§The Complexity of Algorithms and the Lower Bounds of Problems

- **The goodness of an algorithm**

time complexity (more important)
space complexity
for a parallel algorithm :
    time-processor product
for a VLSI circuit :
    area-time (AT, AT^2)

- **Measure the goodness of an <u>algorithm</u>**

time complexity of an algorithm
    worst-case
    average-case
    amortized

- **Measure the difficulty of a problem**

NP-complete
undecidable

- **Is the algorithm best?**

optimal (algorithm)

We can use # of comparisons to measure a sorting algorithm.

● Asymptotic notations

Def: $f(n) = O(g(n))$ "at most"

$\exists c$, and n_0, э $|f(n)| \le c|g(n)| \quad \forall\ n \ge n_0$

e.g. $f(n) = 3n^2 + 2$

$g(n) = n^2$

$\Rightarrow n_0=2, c=4$

$\therefore f(n) = O(n^2)$

e.g. $f(n) = n^3 + n = O(n^3)$

Def: $f(n) = \Omega(g(n))$ "at least"

"lower bound"

$\exists$ c, and n_0, э $|f(n)| \ge c|g(n)| \ \forall\ n \ge n_0$

Def: $f(n) = \Theta(g(n))$

$\exists\ c_1, c_2$, and n_0, э $c_1|g(n)| \le |f(n)| \le c_2|g(n)| \ \forall\ n \ge n_0$

Def: $f(n) \sim o(g(n))$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \to 1$$

e.g. $f(n) = 3n^2+n = o(3n^2)$

Problem size

	10	10^2	10^3	10^4
$\log_2 n$	3.3	6.6	10	13.3
n	10	10^2	10^3	10^4
$n\log_2 n$	$0.33\text{x}10^2$	$0.7\text{x}10^3$	10^4	$1.3\text{x}10^5$
n^2	10^2	10^4	10^6	10^8
2^n	1024	$1.3\text{x}10^{30}$	$>10^{100}$	$>10^{100}$
n!	$3\text{x}10^6$	$>10^{100}$	$>10^{100}$	$>10^{100}$

Table 2.1 Time Complexity Functions

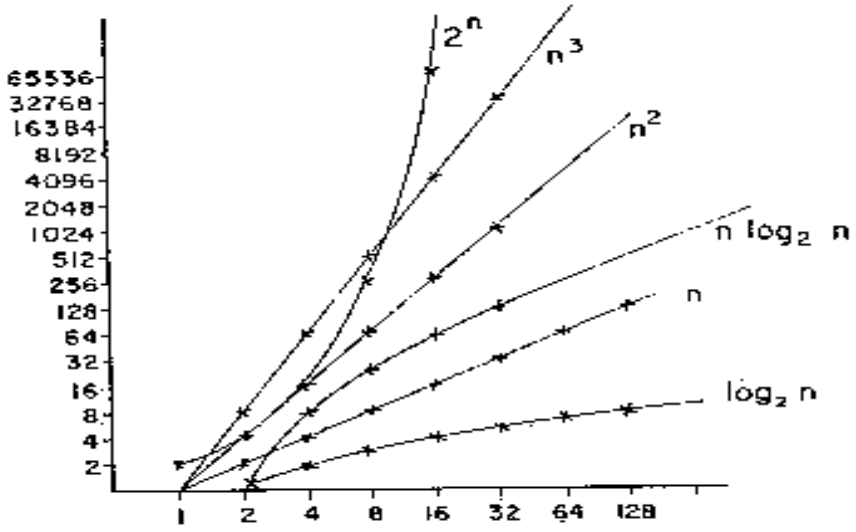


Figure: Rate of growth of common computing time functions

$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!) < O(n^n)$

exponential algorithm: $O(2^n)$

polynomial algorithm

Algorithm A: $O(n^3)$

Algorithm B: $O(n)$

Should Algorithm B run faster than A?

NO!

It is true only when n is large enough!

- **Analysis of algorithms**
  - best case: easiest
  - worst case
  - average case: hardest

● straight insertion sort

input: 7,5,1,4,3

7,5,1,4,3

5,7,1,4,3

1,5,7,4,3

1,4,5,7,3

1,3,4,5,7

Algorithm 2.1 Straight Insertion Sort

Input: $x_1,x_2,...,x_n$

Output: The sorted sequence of $x_1,x_2,...,x_n$

```
For j := 2 to n do
Begin
    i := j-1
    x := x_j
    While x<x_i and i > 0 do
    Begin
        x_{i+1} := x_i
        i := i-1
    End
    x_{i+1} := x
End
```

● Inversion table

$(a_1,a_2,...,a_n)$: a permutation of $\{1,2,...,n\}$

$(d_1,d_2,...,d_n)$: the inversion table of $(a_1,a_2,...a_n)$

d_j: the number of elements to the left of j that are greater than j

e.g. (7 5 1 4 3 2 6)

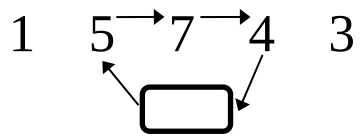
2 4 3 2 1 1 0 inversion table

e.g. (7 6 5 4 3 2 1)

6 5 4 3 2 1 0 inversion table

- M: # of data movements

$$M = \sum_{i=1}^{n-1} (2 + d_i)$$



$d_3=2$

best case: already sorted

$d_i = 0$ for $1 \le i \le n$

$\Rightarrow M = 2(n - 1) = O(n)$

worst case: reversely sorted

$d_1 = n - 1$

$d_2 = n - 2$

:

$d_i = n - i$

:

$d_n = 0$

$$\sum_{i=0}^{n-1} d_i = \frac{n}{2}(n - 1)$$

$$\Rightarrow M = \frac{n}{2}(n - 1)+2(n - 1) = O(n^2)$$

average case:

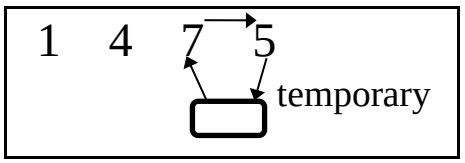
x_j is being inserted into the sorted sequence x_1 x_2 ... x_{j-1}

the probability that x_j is the largest: $\frac{1}{j}$

takes 2 data movements

the probability that x_j is the second largest: $\frac{1}{j}$

takes 3 data movements



:

:

# of movements for x_j to be inserted:

$$\frac{2}{j} + \frac{3}{j} + \ldots + \frac{j+1}{j} = \frac{j+3}{2}$$

$$M = \sum_{j=2}^{n} \frac{j+3}{2} = \frac{1}{4}(n + 8)(n - 1) = O(n^2)$$

<u>Alternative view</u> (average case)

Only consider # of exchanges.

