2-2 The Efficiency of Algorithms

魏恒峰

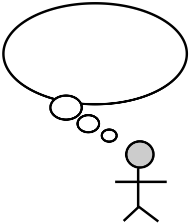
hfwei@nju.edu.cn

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 ($\Theta(n)$)



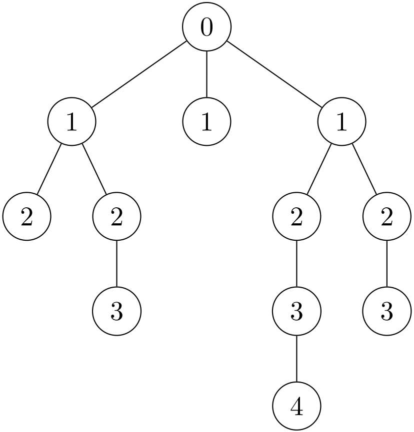
(1) Diameter of Convex Polygon: $\Theta(n)$

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

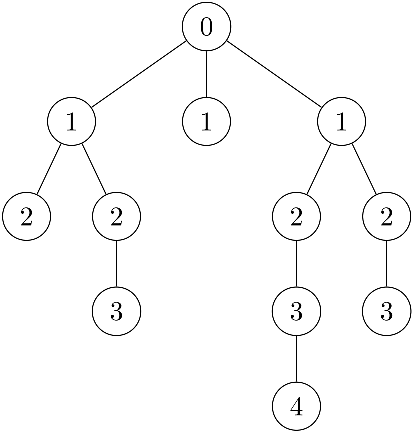
(3) Traversal over Trees: DFS/BFS ($\Theta(n)$)

I have thought that $\cdots$

DH 4.2 (a): Sum of Depths

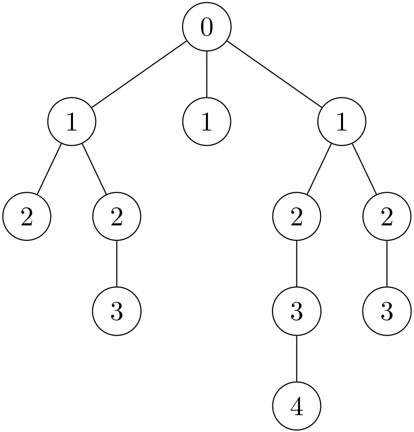


DH 4.2 (a): Sum of Depths



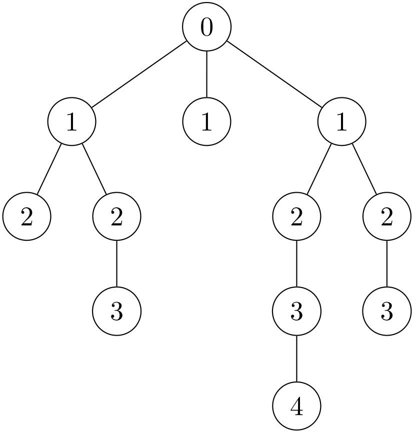
$$\text{sum-of-depths}(r) = \begin{cases} & \\ \displaystyle\sum_{v:\text{child of } r} \text{sum-of-depths}(v) + \text{depth of } r, & \end{cases}$$

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



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

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 $\textsc{Sum-of-Depths}(r, \mathit{depth})$ ▷ r: root of a tree
    if T is a leaf **then**
        return depth
    for all child vertex v of r **do**
        $\mathit{depth} \leftarrow \mathit{depth} + \textsc{Sum-of-Depths}(v, \mathit{depth} + 1)$
    return depth

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

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

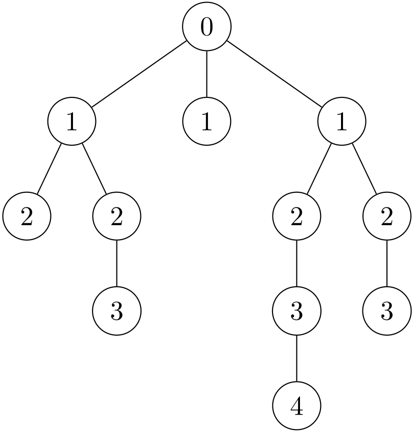
Master Theorem?

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

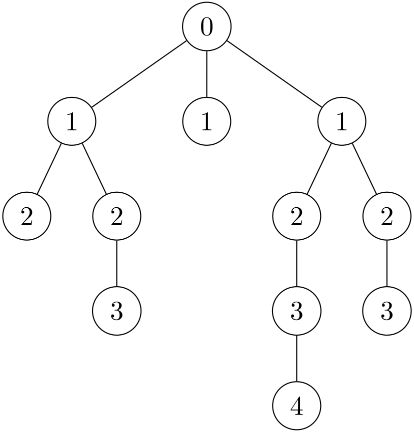
Master Theorem?

