aligned}$$

Merge-Sort – Complexity for $n \neq 2^k$

$$
\begin{aligned}
T(1) &= 0 \\
T(2) &\leq T(1) + T(1) + (2-1) = 1 \\
T(3) &\leq T(2) + T(1) + (3-1) = 3 \\
T(4) &\leq T(2) + T(2) + (4-1) = 5 \\
T(5) &\leq T(3) + T(2) + (5-1) = 8 \\
T(6) &\leq T(3) + T(3) + (6-1) = 11 \\
T(7) &\leq T(4) + T(3) + (7-1) = 14 \\
T(8) &\leq T(4) + T(4) + (8-1) = 17 \\
T(9) &\leq T(5) + T(4) + (9-1) = 21
\end{aligned}
$$

Merge-Sort – Complexity for $n \neq 2^k$

Guess: $T(n) \leq n \lceil \log_2 n \rceil - \left(2^{\lceil \log_2 n \rceil} - 1\right)$.

Verify for $n = 1$: $1 \lceil \log_2 1 \rceil - (2^{\lceil \log_2 1 \rceil} - 1) = 0 \geq T(1)$.

Verify for $n = 3$: $3 \lceil \log_2 3 \rceil - (2^{\lceil \log_2 3 \rceil} - 1) = 3 \geq T(3)$.

Verify for $n = 6$: $6 \lceil \log_2 6 \rceil - (2^{\lceil \log_2 6 \rceil} - 1) = 11 \geq T(6)$.

Verify for $n = 8$: $8 \lceil \log_2 8 \rceil - (2^{\lceil \log_2 8 \rceil} - 1) = 17 \geq T(8)$.

Verify for $n = 9$: $9 \lceil \log_2 9 \rceil - (2^{\lceil \log_2 9 \rceil} - 1) = 21 \geq T(9)$.

Observations

- $\star$ $\lceil \log_2(k+1) \rceil = \lceil \log_2 k \rceil$ for $k \neq 2^h$.
- $\star$ $\lceil \log_2(k+1 \rceil = \lceil \log_2 k \rceil + 1$ for $k = 2^h$.
- $\star$ $\lceil \log_2(2k) \rceil = \lceil \log_2 k \rceil + 1$.
- $\star$ $\lceil \log_2(2k+1) \rceil = \lceil \log_2 k \rceil + 1$ for $k \neq 2^h$.
- $\star$ $\lceil \log_2(2k+1) \rceil = \lceil \log_2 k \rceil + 2$ for $k = 2^h$.

Case $n = 2k$

Claim: $T(n) \le n \lceil \log_2 n \rceil - (2^{\lceil \log_2 n \rceil} - 1)$.

Induction step:

$$
\begin{aligned}
T(n) &\le 2T(k) + (n-1) \\
&\le 2(k \lceil \log_2 k \rceil - (2^{\lceil \log_2 k \rceil} - 1)) + (n-1) \\
&= n(\lceil \log_2 k \rceil + 1) - (2^{\lceil \log_2 k \rceil + 1} - 1) \\
&= n \lceil \log_2 n \rceil - (2^{\lceil \log_2 n \rceil} - 1) \ .
\end{aligned}
$$

Case $n = 2k + 1$ and $k \neq 2^h$

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

Induction step:

$$\begin{aligned}
T(n) &\leq T(k+1) + T(k) + (n-1) \\
&\leq ((k+1) \lceil \log_2 (k+1) \rceil - (2^{\lceil \log_2 (k+1) \rceil} - 1)) \\
&\quad + (k \lceil \log_2 k \rceil - (2^{\lceil \log_2 k \rceil} - 1)) + (n-1) \\
&= n(\lceil \log_2 k \rceil + 1) - (2^{\lceil \log_2 k \rceil + 1} - 1) \\
&= n \lceil \log_2 n \rceil - (2^{\lceil \log_2 n \rceil} - 1) \ .
\end{aligned}$$

