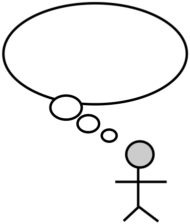
2-2 The Efficiency of Algorithms

魏恒峰

hfwei@nju.edu.cn

2018 年 04 月 02 日



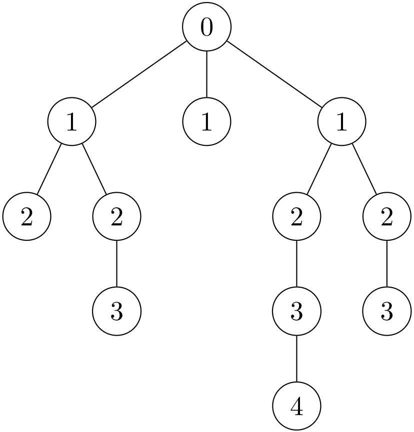
(1) Diameter of Convex Polygon: $\Theta(n)$

(2) Lower Bound for Sorting: $\Omega(n \lg n)$

(3) Traversal over Trees: DFS/BFS ($\Theta(n)$)

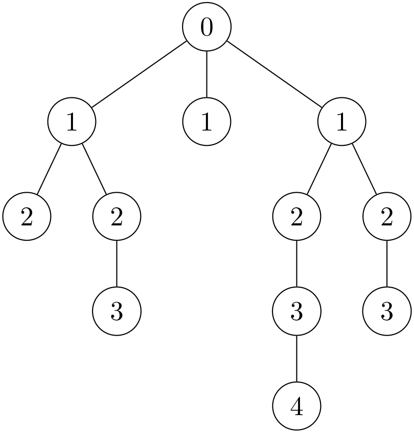
I have thought that $\cdots$

DH 4.2 (a): Sum of Depths



$$\text{sum-of-depths}(r) = \begin{cases} \text{depth of } r, & r \text{ is a leaf} \\ \displaystyle\sum_{v:\text{child of } r} \text{sum-of-depths}(v) + \text{depth of } r, & \text{o.w.} \end{cases}$$

DH 4.2 (a): Sum of Depths



$$\text{sum-of-depths}(r, {\color{red}d}) = \begin{cases} d, & r \text{ is a leaf} \\ \displaystyle\sum_{v:\text{child of } r} \text{sum-of-depths}(v, {\color{red}d+1}) + d, & \text{o.w.} \end{cases}$$

Algorithm 1 Calculate the sum of depths of all nodes of a tree T.

procedure $\textsc{Sum-of-Depths}()$
    return $\textsc{Sum-of-Depths}(T, 0)$

procedure $\textsc{Sum-of-Depths}(r, depth)$ ▷ r: root of a tree
    if r is a leaf **then**
        return $depth$

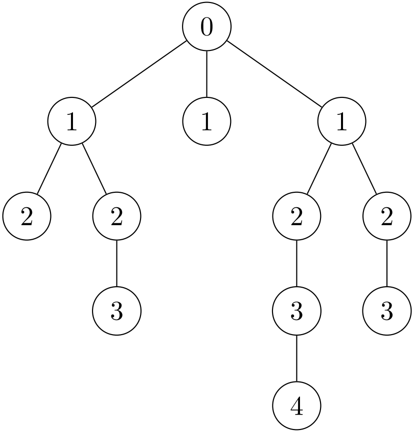
    $sum \leftarrow 0$
    for all child vertex v of r **do**
        $sum \leftarrow sum + \textsc{Sum-of-Depths}(v, depth + 1)$
    return sum

$$\text{sum-of-depths}(r, d) = \begin{cases} d, & r \text{ is a leaf} \\ \displaystyle\sum_{v:\text{child of } r} \text{sum-of-depths}(v, d+1) + d, & \text{o.w.} \end{cases}$$

Master Theorem?