<u>Method 1</u>

x_j is being inserted into the sorted sequence x_1 x_2 .... x_{j-1}

If x_j is the *k*th ($1 \le k \le j$) largest, it takes (k–1) exchanges.

e.g. 1 5 7↔4

1 5↔4 7

1 4 5 7

# of exchanges required for x_j to be inserted:

$$\frac{0}{j} + \frac{1}{j} + \frac{2}{j} + \ldots + \frac{j-1}{j} = \frac{j-1}{2}$$

$\Rightarrow$ # of exchanges for sorting:

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

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

$$= \frac{1}{2} \cdot \frac{(n-1)(n+2)}{2} - \frac{n-1}{2}$$

$$= \frac{1}{4} n(n-1)$$

<u>Method 2</u> (using inversion table) [Knuth 1973]

$I_n(k)$: # of permutations which have exactly k inversions

$P_n(k)$: the probability that a given permutation of n numbers has k inversions

$P_n(k)=I_n(k)/n!$

average # of inversions(exchanges):

$$\sum_{k=0}^{m} k\, P_n(k), \text{ where } m = \frac{n(n-1)}{2}$$

$I_n(k)$:

$k<n$ ↑

$I_{n-1}(k)$	largest
$I_{n-1}(k-1)$	2nd
$I_{n-1}(k-2)$	3rd
	:
$I_{n-1}(0)$	$(k+1)$th largest

$k \geq n$ ↑

$I_{n-1}(k)$	largest
$I_{n-1}(k-1)$	2nd
	:
$I_{n-1}(k-(n-1))$	nth largest (smallest)

$$\begin{cases} I_n(k)=I_{n-1}(0)+I_{n-1}(1)+\ldots+I_{n-1}(k), & k<n \\ I_n(k)=I_{n-1}(k-n+1)+\ldots+I_{n-1}(k), & k\geq n \end{cases}$$

$$I_n(k)=\sum_{i=0}^{k} I_{n-1}(i),\ k< n$$

$$I_n(k)=\sum_{i=k-n+1}^{k} I_{n-1}(i-1),\ k\geq n$$

Prove

$$\sum_{k=0}^{m} k\, I_n(k)= \frac{n!}{4}n(n-1),\ \text{where } m = \frac{n(n-1)}{2}$$

Proof:

by induction

- $n=2,\quad m = \frac{n(n-1)}{2} = 1$

$$I_2(1) = \frac{2!}{4}\cdot 2\cdot 1 = 1$$

- Assume that it is true for n.

  We shall prove

$$\sum_{k=0}^{m'} k\, I_{n+1}(k) = \frac{(n+1)!}{4} n(n+1), \quad m' = \frac{n(n+1)}{2}$$

$$= m + n$$

$$\sum_{k=0}^{m'} k\, I_{n+1}(k) = \sum_{k=0}^{m+n} k\, I_{n+1}(k)$$

$$= \sum_{k=0}^{n} k \sum_{i=0}^{k} I_n(i) + \sum_{k=n+1}^{m+n} k \sum_{i=k-n}^{k} I_n(i) \quad \cdots\cdots\cdots\cdots \textbf{(1)}$$

$$(\because \quad I_{n+1}(k) = \sum_{i=0}^{k} I_n(i), \; k<n+1$$

$$I_{n+1}(k) = \sum_{i=k-n}^{k} I_n(i), \; k \geq n+1)$$

Let (s, t) denote $s \cdot I_n(t)$

k									
0	(0,0)								
1	(1,0)	(1,1)							
2	(2,0)	(2,1)	(2,2)						
:									
n	(n,0)	(n,1)	⋯		(n,n)				
n+1		(n+1,1)	(n+1,2)	⋯	⋯	(n+1,n+1)			
n+2			(n+2,2)	⋯	⋯	(n+2,n+1)	(n+2,n+2)		
:									
m								(m,n-m) ⋯	(m,m)
:									
m+n									(m+n,m)

($I_n(m+1) = I_n(m+2) = \cdots = I_n(m+n) = 0$)

(1) can be rewritten as:

$$\sum_{i=0}^{m} I_n(i) \sum_{k=0}^{n} (i + k)$$

$$= \sum_{k=0}^{n} \sum_{i=0}^{m} (i + k) I_n(i)$$

$$= \sum_{k=0}^{n} \sum_{i=0}^{m} i\, I_n(i) + \sum_{k=0}^{n} k \sum_{i=0}^{m} I_n(i)$$

$$\left(\sum_{i=0}^{m} i\, I_n(i) = \frac{n!}{4} n(n-1),\ \sum_{i=0}^{m} I_n(i) = n! \right)$$

$$= \sum_{k=0}^{n} \frac{n!}{4} n(n-1) + \sum_{k=0}^{n} k\, n!$$

$$= \frac{n!}{4} n(n-1)(n+1) + n! \frac{n(n+1)}{2}$$

$$= \frac{n!}{4} (n+1)(n(n-1)+2n)$$

$$= \frac{(n+1)!}{4} n(n+1)$$

Therefore

$$\sum_{k=0}^{m} k\, I_n(k) = \frac{n!}{4} n(n-1), \text{ where } m = \frac{n(n-1)}{2}$$

$\Rightarrow$average # of inversions:

$$\sum_{k=0}^{m} \frac{k I_n(k)}{n!} = \frac{1}{4} n(n-1)$$

<u>Method 3</u> (using inversion table and generating function)[Kronsjö 1979]

$I_n(k)$

n\k	0	1	2	3	4	5	6
1	1	0	0	0	0	0	0
2	1	1	0	0	0	0	0
3	1	2	2	1	0	0	0
4	1	3	5	6	5	3	1