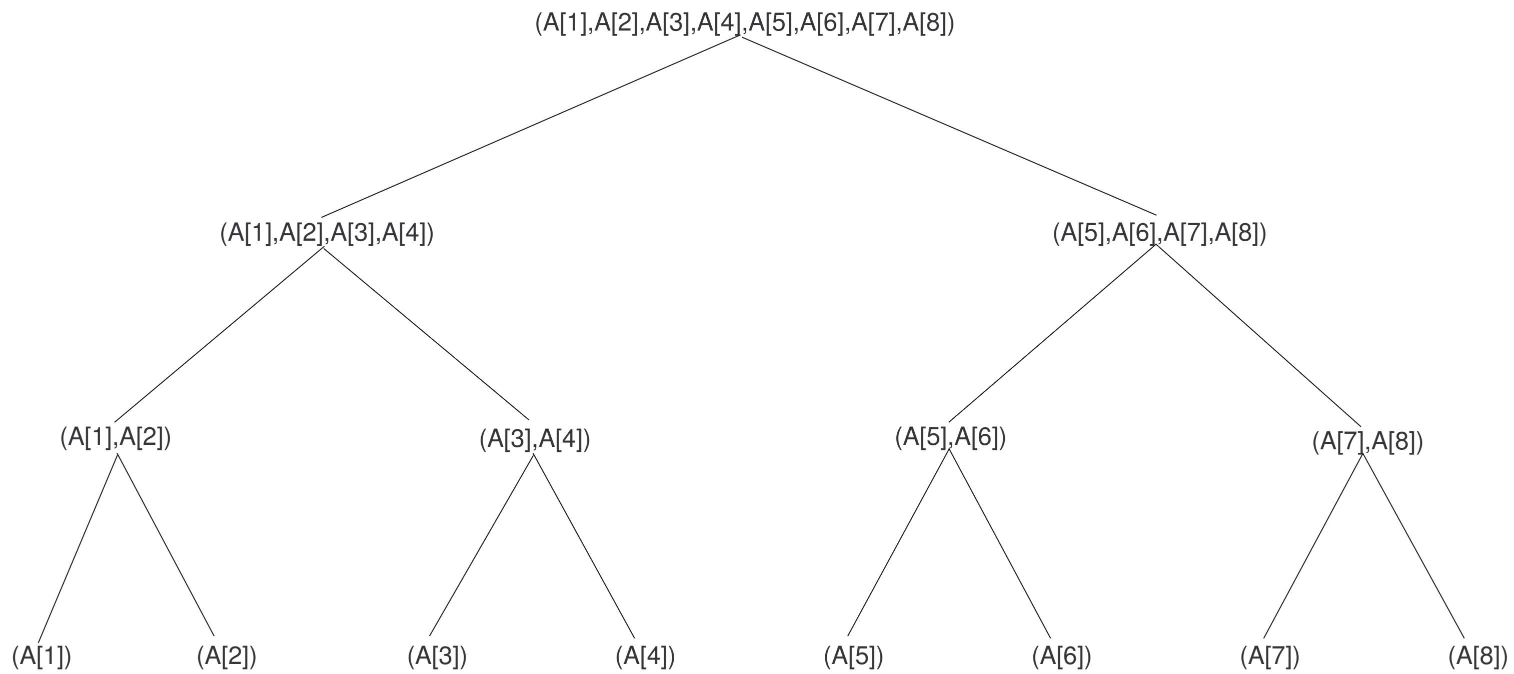
Case $n = 2k + 1$ and $k = 2^h$

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

Induction step:

$$
\begin{aligned}
T(n) &\leq T(k+1) + T(k) + (n-1) \\
&\leq ((k+1) \lceil \log_2 (k+1) \rceil - (2^{\lceil \log_2 (k+1) \rceil} - 1)) \\
&\quad + (k \lceil \log_2 k \rceil - (2^{\lceil \log_2 k \rceil} - 1)) + (n-1) \\
&= (k+1)(h+1) - (2k-1) + kh - (k-1) + 2k \\
&= (2k+1)h + 3 = n(h+2) - (2n-3) \\
&= n \lceil \log_2 n \rceil - (2^{\lceil \log_2 n \rceil} - 1)\ .
\end{aligned}
$$

Merge-Sort – Structure



Non-Recursive Merge-Sort Procedure – $n = 2^k$

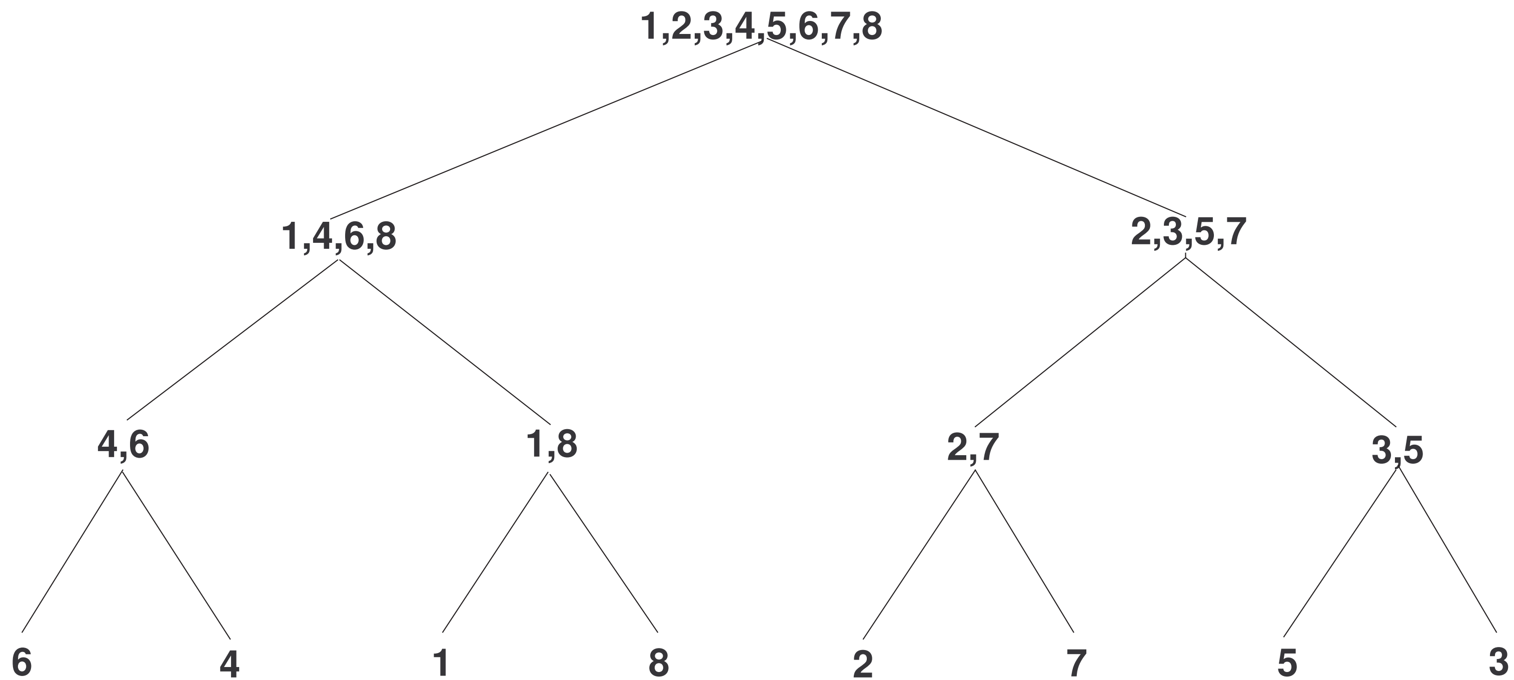
Assumption: $n = 2^k$ for integer $k \geq 1$.