$$\Theta(m + n) = \Theta(n)$$

DH 4.2 (b): Number of Nodes at Depth K



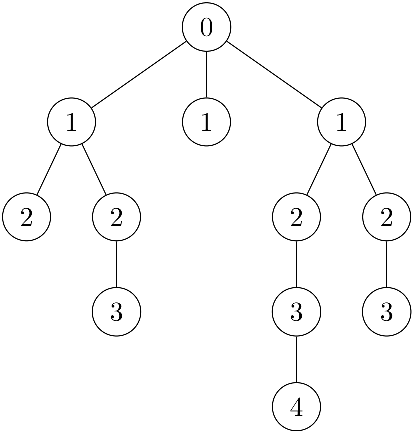
$$\text{nodes-at-depth}(r, {\color{red}k}) = \begin{cases} 1, & k = 0 \\ 0, & k > 0 \land r \text{ is a leaf} \\ \displaystyle\sum_{v:\text{child of } r} \text{nodes-at-depth}(v, {\color{red}k - 1}), & \text{o.w.} \end{cases}$$

Algorithm 2 Count the number of nodes in T at depth K.

procedure $\textsc{Nodes-at-Depth}()$
  return $\textsc{Nodes-at-Depth}(T, K)$

procedure $\textsc{Nodes-at-Depth}(r, k)$ ▷ r: root of a tree
  if $k = 0$ **then**
    return 1
  if r is a leaf **then**
    return 0
  $num \leftarrow 0$
  for all child vertex v of r **do**
    $num \leftarrow num + \textsc{Nodes-at-Depth}(v, k-1)$
  return num

DH 4.2 (c): Any Leaf at an Even Depth?



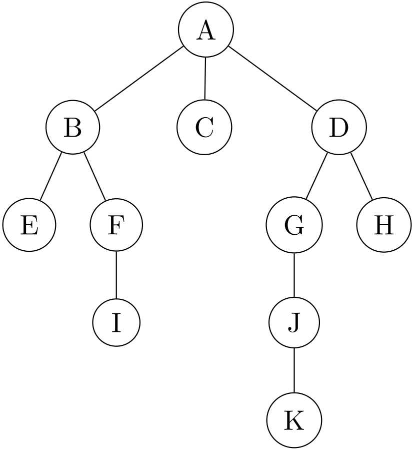
$$\text{leaf-at-depth}(r, {\color{red}parity}) = \begin{cases} 1 - parity, & r \text{ is a leaf} \\ \displaystyle\bigvee_{v\text{:child of } r} (v, {\color{red}1 - parity}), & \text{o.w.} \end{cases}$$

Algorithm 3 Check whether a tree T has any leaf at an even depth.

procedure $\textsc{Leaf-at-Even-Depth}()$
    return $\textsc{Leaf-at-Depth}(T, even = 0)$

procedure $\textsc{Leaf-at-Depth}(r, parity)$ ▷ r: root of a tree
    if r is a leaf **then**
        return $1 - parity$
    $result \leftarrow 0$
    for all child vertex v of r **do**
        $result \leftarrow result \lor \textsc{Leaf-at-Depth}(v, 1 - parity)$
    return $result$

DH 4.3 (a): Sum of Contents at Each Depth



Algorithm 4 Calculate the sum of contents of nodes of a tree T at each depth.

procedure $\textsc{Sum-at-Depth}(r)$ $\triangleright$ r: root of the tree T

$\quad r.depth \leftarrow 0$

$\quad Q \leftarrow \emptyset$

$\quad \textsc{Enqueue}(Q, r)$

$\quad$ **while** $Q \neq \emptyset$ **do**

$\quad\quad u \leftarrow \textsc{Dequeue}(Q)$

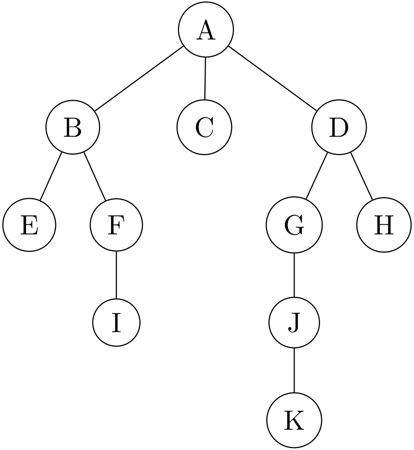
$\quad\quad sumAtDepth[u.depth] \mathrel{+}= u.content$