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

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

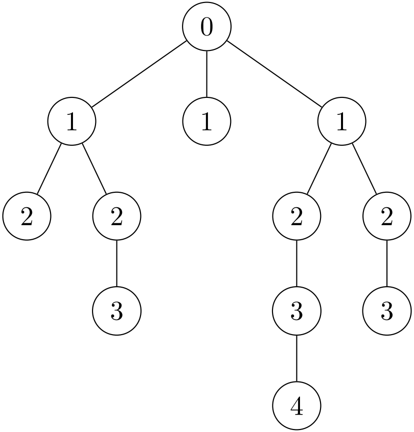


DH 4.2 (b): Number of Nodes at Depth K



$$\text{nodes-at-depth}(r, k) = \begin{cases} \\ \displaystyle\sum_{v:\text{child of } r} \text{nodes-at-depth}(v, k-1), \end{cases}$$

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



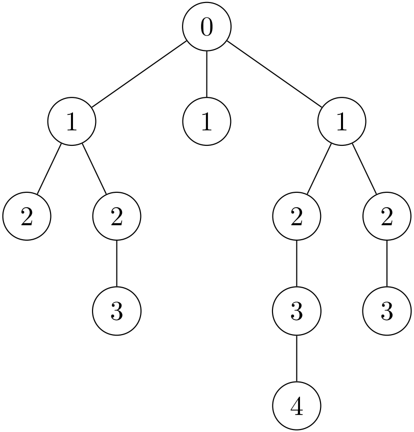
$$\text{nodes-at-depth}(r, k) = \begin{cases} 1, & k = 0 \\ 0, & k > 0 \land r \text{ is a leaf} \\ \displaystyle\sum_{v:\text{child of } r} \text{nodes-at-depth}(v, k-1), & \text{o.w.} \end{cases}$$

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

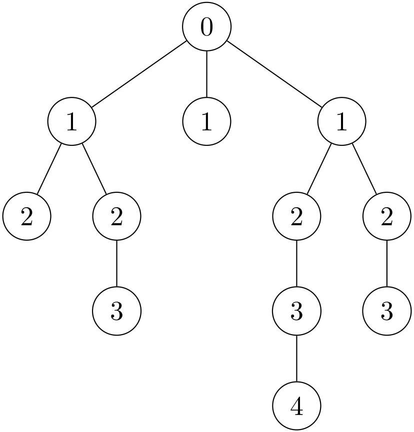
```
procedure NODES-AT-DEPTH()
    return NODES-AT-DEPTH(T, K)

procedure 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 ← 0
    for all child vertex v of r do
        num ← num + NODES-AT-DEPTH(v, k − 1)
    return num
```

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

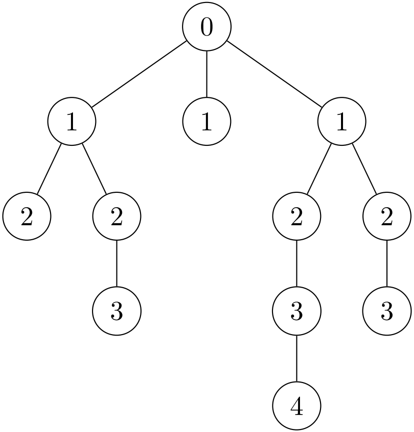


DH 4.2 (c): Any Leaf at an Even Depth?



$$\text{leaf-at-depth}(r, \color{red}{parity}) = \begin{cases} \\ \displaystyle\sum_{v\text{:child of } r} (v, \color{red}{1 - parity}), \end{cases}$$

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



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

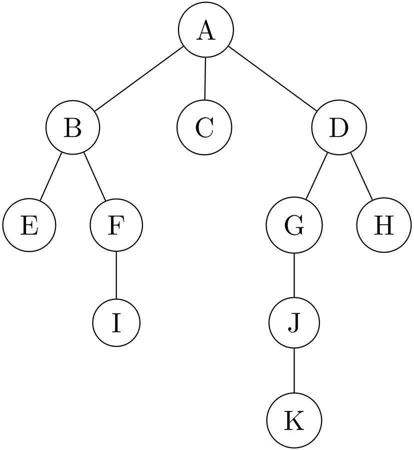
Algorithm 3 Check whether a tree T has any leaf at an even depth.