Merge-Sort$(1, n)$
  for $h = 1$ to k
    for $p = 1$ to n step 2^h
      $r = p + 2^h - 1$
      $q = p + 2^{h-1} - 1$ (* $q = \lfloor (p + r)/2 \rfloor$ *)
      Merge(p, q, r)

Example: variables for $n = 16 = 2^4$

h	p	q	r
1	$1, 3, \ldots, 15$	$1, 3, \ldots, 15$	$2, 4, \ldots, 16$
2	$1, 5, 9, 13$	$2, 6, 10, 14$	$4, 8, 12, 16$
3	$1, 9$	$4, 12$	$8, 16$
4	1	8	16

Example: execution for $n = 8$



Correctness – $n = 2^k$

- ⋆ By induction, for $1 \le h \le k = \log_2 n$, after round h:

$$
\begin{aligned}
&A[1] \le A[2] \le \cdots \le A[2^h] \\
&A[2^h+1] \le A[2^h+2] \le \cdots \le A[2 \cdot 2^h] \\
&A[2 \cdot 2^h+1] \le A[2 \cdot 2^h+2] \le \cdots \le A[3 \cdot 2^h] \\
&\qquad \vdots \qquad\qquad \vdots \\
&A[2^k - 2^h + 1] \le A[2^k - 2^h + 2] \le \cdots \le A[2^k]
\end{aligned}
$$

- ⋆ After $k = \log_2 n$ rounds: $A[1] \le A[2] \le \cdots \le A[n]$.

Correctness – Example for $n = 2^5 = 32$ and $h = 3$

⋆ After 3 rounds:

$$A[1] \leq A[2] \leq \cdots \leq A[8]$$
$$A[9] \leq A[10] \leq \cdots \leq A[16]$$
$$A[17] \leq A[18] \leq \cdots \leq A[24]$$
$$A[25] \leq A[26] \leq \cdots \leq A[32]$$

Non-Recursive Merge-Sort Procedure – $n \neq 2^k$

Assumption: $2^{k-1} < n < 2^k$ for integer $k \geq 1$.

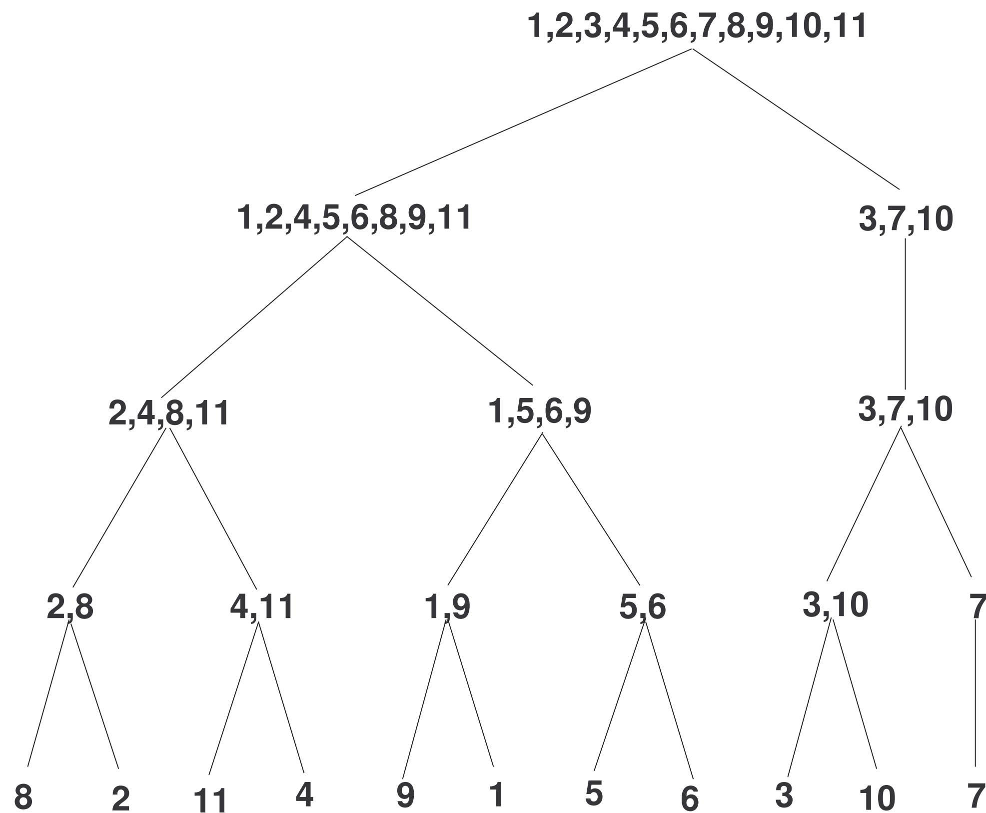
Merge-Sort$(1, n)$
  for $h = 1$ to k
    for $p = 1$ to n step 2^h
      $r = \min\left\{n, (p + 2^h - 1)\right\}$
      $q = \min\left\{n, (p + 2^{h-1} - 1)\right\}$
      if $r > q$ then Merge(p, q, r)

Remark: In some rounds, the suffix is not merged.