$\quad\quad$ **for all** child vertex v of u **do**

$\quad\quad\quad v.depth \leftarrow u.depth + 1$

$\quad\quad\quad \textsc{Enqueue}(Q, v)$

DH 4.3 (b): Depth K with the Maximum Number of Nodes



Algorithm 5 Count the number of nodes of a tree T at each depth.

procedure $\textsc{Nodes-at-Depth}(r)$ ▷ r: root of the tree T

$r.depth \leftarrow 0$

$Q \leftarrow \emptyset$

$\textsc{Enqueue}(Q, r)$

while $Q \neq \emptyset$ **do**

  $u \leftarrow \textsc{Dequeue}(Q)$

  $nodesAtDepth[u.depth] \mathrel{+}= 1$

  for all child vertex v of u **do**

    $v.depth \leftarrow u.depth + 1$

    $\textsc{Enqueue}(Q, v)$

return $\text{argmax}_K nodesAtDepth[k]$

$v.depth$

$sumAtDepth[]$

Q : Space $\Theta(n) \to \Theta(1)$

Lower Bound for Comparion-based Sorting

Lower Bound for Comparison-based Sorting (DH 6.13)

Prove a lower bound of $O(n \lg n)$ on the time complexity of any comparison-based sorting algorithm.

Lower Bound for Comparison-based Sorting (DH 6.13)

Prove a lower bound of $\Omega(n \lg n)$ on the time complexity of any comparison-based sorting algorithm on inputs of size n.

Lower Bound for Comparison-based Sorting (DH 6.13)

Prove a lower bound of $\Omega(n \lg n)$ on the time complexity of any **comparison-based** sorting algorithm on inputs of size n.

Cost Model:

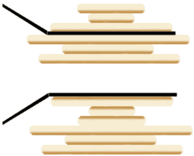
the critical operations to count

Computational Model:

the only way to gain order info.

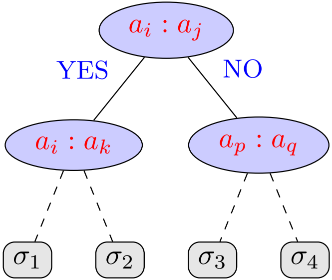
$x \in [1 \cdots 99]$

$x/10$



"Bounds For Sorting By Prefix Reversal", 1979

Decision Tree Model



Nodes: comparisions $a_i : a_j$

$$<,\ \leq,\ =,\ \geq,\ >$$

Edges: two-way decisions (Y/N)

Leaves: possible permutations

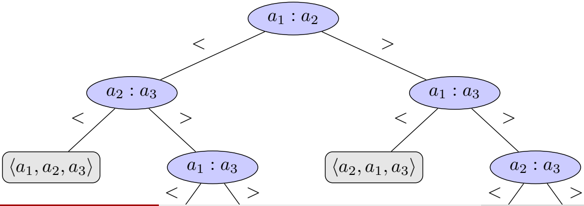
Assumption (By aware of any assumptions !!!):

All the input elements are **distinct**.

$$a_i < a_j$$

Decision Tree Model

Any Comparison-based Sorting Algorithm $\xRightarrow{\text{modeled by}}$ A Decision Tree