Assume we have $I_3(k)$, $0 \le k \le 3$, and $a_4 > a_i$, $1 \le i \le 3$

$\leftarrow \quad a_1 \quad a_2 \quad a_3 \quad a_4 \quad \uparrow \quad a_1 \quad a_2 \quad a_3 \quad a_4$

largest $G_3(Z)$ — second largest $ZG_3(Z)$

$\rightarrow \quad a_1 \quad a_2 \quad a_3 \quad a_4 \quad \downarrow \quad a_1 \quad a_2 \quad a_3 \quad a_4$

third largest $Z^2G_3(Z)$ — smallest $Z^3G_3(Z)$

	$I_4(0)$	$I_4(1)$	$I_4(2)$	$I_4(3)$	$I_4(4)$	$I_4(5)$	$I_4(6)$
$\leftarrow$	$I_3(0)$	$I_3(1)$	$I_3(2)$	$I_3(3)$			
$\uparrow$		$I_3(0)$	$I_3(1)$	$I_3(2)$	$I_3(3)$		
$\rightarrow$			$I_3(0)$	$I_3(1)$	$I_3(2)$	$I_3(3)$	
$\downarrow$				$I_3(0)$	$I_3(1)$	$I_3(2)$	$I_3(3)$

generating function for $I_n(k)$

$$G_n(Z) = \sum_{k=0}^{m} I_n(k) Z^k$$

for n = 4

$G_4(Z) = 1 + 3Z + 5Z^2 + 6Z^3 + 5Z^4 + 3Z^5 + Z^6$

$= (1 + Z + Z^2 + Z^3)G_3(Z)$

in general,

$G_n(Z) = (1 + Z + Z^2 + \cdots + Z^{n-1})G_{n-1}(Z)$

generating function for $P_n(k)$:

$$g_n(Z) = \sum_{k=0}^{m} P_n(k)Z^k = \sum_{k=0}^{m} \frac{I_n(k)}{n!}Z^k$$

$$= \frac{1}{n!}G_n(Z)$$

$$= \frac{1+Z+Z^2+\cdots+Z^{n-1}}{n} \cdot \frac{1+Z+Z^2+Z^{n-2}}{} \cdots \frac{1+Z}{2} \cdot 1$$

$$\sum_{k=0}^{m} kP_n(k) = g_n'(1)$$

$$= \frac{1+2+\cdots+(n-1)}{n} + \frac{1+2+\cdots+(n-2)}{n-1} + \cdots + \frac{1}{2} + 0$$

$$= \frac{n-1}{2} + \frac{n-2}{2} + \cdots + \frac{1}{2} + 0$$

$$= \frac{1}{4}n(n-1)$$

- **Binary search**

e.g.

sorted sequence: (search 9)

1 4 5 7 9 10 12 15

step 1 ↑

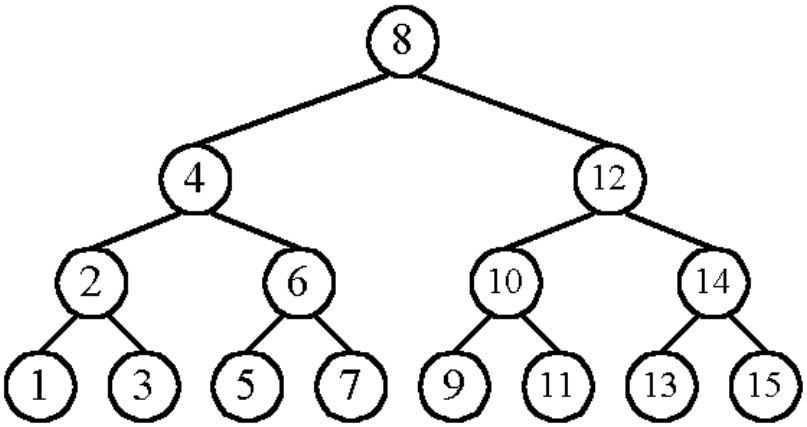
step 2 ↑

step 3 ↑

best case: 1 step = O(1)

worst case: ($\lfloor \log_2 n \rfloor$+1) steps = O(log n)

average case: O(log n)



n cases for successful search

n+1 cases for unsuccessful search

the average # of comparisons done in the binary tree:

$A(n) = \frac{1}{2n+1}\left(\sum_{i=1}^{k} i\,2^{i-1} + k(n+1)\right)$, where $k = \lfloor \log n \rfloor + 1$

successful unsuccessful

Assume $n=2^k$

$\sum_{i=1}^{k} i\,2^{i-1} = 2^k(k-1)+1$ proved by induction on k

$A(n) = \frac{1}{2n+1}((k-1)2^k + 1 + k(2^k+1))$

$\approx k$ as n is very large

$= \log n$

$= O(\log n)$

- **Straight selection sort**

e.g.

7 5 1 4 3

1 5 7 4 3

1 3 7 4 5

1 3 4 7 5

1 3 4 5 7

Only consider # of changes in the flag which is used for selecting the smallest number in each iteration.

best case: O(1)

worst case: $O(n^2)$

average case: O(n log n)

- **Quick sort**

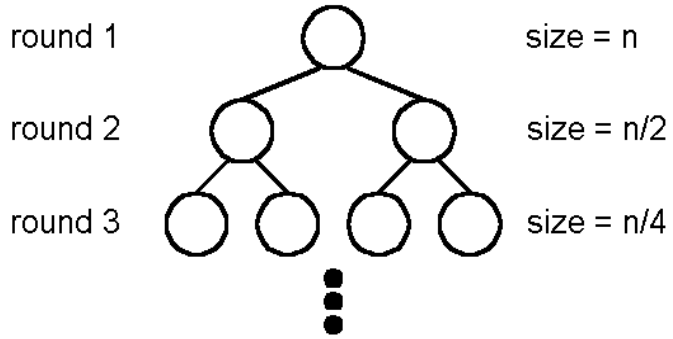
e.g.

11	5	24	2	31	7	8	26	10	15
		↑						↑	
11	5	10	2	31	7	8	26	24	15
				↑		↑			
11	5	10	2	8	7	31	26	24	15
△					△				
7	5	10	2	8	11	31	26	24	15
|←		<11		→|		|←	>11	→|

Recursively apply the same procedure.

best case: O(n log n)

A list is split into two sublists with almost equal size.



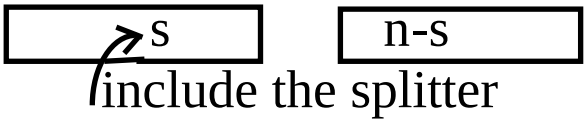
log n rounds are needed.

In each round, n comparisons (ignoring the element used to split) are required.

worst case: $O(n^2)$
In each round, the number used to split is either the smallest or the largest.