Example: variables for $n = 27$

h	p	q	r
1	$1, 3, \ldots, 25, 27$	$1, 3, \ldots, 25, 27$	$2, 4, \ldots, 26, 27$
2	$1, 5, \ldots, 21, 25$	$2, 6, \ldots, 22, 26$	$4, 8, \ldots, 24, 27$
3	$1, 9, 17, 25$	$4, 12, 20, 27$	$8, 16, 24, 27$
4	$1, 17$	$8, 24$	$16, 27$
5	1	16	27

Example: execution for $n = 11$



Correctness – $2^{k-1} < n < 2^k$

- By induction, for $1 \le h \le k = \lceil \log_2 n \rceil$, after round h:

$$
\begin{array}{l}
A[1] \le A[2] \le \cdots \le A[2^h] \\
A[2^h + 1] \le A[2^h + 2] \le \cdots \le A[2 \cdot 2^h] \\
A[2 \cdot 2^h + 1] \le A[2 \cdot 2^h + 2] \le \cdots \le A[3 \cdot 2^h] \\
\qquad\qquad \vdots \qquad\qquad\qquad \vdots \\
A[\lfloor n/2^h \rfloor \cdot 2^h + 1] \le A[\lfloor n/2^h \rfloor \cdot 2^h + 2] \le \cdots \le A[n]
\end{array}
$$

- After $k = \lceil \log_2 n \rceil$ rounds: $A[1] \le A[2] \le \cdots \le A[n]$.

Correctness – Example for $n = 27$ and $h = 2$

⋆ After 2 rounds:

$$A[1] \le A[2] \le A[3] \le A[4]$$
$$A[5] \le A[6] \le A[7] \le A[8]$$
$$A[9] \le A[10] \le A[11] \le A[12]$$
$$A[13] \le A[14] \le A[15] \le A[16]$$
$$A[17] \le A[18] \le A[19] \le A[20]$$
$$A[21] \le A[22] \le A[23] \le A[24]$$
$$A[25] \le A[26] \le A[27]$$

Complexity – $n = 2^k$

Idea: Procedure Merge is executed $\frac{n}{2^h}$ times on size 2^h for $1 \leq h \leq k = \log_2 n$.

$$
\begin{aligned}
T(2^k) &\leq \sum_{h=1}^{k} \left(\frac{n}{2^h}(2^h - 1) \right) \\
&= \sum_{h=1}^{k} n - n \sum_{h=1}^{k} \frac{1}{2^h} \\
&= n \log_2 n - n \left(1 - \frac{1}{2^k} \right) \\
&= n \log_2 n - (n - 1) \ .
\end{aligned}
$$

Complexity – $2^{k-1} < n < 2^k$

- $\star$ At most $n-1$ comparisons in each of the $\lceil \log_2 n \rceil$ rounds.
- $\star$ **Upper bound:**
  $T(n) \leq (n-1)\lceil \log_2 n \rceil = n \lceil \log_2 n \rceil - \lceil \log_2 n \rceil$.
- $\star$ A careful analysis can improve the above bound. However, the complexity is not $T(n) = n \lceil \log_2 n \rceil - (2^{\lceil \log_2 n \rceil} - 1)$ as with the recursive implementation.
  - $T_{recursive}(5) = 8$ and $T_{non-recursive}(5) = 9$.

Quick-Sort

Divide and Conquer for $n \geq 2$:

- $\star$ Partition the array $A[1..n]$ into two sub-arrays $A[1..q]$ and $A[q+1..n]$ such that all the keys in the sub-array $A[1..q]$ are smaller or equal to all the keys in the sub-array $A[q+1..n]$.
- $\star$ Recursively Sort the sub-arrays $A[1..q]$ and $A[q+1..n]$.

The Partition Procedure

Global array: $A[1], A[2], \ldots, A[n]$.

Partition(p, r)**:**

- $\star$ $r > p$.
- $\star$ Return a value $p \leq q < r$ such that $A[i] \leq A[j]$ for any $p \leq i \leq q$ and $q + 1 \leq j \leq r$.

Method: Pivot partitioning.

- $\star$ One key is compared with the rest of the keys.
- $\star$ This key is the $(q - p)$-smallest in the sub-array $A[p..r]$.

Complexity: Number of comparisons is exactly $(r - p)$.

The Recursive Quick-Sort Procedure

Initial recursive call: Quick-Sort$(1, n)$.

Recursive procedure:

```
Quick-Sort(p, r)
  if r > p then
     q =Partition(p, r)
     Quick-Sort(p, q)
     Quick-Sort(q + 1, r)
```

Quick-Sort – Correctness

Assumption: Make sure that $p \leq q < r$.

Proof:

- ⋆ By induction on $r - p$.
- ⋆ Case $r = p$ the array is sorted trivially.
- ⋆ Case $p \leq q < r$ the induction hypothesis is true:
  - – For sub-array $A[p..q]$ since $q - p < r - p$.
  - – For sub-array $A[q + 1..r]$ since $r - (q + 1) < r - p$.
- ⋆ The induction step is correct since procedure Partition guarantees that all the keys in $A[p..q]$ are smaller or equal to all the keys in $A[q + 1..r]$

Quick-Sort – Complexity

⋆ $T(n)$ - number of comparisons.

⋆ $T(1) = 0$.

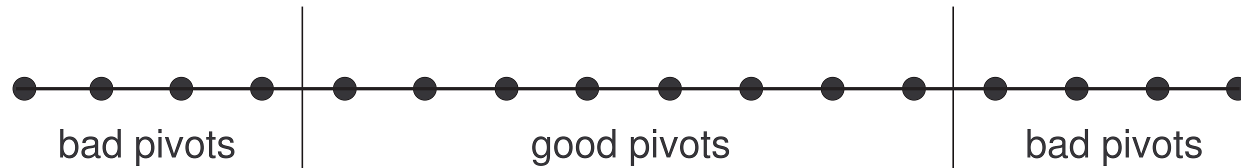
⋆ $T(n) \leq T(q) + T(n-q) + (n-1)$.