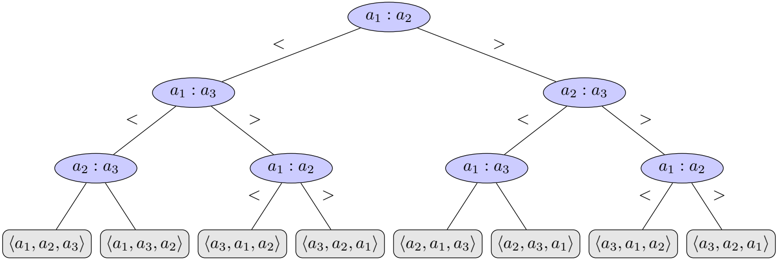
Decision Tree Model

Any Comparison-based Sorting Algorithm $\xRightarrow{\text{modeled by}}$ A Decision Tree

```
procedure SELECTION-SORT(A, n)
    for i ← 1 to n − 1 do
        for j ← i + 1 to n do
            if A[j] < A[i] then
                SWAP(A[j], A[i])
```

Decision Tree Model

Any Comparison-based Sorting Algorithm $\xrightarrow{\text{modeled by}}$ A Decision Tree



The decision tree for selection sort on three elements.

Decision Tree Model

$$\boxed{\text{Any Comparison-based Sorting Algorithm } \mathcal{A} \xRightarrow{\text{modeled by}} \text{A Decision Tree } \mathcal{T}}$$

$$\text{Algorithm } \mathcal{A} \text{ on a specific input of size } n \xRightarrow{\text{modeled by}} \text{A path through } \mathcal{T}$$

$$\text{Worst-case time complexity of } \mathcal{A} \xRightarrow{\text{modeled by}} \text{The height of } \mathcal{T}$$

$$\boxed{\begin{array}{c}\text{Worst-case Lower Bound of Comparison-based Sorting (on inputs of size } n\text{)} \\ \xRightarrow{\text{modeled by}} \\ \text{The Minimum Height of All } \mathcal{T}\text{s}\end{array}}$$

Decision Tree Model

Worst-case Lower Bound of Comparison-based Sorting (on inputs of size n)

$$\xRightarrow{\text{modeled by}}$$

The Minimum Height of All $\mathcal{T}$s

To be a full binary tree:

$$\# \text{ of leaves } \leq 2^h$$

To be a correct sorting algorithm:

$$\# \text{ of leaves } \geq n!$$

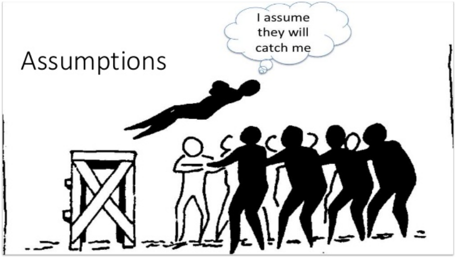
Lower Bound for Comparison-based Sorting

$$\boxed{n! \le \# \text{ of leaves} \le 2^h}$$

$$h \ge \lg n! = \Omega(n \lg n)$$

Stirling Formula (by *James Stirling*):

$$n! = \Theta\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right)$$



Assumptions (By aware of any assumptions !!!):

(a) Comparison-based sorting algorithms.

(b) All the input elements are **distinct**.

(c) Completely sort all possible inputs.

The k-sorted Problem

An array $A[1 \cdots n]$ is k-sorted if it can be divided into k blocks, each of size n/k (we assume that $n/k \in \mathbb{N}$), such that the elements in each block are larger than the elements in earlier blocks and smaller than elements in later blocks. The elements within each block need *not* be sorted.

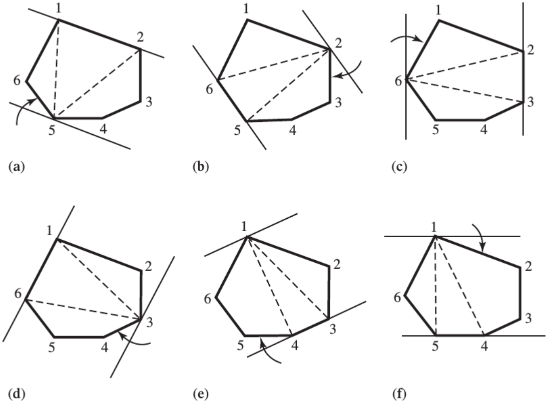
(a) Describe an algorithm that k-sorts an arbitrary array in $O(n \log k)$ time.