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

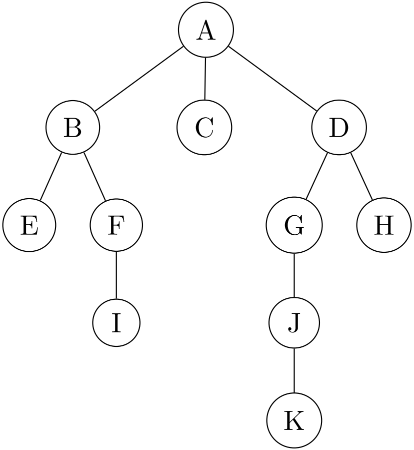
procedure $\textsc{Leaf-at-Depth}(r, parity)$ $\triangleright$ r: root of a tree
    if r is a leaf **then**
        return $1 - parity$

    $result \leftarrow 0$
    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



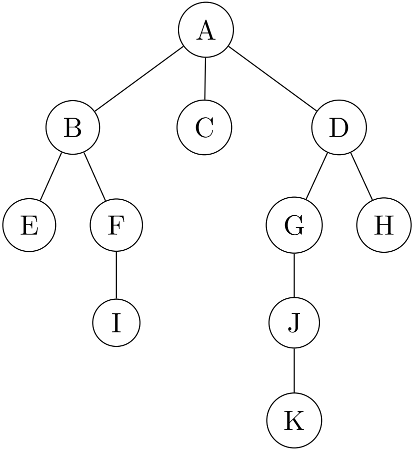
DH 4.3 (a): Sum of Contents at Each Depth



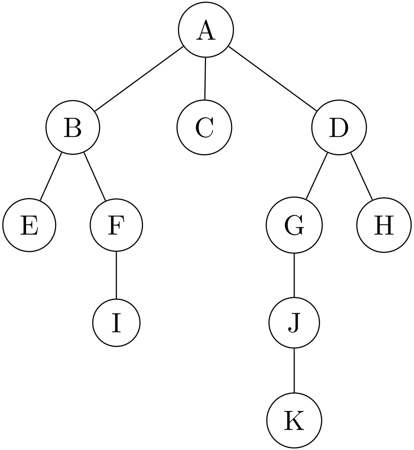
Algorithm 4 Calculate the sum of contents of nodes of a tree T at each depth.

```
procedure SUM-AT-DEPTH(r)                          ▷ r: root of the tree T
    r.depth ← 0
    Q ← ∅
    ENQUEUE(Q, r)
    while Q ≠ ∅ do
        u ← DEQUEUE(Q)
        sumAtDepth[u.depth] += u.content
        for all child vertex v of u do
            v.depth ← u.depth + 1
            ENQUEUE(Q, v)
```

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



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**

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

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

Lower Bound for Comparion-based Sorting

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

Lower Bound for Comparison-based Sorting (DH 6.13)

Prove a lower bound of $O(n \lg n)$ 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.

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.

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.

Computational Model:

the only way to gain order info.

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.

Computational Model:

the only way to gain order info.

$$x \in [1 \cdots 99]$$

$$x/10$$

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.

$$x \in [1 \cdots 99]$$

$$x/10$$

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:

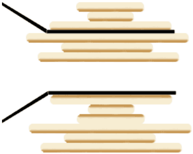
the critical operations to count

Computational Model:

the only way to gain order info.

$$x \in [1 \cdots 99]$$

$$x/10$$



"Bounds For Sorting By Prefix Reversal", 1979

An argument from a student:

- Any comparison-based sorting algorithm can be considered to work by *putting elements into their final positions one by one*. (Take the last time one element is put into its final position.)

An argument from a student:

- Any comparison-based sorting algorithm can be considered to work by *putting elements into their final positions one by one*. (Take the last time one element is put into its final position.)
- To put the element e which is the i-th one being put into its final position, the algorithms *should know the relative ordering between e and the first $n-1$ elements* that are already put into their final positions. This costs at least $\Omega(\lg i)$ comparisons.

An argument from a student:

- Any comparison-based sorting algorithm can be considered to work by *putting elements into their final positions one by one*. (Take the last time one element is put into its final position.)
- To put the element e which is the i-th one being put into its final position, the algorithms *should know the relative ordering between e and the first $n-1$ elements* that are already put into their final positions. This costs at least $\Omega(\lg i)$ comparisons.
- Therefore, *the total number of comparisons* is at least

$$\sum_{i=1}^{n-1} \lg i = \Omega(n \lg n).$$

COMPUTER **SCIENCE** Questions Tags

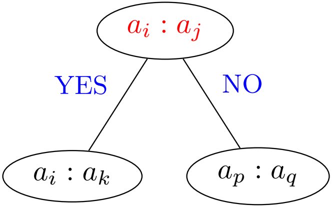
Is this lower bound proof for the comparison-based sorting problem correct?