Best: $T(n) \leq 2T(n/2) + (n-1) = \Theta(n \log n)$.

Good: $T(n) \leq T(n/10) + T(9n/10) + (n-1) = \Theta(n \log n)$.

Worst: $T(n) \geq T(n-1) + (n-1) = \Theta(n^2)$.

Quick-Sort – Expected Number of Comparisons



A good pivot: Greater than at least $n/4$ keys and smaller than at least $n/4$ keys.

Probabilities facts:

- ⋆ With probability $1/2$ a random pivot is good.
- ⋆ Expected number of random pivots until finding a good pivot is 2.

$\Theta(n \log n)$ Expected Number of Comparisons

- $\star$ $\Theta(n)$ to perform one partition.
- $\star$ $\Theta(n)$ until a good partition is performed.
- $\star$ For a recursive inequality of the type

$$T(n) \leq T(\gamma n) + T((1-\gamma)n) + \Theta(n)$$

  the worst case is when $\gamma \to 1$.

- $\star$ $\gamma \leq 3/4$ for a good pivot.
- $\star$ Therefore for Quick-Sort:

$$T(n) \leq T(3n/4) + T(n/4) + \Theta(n) = \Theta(n \log n).$$

  – The expectation of a sum is the sum of expectations.

Solving the Recursive Formula

Assumption: ignore floors and ceilings.

Formula: $T(n) \leq T(3n/4) + T(n/4) + \alpha n$ for constant $\alpha > 0$.

Claim: $T(n) \leq \beta n \log n$ for constant $\beta > 1.25\alpha$.

Solving the Recursive Formula

Induction step:

$$\begin{aligned} T(n) &\leq \beta \frac{3n}{4} \log_2 \left(\frac{3n}{4} \right) + \beta \frac{n}{4} \log_2 \left(\frac{n}{4} \right) + \alpha n \\ &= \beta \left(\frac{3n}{4} \log_2 n + \frac{n}{4} \log_2 n \right) - \beta \frac{3n}{4} \log_2 \left(\frac{4}{3} \right) - \beta \frac{n}{4} 2 + \alpha n \\ &= \beta n \log_2 n + \left(\alpha - \frac{\beta}{2} - \frac{3\beta}{4} \log_2 \left(\frac{4}{3} \right) \right) n \\ &\leq \beta n \log_2 n \ . \end{aligned}$$

Solving the Recursive Formula

- ⋆ The coefficient of n must be negative if $T(n) \le \beta n \log_2 n$.
- ⋆ $\alpha < \frac{\beta}{2} + \frac{3\beta}{4} \log_2 \left(\frac{4}{3}\right)$.
- ⋆ $\beta > \frac{1}{0.5+0.75 \log_2(1.333)} \alpha \approx 1.233\alpha$.

Quick-Sort – Average Case Complexity

Assumption: For $n \geq 2$ and $1 \leq q < n$, with probability $1/(n-1)$ the value of q is $1, 2, \ldots, n-1$.

Fix q: $(T(q) + T(n-q))$ comparisons in the recursive calls.

Procedure Partition: Exactly $n-1$ comparisons.

Recursive Formula for Average Case Complexity

$$
\begin{aligned}
T(n) &= (n-1) + \frac{1}{n-1}\sum_{q=1}^{n-1}(T(q) + T(n-q)) \\
&= (n-1) + \frac{2}{n-1}\sum_{q=1}^{n-1} T(q) \\
&= \Theta(n \log n)\ .
\end{aligned}
$$

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

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

$$\begin{aligned} \sum_{q=1}^{n-1} q \ln(q) &\leq \int_1^n x \ln(x) dx \\ &= \frac{1}{2} n^2 \ln(n) - \frac{1}{4} n^2 + \frac{1}{4} . \end{aligned}$$

Solving the Recursive Formula for Quick-Sort

⋆ Induction hypothesis: $T(q) \le cq\ln(q)$ for $1 \le q < n$.

$$\begin{aligned}
T(n) &= (n-1) + \frac{2}{n-1}\sum_{q=1}^{n-1} T(q) \\
&\le (n-1) + \frac{2c}{n-1}\sum_{q=1}^{n-1} q\ln(q) \\
&\le (n-1) + \frac{2c}{n-1}\left(\frac{1}{2}n^2\ln(n) - \frac{1}{4}n^2 + \frac{1}{4}\right)
\end{aligned}$$

⋆ $T(n) \le cn\ln(n)$ for some constant c.

Another Method

- $T(n) = (n-1) + \frac{2}{n-1}\sum_{q=1}^{n-1} T(q)$.
- $T(n-1) = (n-2) + \frac{2}{n-2}\sum_{q=1}^{n-2} T(q)$.
- $(n-1)T(n) - (n-2)T(n-1) = (2n-3) + 2T(n-1)$.
- $(n-1)T(n) - nT(n-1) = 2n-3$.
- $\frac{T(n)}{n} - \frac{T(n-1)}{n-1} = \frac{2n-3}{n(n-1)}$.

Another Method – Continue

- ⋆ $S(n) = \frac{T(n)}{n}$ and $S(1) = 0$.
- ⋆ $S(n) = S(n-1) + \frac{2n-3}{n(n-1)}$
- ⋆ $S(n) = \sum_{i=2}^{n} \frac{2i-3}{i(i-1)} < \sum_{i=2}^{n} \frac{2}{i} = 2H(n) - 2 \leq 2\ln(n)$.
- ⋆ $T(n) = nS(n) \leq 2n\ln(n) \approx 1.386n\log_2(n)$.