(b) Prove that any comparison-based k-sorting algorithm requires $\Omega(n \log k)$ comparisons in the worst case.

(c) Describe an algorithm that completely sorts an already k-sorted array in $O(n \log(n/k))$ time.

(d) Prove that any comparison-based algorithm to completely sort a k-sorted array requires $\Omega(n \log(n/k))$ comparisons in the worst case.

Convex Polygon Diameter



Convex Polygon Diameter (DH 6.8)

Show that the "Convex Polygon Diameter" algorithm is of **linear-time** complexity.

Q : Linear-time of **WHAT**?

A : Linear-time of **the size of input**

Q : What is the input?

A : A convex polygon represented by n vertices

Q : What are the critical operations?

A : $d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

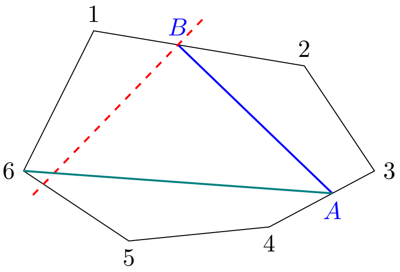
$$\Theta(c \cdot n) = \Theta(n)$$

Correctness

WAIT BUT WHY

Theorem

For a convex polygon, a pair of vertices determine the diameter.



BUT, we have *not* enumerated *all* pairs of vertices.

We have enumerated *all* pairs of vertices
that *admits parallel supporting lines*.

Definition (Line of Support)

A line L is a *line of support* of a convex polygon P if

$$L \cap P = \text{ a vertex/an edge of } P.$$

$$L \cap P \neq \emptyset \text{ and } P \text{ lies entirely on one side of } L.$$

Definition (Antipodal)

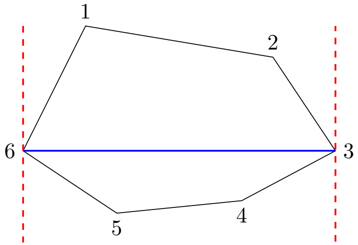
An *antipodal* is a pair of points that admits parallel supporting lines.

We have enumerated *all* antipodals.

Theorem

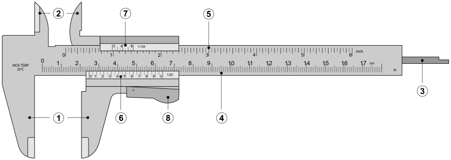
If AB is a diameter of a convex polygon P, then AB is an antipodal.

Proof.



□

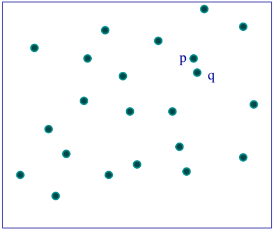
Rotating Caliper



"Computational Geometry"
Ph.D Thesis, Michael Shamos, 1978

"Solving Geometric Problems with the Rotating Calipers", 1983

Finding the Closest Pair of Points



A Classic and Beautiful Divide-Conquer-Combine Algorithm:

Section 33.4, CLRS


TREASURE
HUNT
on foot


Hints
&
Tips


Help
Wanted

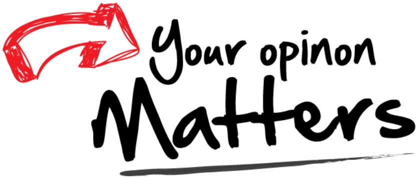
(a) Given an array $A[0 \cdots n-1]$, to determine whether there is a value that *occurs more than* $\lfloor n/k \rfloor$ *times* in $\Theta(n \lg k)$ time and $\Theta(k)$ extra space.

(b) Prove that the *lower bound* of this problem is $\Theta(n \lg k)$.

Take $k = 2$.

$\Theta(n)$ time $\&$ $\Theta(1)$ space


Thank
You!



Office 302

Mailbox: H016

hfwei@nju.edu.cn