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

average case: $O(n \log n)$



$$T(n) = \underset{1 \le s \le n}{\text{Avg}}\,(T(s) + T(n-s)) + cn$$

$$= \frac{1}{n}\sum_{s=1}^{n}(T(s) + T(n-s)) + cn$$

$$= \frac{1}{n}(T(1)+T(n-1)+T(2)+T(n-2)+\cdots+T(n)+T(0)) + cn,\ T(0)=0$$

$$= \frac{1}{n}(2T(1)+2T(2)+\cdots+2T(n-1)+T(n)) + cn$$

$$(n-1)T(n) = 2T(1)+2T(2)+\cdots+2T(n-1) + cn^2 \cdots\cdots (1)$$

$$(n-2)T(n-1) = 2T(1)+2T(2)+\cdots+2T(n-2)+c(n-1)^2 \cdots (2)$$

(1) – (2)

$$(n-1)T(n) - (n-2)T(n-1) = 2T(n-1)+c(2n-1)$$

$$(n-1)T(n) - nT(n-1) = c(2n-1)$$

$$\frac{T(n)}{n} = \frac{T(n-1)}{n-1} + c\left(\frac{1}{n} + \frac{1}{n-1}\right)$$

$$= c\left(\frac{1}{n} + \frac{1}{n-1}\right) + c\left(\frac{1}{n-1} + \frac{1}{n-2}\right) + \cdots + c\left(\frac{1}{2} + 1\right) + T(1),\ T(1) = 0$$

$$= c\left(\frac{1}{n} + \frac{1}{n-1} + \ldots + \frac{1}{2}\right) + c\left(\frac{1}{n-1} + \frac{1}{n-2} + \ldots + 1\right)$$

Harmonic number[Knuth 1986]

$H_n = 1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}$

$=\ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} - \varepsilon$, where $0<\varepsilon<\frac{1}{252n^6}$

$\gamma = 0.5772156649\cdots$.

$H_n = O(\log n)$

$\frac{T(n)}{n} = c(H_n - 1) + cH_{n-1}$

$= c(2H_n - \frac{1}{n} - 1)$

$\Rightarrow T(n) = 2cnH_n - c(n+1)$

$= O(n \log n)$

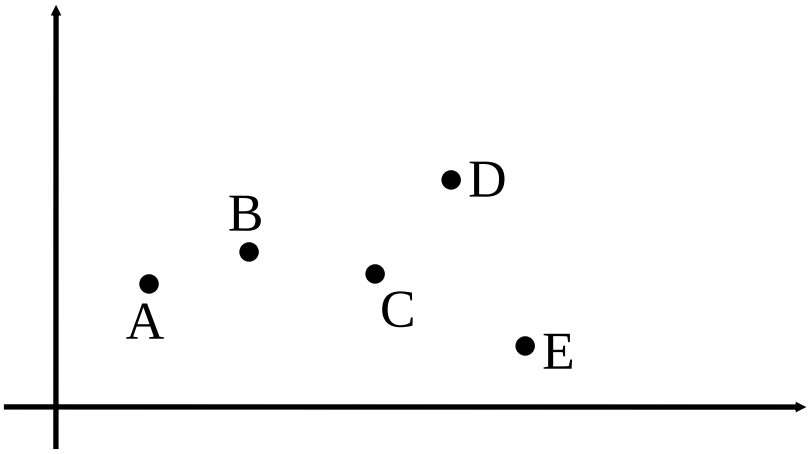
● 2-D rank finding

Def: $A = (a_1,a_2)$, $B = (b_1,b_2)$

A dominates B iff $a_1 > b_1$ and $a_2 > b_2$.

Def: Given a set s of n point, the rank of a point x is the number of points dominated by x.

e.g.



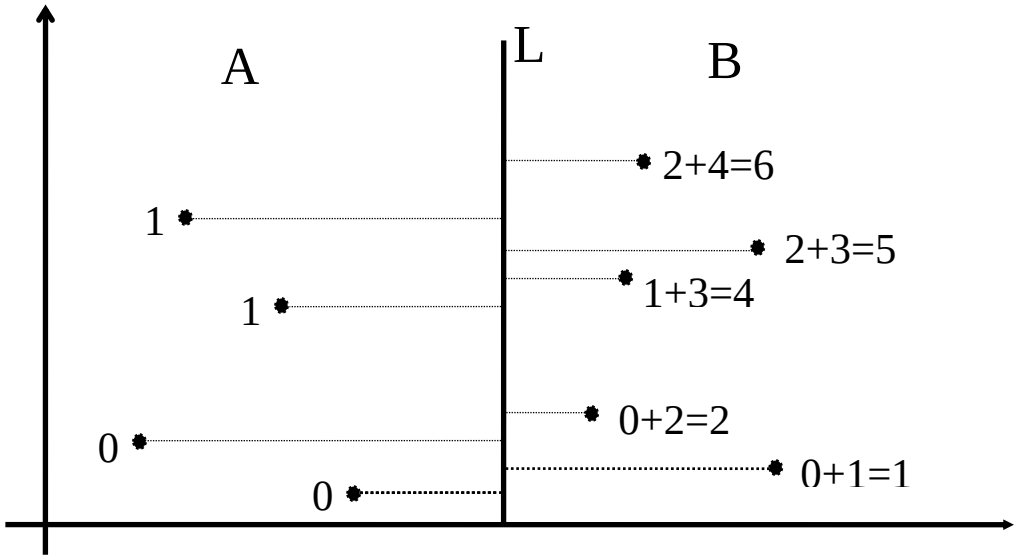
rank(A) = 0 rank(B) = 1 rank(C) = 1

rank(D) = 3 rank(E) = 0

straightforward algorithm:

compare all pairs of points: $O(n^2)$

more efficient algorithm



step 1: Split the points along the median line L into A and B.

step 2: Find ranks of points in A and ranks of points in B, recursively.

step 3: Sort points in A and B according to their y-values. Update the ranks of points in B.

time complexity:

step 1: O(n) (finding median)

step 3: O(n log n) (sorting)

total time complexity

$$T(n) \leq 2T(\tfrac{n}{2}) + c_1 n \log n + c_2 n$$

$$\leq 2T(\tfrac{n}{2}) + c\, n \log n$$

$$\leq 4T(\tfrac{n}{4}) + c\, n \log \tfrac{n}{2} + c\, n \log n$$

$$\leq nT(1) + c(n \log n + n \log \tfrac{n}{2} + n \log \tfrac{n}{4} + \cdots + n \log 2)$$

$$= nT(1) + \frac{cn \log n(\log n + \log 2)}{2}$$

$$= O(n \log^2 n)$$

- **Lower bound**

Def: A lower bound of a problem is the least time complexity required for any algorithm which can be used to solve this problem.

☆ **worst case lower bound**

average case lower bound

The lower bound for a problem is not unique.

e.g. $\Omega(1)$, $\Omega(n)$, $\Omega(n \log n)$ are all lower bounds for sorting.

($\Omega(1)$, $\Omega(n)$ are trivial)

(1)At present, if the highest lower bound of a problem is $\Omega(n \log n)$ and the time complexity of the best algorithm is $O(n^2)$.

(i)We may try to find a higher lower bound.

(ii)We may try to find a better algorithm.

(iii)Both of the lower bound and the algorithm may be improved.