Quick-Sort vs. Merge-Sort

- ⋆ Merge-Sort performs $O(n \log n)$ 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.
- ⋆ Merge-Sort performs less comparisons in the worst case than Quick-Sort performs in the average case.
- ⋆ However, the overall complexity of Quick-Sort in the average case is better than the overall complexity of Merge-Sort in the average case.

Deterministic Lower Bound for the Comparison 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 the 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.

Adversary Strategy

- ⋆ 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 $n!$ permutations.
- ⋆ At the end S_h contains exactly one permutation.
- ⋆ Let the k-th comparison be $(A[i] : A[j])$:
  - S the consistent permutations if $A[i] < A[j]$.
  - S' the consistent permutations if $A[i] > A[j]$.
  - $S_k = S'$ if $|S| \leq |S'|$.
  - $S_k = S$ if $|S| > |S'|$.

Example $n = 4$

- ⋆ 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.

Example $n = 4$

- ⋆ Assume the numbers $1, 2, 3, 4$ are stored at x, y, z, w.
- ⋆ A permutation is represented by the letters x, y, z, w:
  - $x = 2, y = 3, z = 1, w = 4$ implies permutation $zxyw$.
  - 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.

$n = 4$: S_0

$xyzw$	$yxzw$	$zxyw$	$wxyz$
$xywz$	$yxwz$	$zxwy$	$wxzy$
$xzyw$	$yzxw$	$zyxw$	$wyxz$
$xzwy$	$yzwx$	$zywx$	$wyzx$
$xwyz$	$ywxz$	$zwxy$	$wzxy$
$xwzy$	$ywzx$	$zwyx$	$wzyx$

⋆ $|S_0| = 24 = 4!$

$n = 4$: S_1 after $x < y$ is True

$xyzw$	$****$	$zxyw$	$wxyz$
$xywz$	$****$	$zxwy$	$wxzy$
$xzyw$	$****$	$****$	$****$
$xzwy$	$****$	$****$	$****$
$xwyz$	$****$	$zwxy$	$wzxy$
$xwzy$	$****$	$****$	$****$

⋆ $|S_1| = 12$ for every comparison and every answer.

$n = 4$: the Comparison is $x < z$

S_2 if $x < z$ is true		S_2 if $z < x$ is true	
$xyzw$	$wxyz$	$zxyw$	$****$
$xywz$	$wxzy$	$zxwy$	$****$
$xzyw$	$****$	$****$	$****$
$xzwy$	$****$	$****$	$****$
$xwyz$	$****$	$zwxy$	$wzxy$
$xwzy$	$****$	$****$	$****$

⋆ $|S_2| = 8$ since the adversary answers $x < z$ true.

$n = 4$: S_2 after $z < w$ is True

$xyzw$	$****$	$zxyw$	$****$
$****$	$****$	$zxwy$	$****$
$xzyw$	$****$	$****$	$****$
$xzwy$	$****$	$****$	$****$
$****$	$****$	$zwxy$	$****$
$****$	$****$	$****$	$****$

⋆ $|S_2| = 6$ also if $w < z$ is true.

Lower Bound for the Number of Comparisons

- The algorithm finds the permutation with h comparisons.
- $S_{k-1} = S \cup S' \quad \Rightarrow \quad |S_k| \geq \frac{|S_{k-1}|}{2}$.
- $h \geq \log_2(|S_0|) = \log_2(n!) = \Omega(n \log n)$.

$$\log_2(n!) = \Omega(n \log n)$$

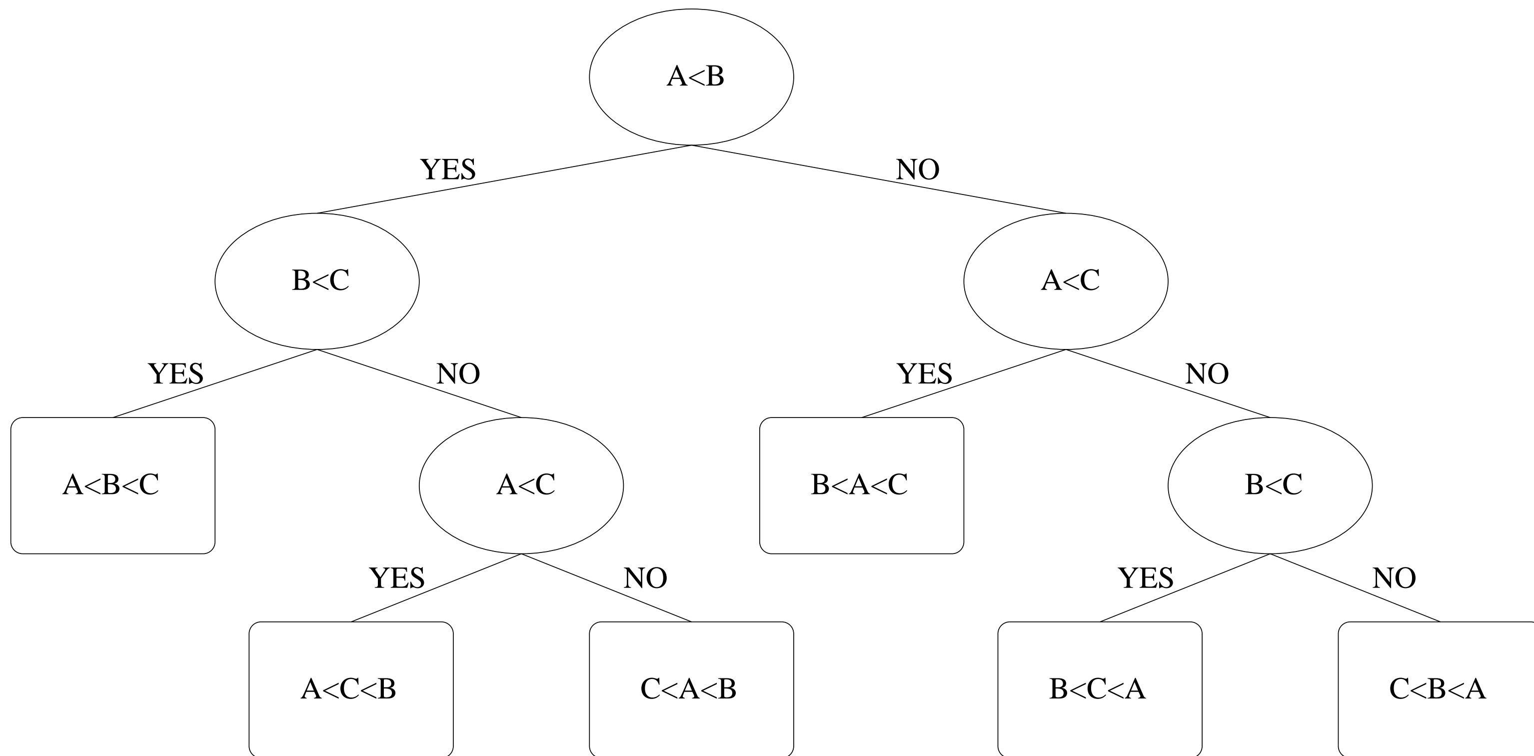
Direct approach:

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

Stirling's approximation:

- $\star$ $n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1 + \Theta\left(\frac{1}{n}\right)\right)$.
- $\star$ $h \ge \log_2(n!) = \Theta(n \log n)$.

Comparison Tree Algorithm to Sort 3 Keys



Comparison Tree for Sorting Algorithms

- ⋆ A full binary tree with $n!$ leaves.
- ⋆ 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.

Binary Trees

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

Full Binary Trees: Height and Average Height

Notations:

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

Lemma I: $h(T) \geq \lceil \log_2 k \rceil$.

Lemma II: $\hat{h}(T) = \Omega(\log_2 k)$.

Example: Height and Average Height

Balanced tree: $h(T) = 2$ and $\hat{h}(T) = 2$.

Non-balanced tree: $h(T) = 3$ and $\hat{h}(T) = 9/4$.

Proofs

Proof I:

- ⋆ The shortest full tree with k leaves is the balance full binary tree.
- ⋆ $h(T) \geq \lceil \log_2 k \rceil$ in a balance full binary tree T.

Proof II:

- ⋆ The tree with the smallest average height among all full trees with k leaves is the balance full binary tree.
- ⋆ $\hat{h}(T) \geq \lfloor \log_2 k \rfloor$ in a balance full binary tree T.

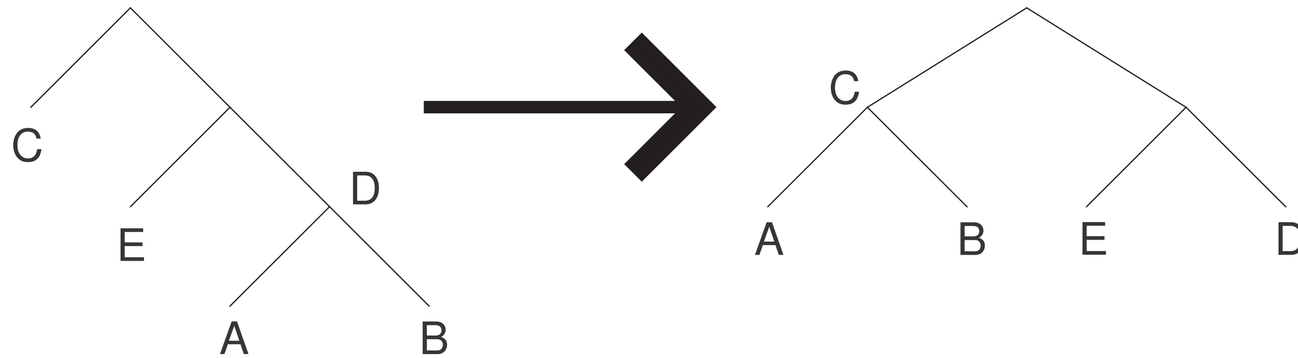
Why Balanced Trees are the Shortest?

- ⋆ Transform a non-balanced tree to a balanced tree by reducing the height of tall leaves and Increasing the height of short leaves while preserving the number of leaves.

- ⋆ Assume there are 2 leaves A and B of height x and one leaf C of height $x-2$.

- ⋆ Replace these 3 leaves with the parent D of A and B of height $x-1$ and move A and B to be 2 new children of C of height $x-1$.

Why Balanced Trees are the Shortest?



⋆ The proofs follow since

- $x > x - 1$ for the maximum height.
- $(x-2)+(x-1)+2x > 4(x-1)$ for the average height.

Deterministic Lower Bound for the Comparison Model

- ⋆ 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 $\lceil \log_2(n!) \rceil = \Omega(n \log n)$ comparisons.

Randomized Lower Bound for the Comparison Model

- ⋆ Any randomized sorting algorithm that sorts n keys can be represented by a comparison tree with $n!$ leaves.
- ⋆ The average height of the comparison tree is the average number of comparisons performed by the algorithm.
- ⋆ **Lemma II** implies that any randomized sorting algorithm must perform $\Omega(n \log n)$ comparisons.

Sort in Linear Time

Idea: Sort without comparisons using memory locations.

Complexity: Sometimes $o(n \log n)$ operations for sorting an array of n keys.

A contradiction? No!