Is this lower bound proof for the comparison-based sorting problem correct?

"This makes claims without justification." — *D.W.*

Decision Tree Model

Decision Tree Model



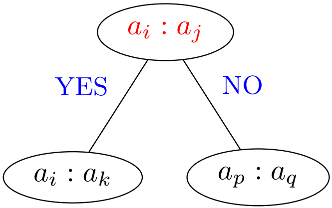
Nodes: comparisions $a_i : a_j$

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

Edges: two-way decisions (Y/N)

Leaves: possible permutations

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

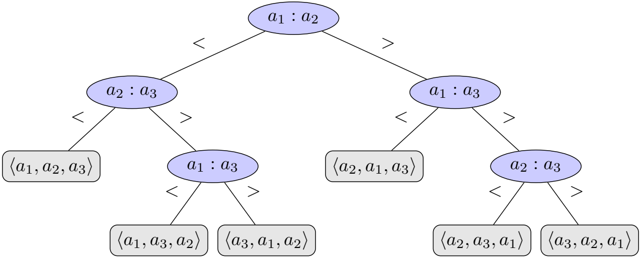
$$\boxed{\text{Any Comparison-based Sorting Algorithm} \xRightarrow{\text{modeled by}} \text{A Decision Tree}}$$

Decision Tree Model

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

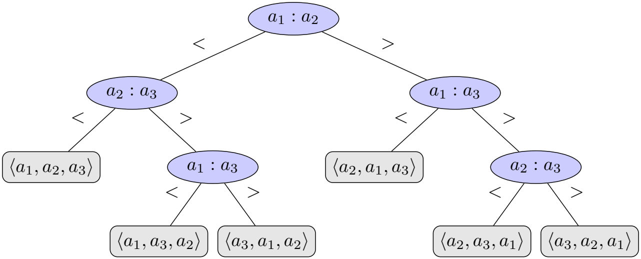
Decision Tree Model

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



Decision Tree Model

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



The decision tree for insertion sort on three elements.

Decision Tree Model

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

Decision Tree Model

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

procedure $\text{-}\textsc{Sort}(A, n)$
  for $i \leftarrow 1$ **to** $n - 1$ **do**
    for $j \leftarrow i + 1$ **to** n **do**
      if $A[j] < A[i]$ **then**
        $\textsc{Swap}(A[j], A[i])$

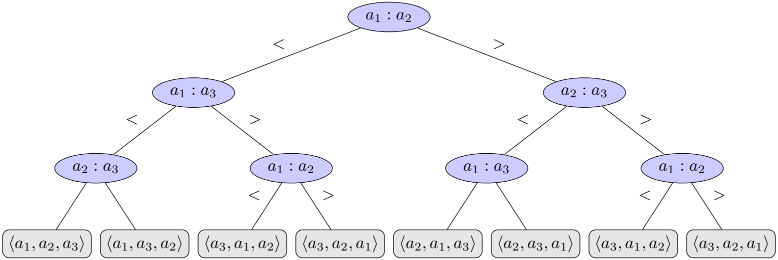
Decision Tree Model

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

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

Decision Tree Model

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

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

Decision Tree Model

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

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

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

Decision Tree Model

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

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

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

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

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

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:

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

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:

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

To be a correct sorting algorithm:

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

Lower Bound for Comparison-based Sorting

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

Lower Bound for Comparison-based Sorting

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

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

Lower Bound for Comparison-based Sorting

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

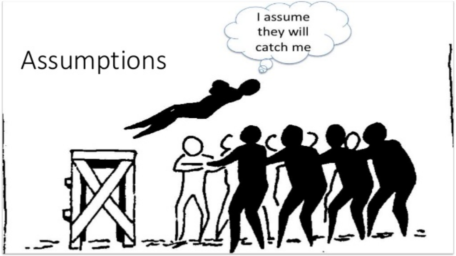
$$h \ge \lg n! = \Omega(n \lg n)$$

Stirling Formula (by *James Stirling*):

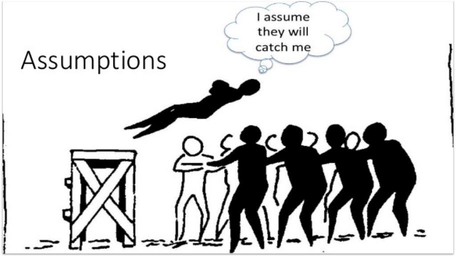
$$n! = \Theta\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right)$$

Looking Back

Looking Back



Looking Back



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

(a) Comparison-based sorting algorithms.