(2)If the present lower bound is $\Omega(n \log n)$ and there is an algorithm with time complexity $O(n \log n)$, then the algorithm is optimal.

- **The worst case lower bound of sorting**

6 permutations for 3 data elements

a_1	a_2	a_3
1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

straight insertion sort:

input data: (2, 3, 1)

(1) $a_1:a_2$

(2) $a_2:a_3$, $a_2 \leftrightarrow a_3$

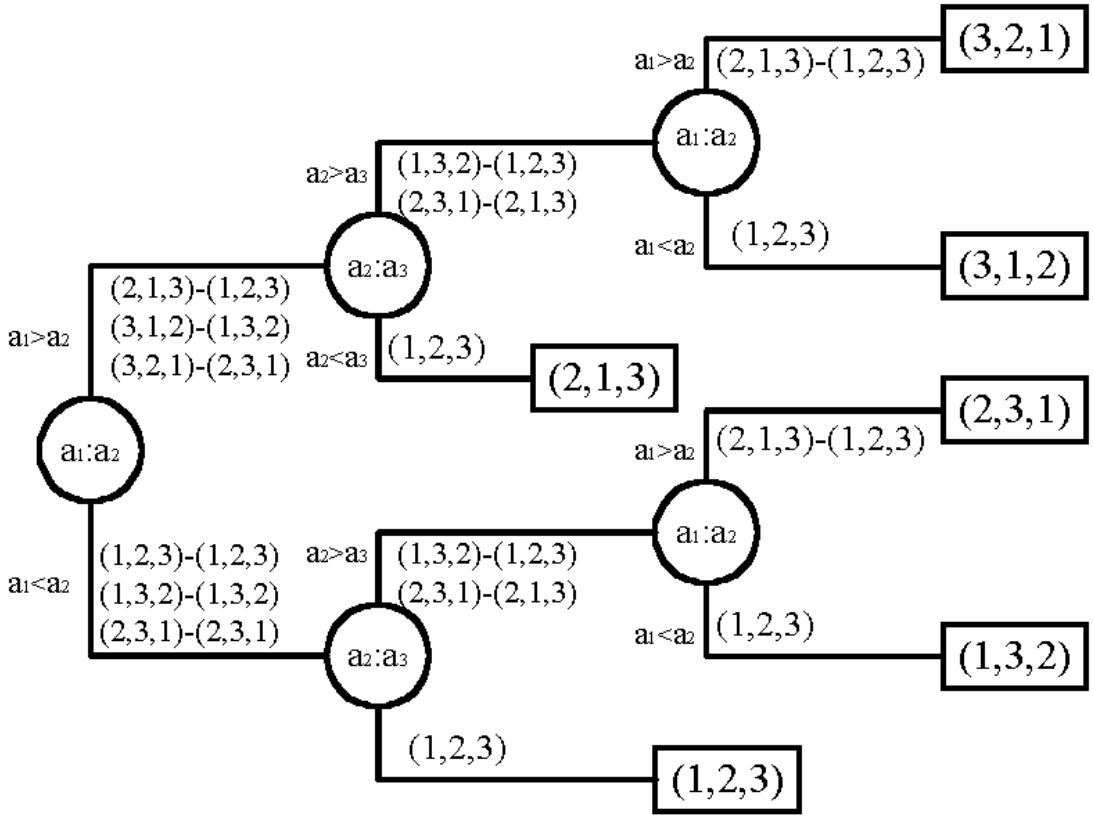
(3) $a_1:a_2$, $a_1 \leftrightarrow a_2$

input data: (2, 1, 3)

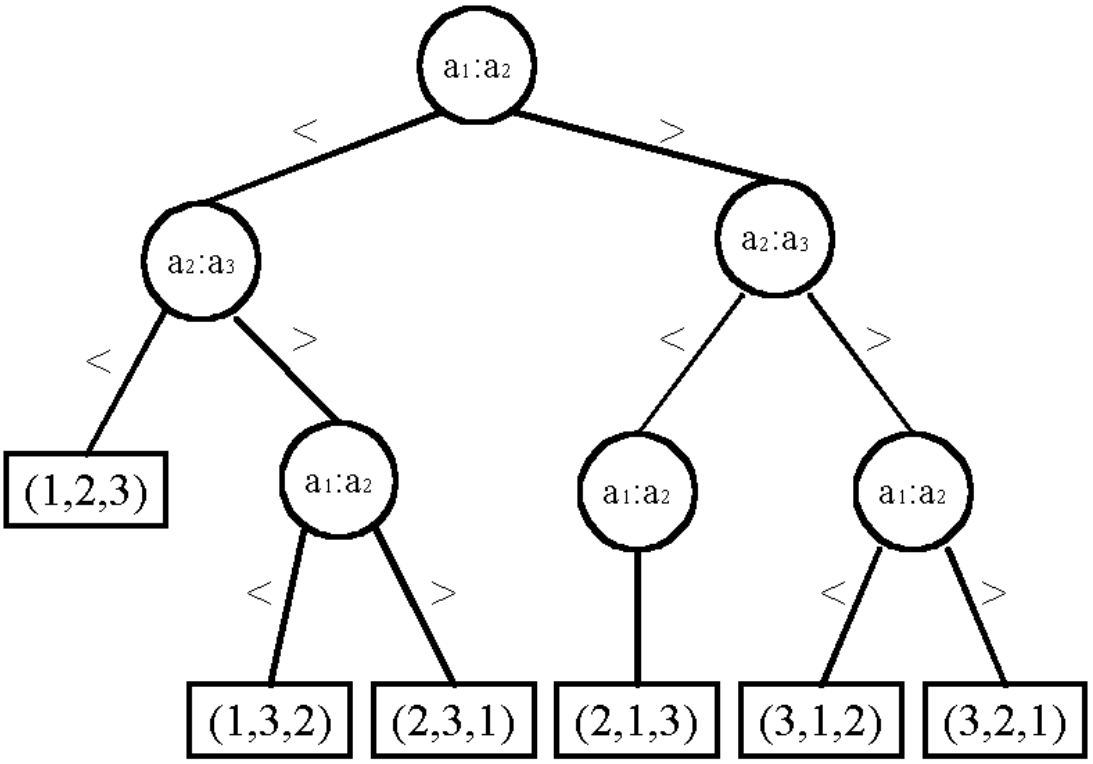
(1)$a_1:a_2$, $a_1 \leftrightarrow a_2$

(2)$a_2:a_3$

decision tree for straight insertion sort:



decision tree for bubble sort:



- To find the lower bound, we have to find the smallest depth of a binary tree.
- n! distinct permutations

n! leaf nodes in the binary decision tree.

