3 Sorting Algorithms

Sorting

input: sequence $(a(1), \dots, a(n))$ or set $\{a(1), \dots, a(n)\}$

output: sequence $(a(\pi(1)), \dots, a(\pi(n)))$ with $a(\pi(1)) \le \cdots \le a(\pi(n))$

Sorting

input: sequence $(a(1), \dots, a(n))$ or set $\{a(1), \dots, a(n)\}$

output: sequence $(a(\pi(1)), \dots, a(\pi(n)))$ with $a(\pi(1)) \leq \cdots \leq a(\pi(n))$

naive algorithm

1) compute minimum $min\{a(1), \dots, a(n)\}$

Sorting

input: sequence $(a(1), \ldots, a(n))$ or set $\{a(1), \ldots, a(n)\}$

output: sequence $(a(\pi(1)), \ldots, a(\pi(n)))$ with $a(\pi(1)) \leq \cdots \leq a(\pi(n))$

naive algorithm

1) compute minimum $min\{a(1), \ldots, a(n)\} = min\{min\{a(1), a(2)\}, a(3), \ldots, a(n)\}$

$M(n)$ comparisons

$$\begin{aligned} M(1) &= 0 \\ M(n) &= 1 + M(n-1) \end{aligned}$$

Sorting

input: sequence $(a(1), \dots, a(n))$ or set $\{a(1), \dots, a(n)\}$

output: sequence $(a(\pi(1)), \dots, a(\pi(n)))$ with $a(\pi(1)) \leq \cdots \leq a(\pi(n))$

naive algorithm

1) compute minimum

$$min\{a(1), \dots, a(n)\} \quad = min\{min\{a(1), a(2)\}, a(3), \dots, a(n)\}$$

$M(n)$ comparisons

$$\begin{aligned} M(1) &= 0 \\ M(n) &= 1 + M(n-1) \end{aligned}$$

expand

$$\begin{aligned} M(b) &= 1 + M(n-1) \\ &= 1 + 1 + M(n-2) \\ &= 2 + M(n-2) \\ &\dots \\ &= x + M(n-x) \end{aligned}$$

Sorting

input: sequence $(a(1), \dots, a(n))$ or set $\{a(1), \dots, a(n)\}$

output: sequence $(a(\pi(1)), \dots, a(\pi(n)))$ with $a(\pi(1)) \leq \dots \leq a(\pi(n))$

naive algorithm

1) compute minimum

$$min\{a(1), \dots, a(n)\} \; = min\{min\{a(1), a(2)\}, a(3), \dots, a(n)\}$$

$$\begin{aligned} M(1) &= 0 \\ M(n) &= 1 + M(n-1) \end{aligned}$$

$M(n)$ comparisons

expand

$$\begin{aligned} M(b) &= 1 + M(n-1) \\ &= 1 + 1 + M(n-2) \\ &= 2 + M(n-2) \\ &\dots \\ &= x + M(n-x) \end{aligned}$$

stop at

$$n - x = 1 \quad , \quad x = n - 1$$

$$M(n) = n - 1$$

Sorting

input: sequence $(a(1), \dots, a(n))$ or set $\{a(1), \dots, a(n)\}$

output: sequence $(a(\pi(1)), \dots, a(\pi(n)))$ with $a(\pi(1)) \leq \cdots \leq a(\pi(n))$

naive algorithm

1) compute minimum

$$min\{a(1), \dots, a(n)\} = min\{min\{a(1), a(2)\}, a(3), \dots, a(n)\}$$

$M(n) = n - 1$ comparisons

2) repeat

$$sort(A) = min(A) \circ sort(A \setminus min(A)\}$$

Sorting

input: sequence $(a(1), \dots, a(n))$ or set $\{a(1), \dots, a(n)\}$

output: sequence $(a(\pi(1)), \dots, a(\pi(n)))$ with $a(\pi(1)) \leq \cdots \leq a(\pi(n))$

naive algorithm

1) compute minimum

$$min\{a(1), \dots, a(n)\} = min\{min\{a(1), a(2)\}, a(3), \dots, a(n)\}$$

$M(n) = n - 1$ comparisons

2) repeat

$$sort(A) = min(A) \circ sort(A \setminus min(A)\}$$

$S(n)$ comparisons

$$\begin{aligned} S(1) &= 0 \\ S(n) &= M(n) + S(n-1) \\ &= n - 1 + S(n-1) \end{aligned}$$

Sorting

input: sequence $(a(1), \dots, a(n))$ or set $\{a(1), \dots, a(n)\}$

output: sequence $(a(\pi(1)), \dots, a(\pi(n)))$ with $a(\pi(1)) \leq \cdots \leq a(\pi(n))$

naive algorithm

1) compute minimum

$$min\{a(1), \dots, a(n)\} = min\{min\{a(1), a(2)\}, a(3), \dots, a(n)\}$$

$M(n) = n - 1$ comparisons

2) repeat

$$sort(A) = min(A) \circ sort(A \setminus min(A)\}$$

$S(n)$ comparisons

expand

stop at

$$\begin{aligned} S(1) &= 0 \\ S(n) &= M(n) + S(n-1) \\ &= n - 1 + S(n-1) \end{aligned}$$

$$\begin{aligned} S(n) &= M(n) + S(n-1) \\ &= n - 1 + S(n-1) \\ &= n - 1 + n - 2 + S(n-2) \\ &\dots \\ &= \sum_{i=n-x}^{n-1} i + S(n-x) \end{aligned}$$