- ⋆ A different model.
- ⋆ A bounded range for the keys.

Sort in Linear Time

Bucket & Counting Sort

⋆ Integers from the range $[1..k]$.

⋆ $\Theta(n+k)$ operations.

⋆ $k = O(n)$
$\Rightarrow \Theta(n)$ operations.

⋆ $k = o(n \log n)$
$\Rightarrow$ Better than $\Omega(n \log n)$.

Radix Sort

⋆ Integers from the range $[1..k^d]$.

⋆ $\Theta(d(n+k))$ operations.

⋆ $k = O(n)$ and constant d
$\Rightarrow \Theta(n)$ operations.

⋆ $k = O(n)$ and $d = o(\log n)$
$\Rightarrow$ Better than $\Omega(n \log n)$.

Bucket Sort

Input:

- ⋆ An unsorted array $A[1], A[2], \ldots, A[n]$ of n keys.
- ⋆ Keys belong to a bounded domain of size k.

Output: A sorted array $A[1] \leq A[2] \leq \cdots \leq A[n]$.

Idea: For each value between 1 and k, count the number of times it appears in A and then rearrange A.

Complexity: $\Theta(n + k)$ operations.

Bucket Sort – Implementation

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

for $i = 1$ to k do $B[i] = 0$ (* prepare k empty buckets *)

for $j = 1$ to n do $B[A[j]] = B[A[j]] + 1$

(* fill the buckets *)

$j = 0$

for $i = 1$ to k do (* spill the buckets *)

while $B[i] > 0$ do (* spill the buckets *)

$j = j + 1; \quad A[j] = i; \quad B[i] = B[i] - 1$

Complexity: $\Theta(k) + \Theta(n) + \Theta(n) = \Theta(n + k)$.

Stable Sorting Algorithms

- $\star$ 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 $i' < j'$.
- $\star$ Important when satellite data are carried with the keys.
- $\star$ Counting Sort is stable: Crucial for Radix Sort.
- $\star$ Most of the sorting algorithms can be implemented stable.

Counting Sort

Input:

- ⋆ An unsorted array $A[1], A[2], \ldots, A[n]$ of n keys.
- ⋆ Keys belong to a bounded domain of size k.

Output: A sorted array $A[1] \leq A[2] \leq \cdots \leq A[n]$.

Idea: A stable way to rearrange A.

Complexity: $\Theta(n + k)$ operations.

Counting Sort – Distinct Keys

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

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

for $j = 1$ to n do $C[A[j]] = 1$

(* if $C[i] = 1$ then the key i is in A *)

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

(* $C[i]$ – number of keys "$\leq i$" in A *)

(* $C[i]$ – new location of i in A *)

for $j = n$ downto 1 do $B[C[A[j]]] = A[j]$

for $j = 1$ to n do $A[j] = B[j]$

Complexity: $\Theta(k) + \Theta(n) + \Theta(k) + \Theta(n) + \Theta(n) = \Theta(n + k)$.

Counting Sort: Arbitrary Keys

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

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

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

(* $C[i]$ – number of keys "$= i$" in A *)

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

(* $C[i]$ – number of keys "$\leq i$" in A *)

(* $C[i]$ – new location of last i in A *)

for $j = n$ downto 1 do

$B[C[A[j]]] = A[j]$

$C[A[j]] = C[A[j]] - 1$

(* $C[i]$ – new location of current i in A *)

for $j = 1$ to n do $A[j] = B[j]$

Complexity: $\Theta(n + k)$ – the same as for distinct keys.

Tuples as Keys

- For positive integers d, k:
  - A key is a tuple $\langle d_1, \ldots, d_d \rangle$ of d digits.
  - Digits from the range $[1..k]$.
  - Keys from the range $[1..k^d]$.
  - d_1 is the most significant digit.
  - d_d is the least significant digit.

Lexicographic Order of Tuples

$\star$ $\langle d_1, \ldots, d_d \rangle < \langle d'_1, \ldots, d'_d \rangle$

if $(d_1 < d'_1)$ $1999... < 2012...$

or $(d_1 = d'_1$ and $d_2 < d'_2)$ $2399... < 2412...$

– $\vdots$

or $(\forall_{1 \leq i < j < d} d_i = d'_i$ and $d_j < d'_j)$ $12...9598... < 12...9612...$

– $\vdots$

or $(\forall_{1 \leq i < d} d_i = d'_i$ and $d_d < d'_d)$ $12...8 < 12...9$

Lexicographic Sort of Tuples

Lexicographic-Sort$(A[1], \ldots, A[n])$
  for $i = 1$ to d do
    sort A on digit i.

Correctness: By definition of lexicographic order.

Implementation: A complicate memory handling.

Complexity: $\Theta(d(n+k))$ using Counting-Sort.

Radix Sort of Tuples

Radix-Sort$(A[1], \ldots, A[n])$
  for $i = d$ downto 1 do
    apply a stable sort to sort A on digit i.

Correctness: By induction on the digit being sorted relying on the stability of the digit sort.

Implementation: Easy due to the stability of the digit sorting.

Complexity: $\Theta(d(n + k))$ using Counting-Sort.

Example

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

Radix Sort – Correctness

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

Verifying the claim 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 sort, the induction claim is true for suffixes of length $d - i + 1$.

Radix Sort of Integers

- ⋆ Keys: Tuples of d digits each from the range $[1..k]$.
- ⋆ Set $k = O(n)$.
- ⋆ Keys are integers from the range $[1..(O(n))^d]$.
- ⋆ $\Theta(d(n+k))$ complexity becomes $\Theta(dn)$ complexity.
- ⋆ Constant d implies a linear time algorithm.
- ⋆ Counting-Sort is linear only for a range $[1..O(n)]$.