- balanced tree has the smallest depth:

$\lceil \log(n!) \rceil = \Omega(n \log n)$

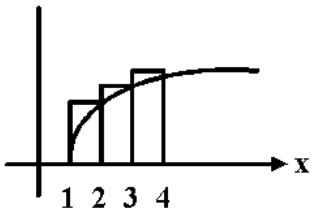
lower bound for sorting: $\Omega(n \log n)$

Method 1:

$\log(n!) = \log(n(n-1)\cdots 1)$

$= \log 2 + \log 3 + \cdots + \log n$

$> \int_1^n \log x dx$



$= \log e \int_1^n \ln x dx$

$= \log e[x \ln x - x]_1^n$

$= \log e(n \ln n - n + 1)$

$= n \log n - n \log e + 1.44$

$\geq n \log n - 1.44n$

$= \Omega(n \log n)$

Method 2:

Stirling approximation:

$n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$

$\log n! \approx \log\sqrt{2\pi} + \frac{1}{2}\log n + n \log\frac{n}{e}$

$\approx n \log n$

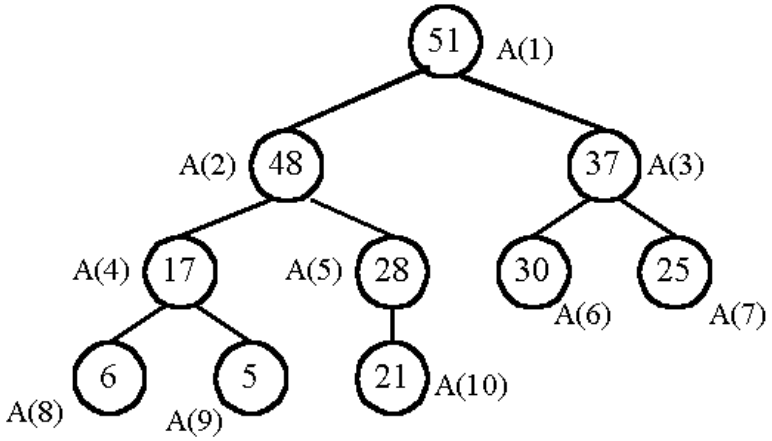
$\approx \Omega(n \log n)$

n	n!	S_n
1	1	0.922
2	2	1.919
3	6	5.825
4	24	23.447
5	120	118.02
6	720	707.39
10	3,628,800	3,598,600
20	2.433×10^{18}	2.423×10^{18}
100	9.333×10^{157}	9.328×10^{157}

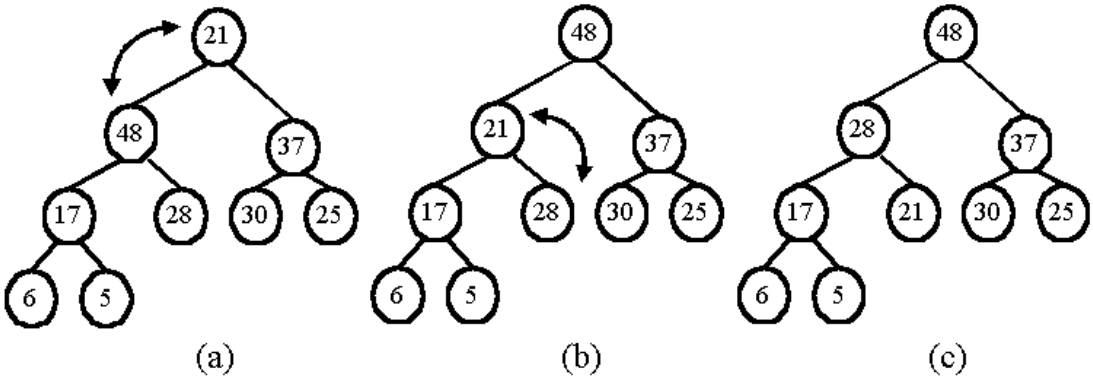
- **Heapsort—An optimal sorting algorithm**

a heap:

parent ≥ son



output the maximum and restore:

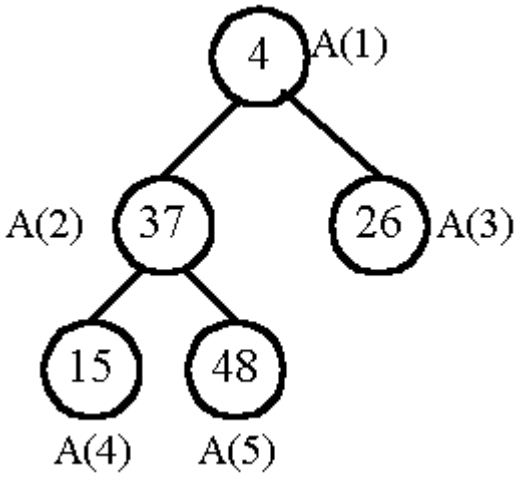


Heapsort: ← **construction**

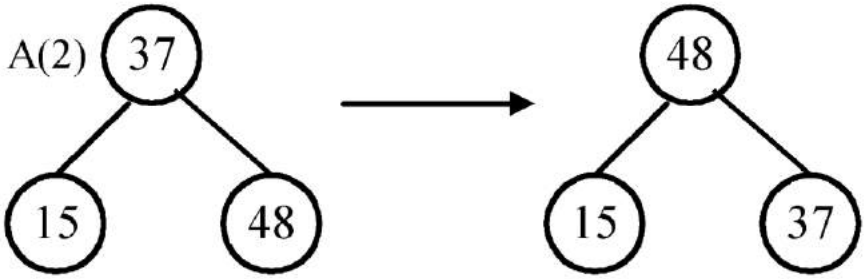
↑**output**

Step 1: <u>construction</u>:

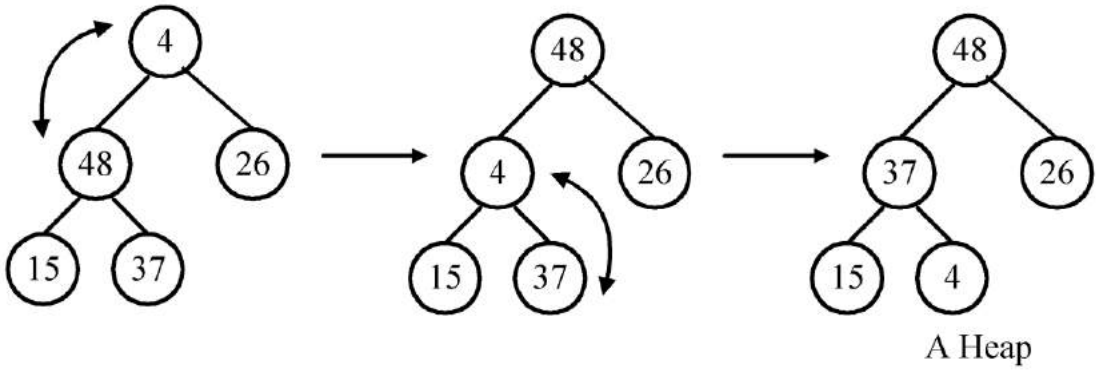
input data: 4, 37, 26, 15, 48



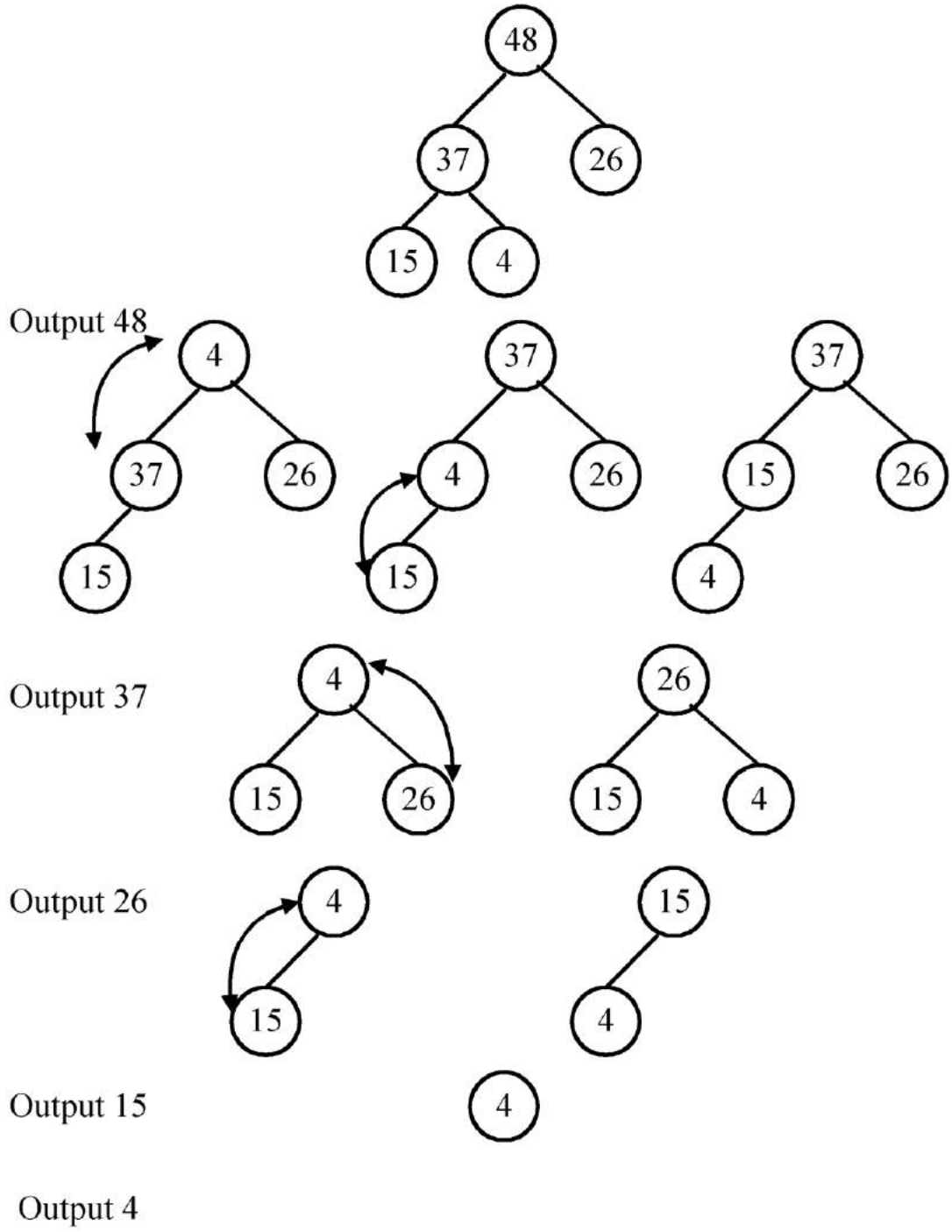
restore the subtree rooted at A(2):



restore the tree rooted at A(1):



Step 2: <u>output</u>:



<u>implementation</u>

using a linear array

not a binary tree.

The sons of A(h) are A(2h) and A(2h+1).

time complexity: O(n log n)

Step 1: <u>construction</u>

$d = \lfloor \log n \rfloor$: depth

# of comparisons is at most:

$$\sum_{L=0}^{d-1} 2(d-L)2^L$$

$$= 2d \sum_{L=0}^{d-1} 2^L - 4 \sum_{L=0}^{d-1} L2^{L-1}$$

$$\left(\sum_{L=0}^{k} L2^{L-1} = 2^k(k-1)+1 \right)$$

$$= 2d(2^d-1) - 4(2^{d-1}(d-1-1)+1)$$

:

$$= cn - 2\lfloor \log n \rfloor - 4, \quad 2 \le c \le 4$$

Step 2: <u>output</u>

$$2 \sum_{i=1}^{n-1} \lfloor \log i \rfloor$$

= :

$$= 2n\lfloor \log n \rfloor - 4cn + 4, \quad 2 \le c \le 4$$

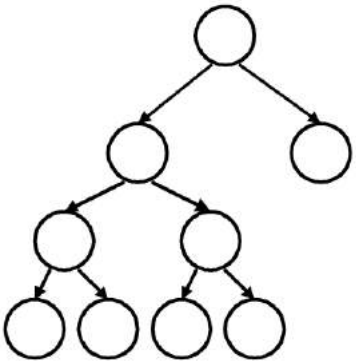
$$= O(n \log n)$$

- Average case lower bound of sorting
  - Binary decision tree
  - The average time complexity of a sorting algorithm:

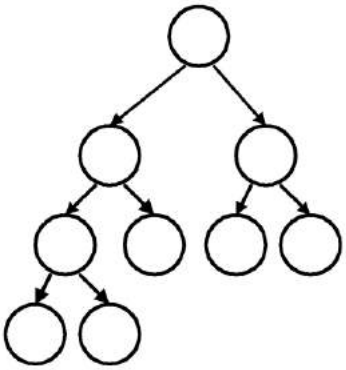
$$\frac{\text{the external path length of the binary tree}}{n!}$$