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

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.

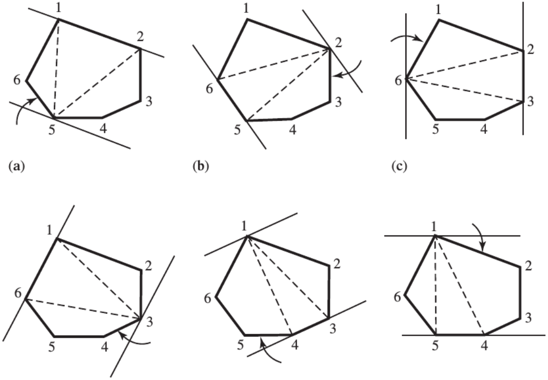
(d) Prove that any comparison-based algorithm to completely sort a k-sorted array requires $\Omega(n \log(n/k))$ comparisons in the worst case.

Convex Polygon Diameter (DH 6.8)

Show that the “Convex Polygon Diameter” algorithm is of linear-time complexity.

Convex Polygon Diameter (DH 6.8)

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



(a) (b) (c)

(d) (e) (f)

Convex Polygon Diameter (DH 6.8)

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

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

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**

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?

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

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

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?

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

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

Theorem (A Simple Observation)

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

Theorem (A Simple Observation)

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

Proof.

□

Theorem (A Simple Observation)

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

Proof.

□

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

Theorem (A Simple Observation)

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

Proof.

□

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

We have enumerated *all* pairs of vertices

Theorem (A Simple Observation)

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

Proof.

□

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

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

Definition (Line of Support)

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

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

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 (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.

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.

Theorem

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

Proof.

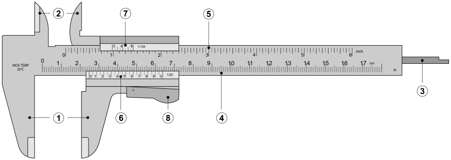
Theorem

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

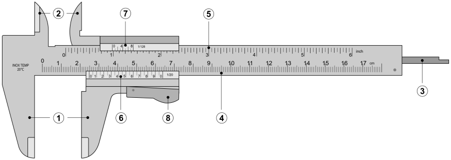
Proof.

$\square$

Rotating Caliper



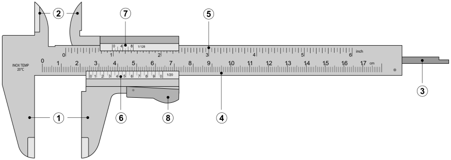
Rotating Caliper



"Computational Geometry"

Ph.D Thesis, Michael Shamos, 1978

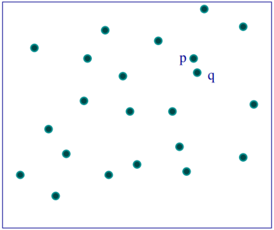
Rotating Caliper



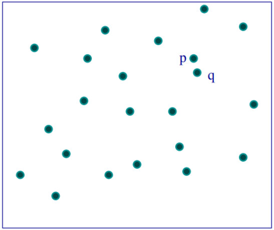
"Computational Geometry"
Ph.D Thesis, Michael Shamos, 1978

"Solving Geometric Problems with the Rotating Calipers", 1983

Finding the Closest Pair of Points

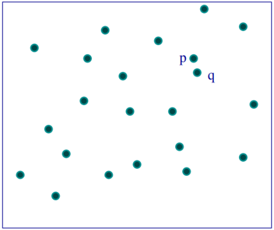


Finding the Closest Pair of Points



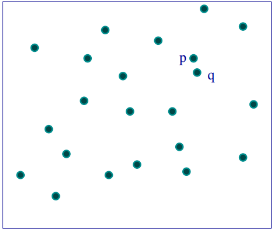
A Classic and Beautiful Divide-Conquer Algorithm:

Finding the Closest Pair of Points



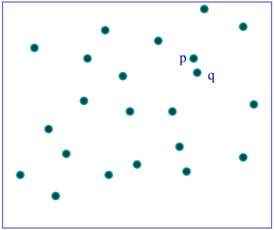
A Classic and Beautiful Divide-Conquer Algorithm:

Finding the Closest Pair of Points



A Classic and Beautiful Divide-Conquer-Combine Algorithm:

Finding the Closest Pair of Points



A Classic and Beautiful Divide-Conquer-Combine Algorithm:

Section 33.4, CLRS


TREASURE
HUNT
on foot


TREASURE
HUNT
on foot


Hints
&
Tips


Help
Wanted

Repeated Elements Problem


Thank You!