  - The external path length is minimized if the tree is balanced

(all leaf nodes on level d or level d–1)



unbalanced
external path length
$= 4 \cdot 3 + 1 = 13$



balanced
external path length
$= 2 \cdot 3 + 3 \cdot 2 = 12$

- compute the min external path length

(1)depth of balanced binary tree with c leaf nodes:
$d = \lceil \log c \rceil$
Leaf nodes can appear only on level d or d–1.
(2)x_1 leaf nodes on level d–1
x_2 leaf nodes on level d
$x_1 + x_2 = c$

$$x_1 + \frac{x_2}{2} = 2^{d-1}$$

$\Rightarrow x_1 = 2^d - c$
$x_2 = 2(c - 2^{d-1})$
(3)external path length:
$M = x_1(d-1) + x_2 d$
$= (2^d - 1)(d-1) + 2(c - 2^{d-1})d$
$= c(d-1) + 2(c - 2^{d-1}), \quad d - 1 = \lfloor \log c \rfloor$
$= c\lfloor \log c \rfloor + 2(c - 2^{\lfloor \log c \rfloor})$

(4)c = n!

$M = n!\lfloor \log n! \rfloor + 2(n! - 2^{\lfloor \log n! \rfloor})$

$$\frac{M}{n!} = \lfloor \log n! \rfloor + 2\frac{n! - 2^{\lfloor \log n! \rfloor}}{n!}$$

$= \lfloor \log n! \rfloor + c, \quad 0 \le c \le 1$

$= \Omega(n \log n)$

Average case lower bound of sorting: $\Omega(n \log n)$

(1)Quicksort is optimal in the average case.
(O(n log n) in average)

(2)(i)worst case time complexity of heapsort is O(n log n)

(ii)average case lower bound: $\Omega(n \log n)$

$\Rightarrow$average case time complexity of heapsort is O(n log n)

Heapsort is optimal in the average case.

● Improving a lower bound through oracles

problem P: merge two sorted sequences A and B with lengths m and n.

(1)**<u>Binary decision tree</u>:**

There are $\binom{m+n}{n}$ ways

$\binom{m+n}{n}$ leaf nodes in the binary tree.

The lower bound for merging:

$$\lceil \log \binom{m+n}{n} \rceil \leq m + n - 1$$

(conventional merging)

- When m = n

$$\log \binom{m+n}{n} = \log \frac{(2m)!}{(m!)^2} = \log((2m)!) - 2\log m!$$

Using Stirling approximation

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

$$\log \binom{m+n}{n} \approx 2m - \frac{1}{2}\log m + O(1)$$

- Optimal algorithm: 2m – 1 comparisons

$$\log \binom{m+n}{n} < 2m - 1$$

(2)**<u>oracle</u>:**

The oracle tries its best to cause the algorithm to work as <u>hard</u> as it might. (to give a very hard data set)

sorted sequences:

A: $a_1 < a_2 < \cdots < a_m$

B: $b_1 < b_2 < \cdots < b_m$

the very hard case:

$a_1 < b_1 < a_2 < b_2 < \cdots < a_m < b_m$

We must compare:

$a_1 : b_1$
$b_1 : a_2$
$a_2 : b_2$
:
$b_{m-1} : a_{m-1}$
$a_m : b_m$

Otherwise, we may get a wrong result for some input data.

e.g. If b_1 and a_2 are not compared, we can not distinguish

$a_1 < b_1 < a_2 < b_2 < \cdots < a_m < b_m$ and

$a_1 < a_2 < b_1 < b_2 < \cdots < a_m < b_m$

Thus, at least $2m-1$ comparisons are required.

The conventional merging algorithm is optimal for $m = n$.

● Finding lower bound by problem transformation.

Problem A reduces to problem B ($A \propto B$)
iff A can be solved by using any algorithm which solves B.

If $A \propto B$, B is more difficult.

instance of A —transformation $T(tr_1)$→ instance of B

T(B) ↓ solver of B

answer of A ←transformation $T(tr_2)$— answer of B

Note: $T(tr_1) + T(tr_2) < T(B)$

$T(A) \le T(tr_1) + T(tr_2) + T(B) \sim O(T(B))$

● The lower bound of the convex hull problem

sorting $\propto$ convex hull

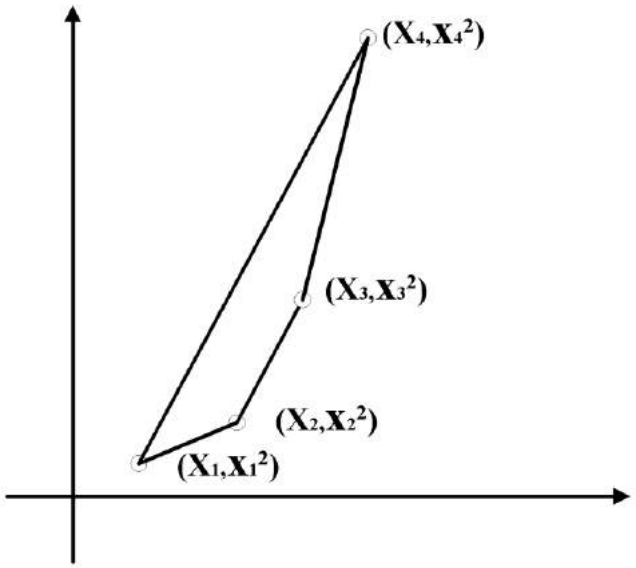
A B

an instance of A: $(x_1, x_2, \cdots, x_n)$

$\downarrow$ transformation

an instance of B: $\{(x_1, x_1^2), (x_2, x_2^2), \cdots, (x_n, x_n^2)\}$

assume: $x_1 < x_2 < \cdots < x_n$



If the convex hull problem can be solved, we can also solve the sorting problem.

The lower bound of sorting: $\Omega(n \log n)$

$\Rightarrow$The lower bound of the convex hull problem:

$\Omega(n \log n)$

● The lower bound of the Euclidean minimal spanning tree (MST) probem

sorting $\propto$ Euclidean MST

A B

an instance of A: $(x_1, x_2, \cdots, x_n)$

$\downarrow$ transformation

an instance of B: $\{(x_1, 0), (x_2, 0), \cdots, (x_n, 0)\}$

$x_1 < x_2 < x_3 < \cdots < x_n$

$\Leftrightarrow$there is an edge between $(x_i, 0)$ and $(x_{i+1}, 0)$ in the MST, where $1 \leq i \leq n-1$

If the Euclidean MST problem can be solved, we can also solve the sorting problem.

The lower bound of sorting: $\Omega(n \log n)$

$\Rightarrow$The lower bound of the Euclidean MST problem: $\Omega(n \log n)$