$n - x = 1$

$$S(n) = \sum_{i=1}^{n-1} i = (n-1) \cdot n/2$$

Merge Sort

merge 2 sorted lists of length n/2

input: sorted sequences

$(a(1), \dots, a(n/2))$

$(b(1)), \dots, b(n/2))$

with

$a(1) \leq \cdots \leq a(n/2)$

$b(1) \leq \cdots \leq b(n/2)$

output: merged sequence $(c(1), \dots, c(n))$ sorted $c(1) \leq \cdots \leq c(n))$

Merge Sort

merge 2 sorted lists of length n/2

input: sorted sequences

$(a(1), \ldots, a(n))$

$(b(1)), \ldots, b(m))$

with

$a(1) \leq \cdots \leq a(n)$

$b(1) \leq \cdots \leq b(nm)$

output: merged sequence $(c(1), \ldots, c(n+m))$ sorted

$c(1) \leq \cdots \leq c(n+m))$

$$merge((a(1),\ldots,a(n)),(b(1),\ldots b(m)))$$
$$= \begin{cases} a(1) \circ merge((a(2),\ldots,a(n)),(b(1),\ldots b(m))) & a(1) \leq b(1) \\ b(1) \circ merge((a(1),\ldots,a(n)),(b(2),\ldots b(m))) & a(1) > b(1) \end{cases}$$

$n+m-1$ comparisons

Merge Sort

merge 2 sorted lists of length n/2

input: sorted sequences

$(a(1), \dots, a(n))$

$(b(1)), \dots, b(m))$

with

$a(1) \leq \cdots \leq a(n)$

$b(1) \leq \cdots \leq b(nm)$

output: merged sequence $(c(1), \dots, c(n+m))$ sorted

$c(1) \leq \cdots \leq c(n+m))$

$$merge((a(1),\dots,a(n)),(b(1),\dots b(m)))$$
$$= \begin{cases} a(1) \circ merge((a(2),\dots,a(n)),(b(1),\dots b(m))) & a(1) \leq b(1) \\ b(1) \circ merge((a(1),\dots,a(n)),(b(2),\dots b(m))) & a(1) > b(1) \end{cases}$$

$n + m - 1$ comparisons

$$sort((a(1), \dots, a(n)) = merge(sort(a(1) \dots a(n/2)), sort(a(n/2+1), \dots, a(n)))$$

Merge Sort

merge 2 sorted lists of length n/2

input: sorted sequences

$(a(1), \dots, a(n))$ with $a(1) \leq \dots \leq a(n)$

$(b(1)), \dots, b(m))$ with $b(1) \leq \dots \leq b(nm)$

output: merged sequence $(c(1), \dots, c(n+m))$ sorted $c(1) \leq \dots \leq c(n+m))$

$$merge((a(1),\dots,a(n)),(b(1),\dots b(m)))$$

$$= \begin{cases} a(1) \circ merge((a(2),\dots,a(n)),(b(1),\dots b(m))) & a(1) \leq b(1) \\ b(1) \circ merge((a(1),\dots,a(n)),(b(2),\dots b(m))) & a(1) > b(1) \end{cases}$$

$n + m - 1$ comparisons

$$sort((a(1), \dots, a(n)) = merge(sort(a(1) \dots a(n/2)), sort(a(n/2+1), \dots, a(n)))$$

$$\begin{aligned} S(1) &= 0 \\ S(n) &< n/2 + n/2 + 2 \cdot S(n/2) \\ &= 2 \cdot S(n/2) + n \end{aligned}$$

difference equations (and difference inequalities)

Let

$$n = 2^k \quad k \in \mathbb{N}$$

be a power of two and assume

$$\boxed{\begin{array}{rcl} f(1) & = & a \\ f(n) & = & 2 \cdot f(n/2) + b \cdot n \end{array}}$$
$$f(n) = ?$$

Idea: expand definition until you see something

$$\begin{array}{rcl} f(n) & = & 2 \cdot f(n/2) + b \cdot n \\ & = & 2 \cdot (2 \cdot f(n/4) + b \cdot n/2) + b \cdot n \\ & = & 2^2 \cdot f(n/2^2) + 2 \cdot b \cdot n \\ & = & \ldots \\ & = & 2^x \cdot f(n/2^x) + x \cdot b \cdot n \end{array}$$

Stop recursion at

$$n/2^x = 1 \quad , \quad x = k = \log n$$

Conjecture

$$\boxed{\begin{array}{rcl} f(n) & = & n \cdot f(1) + b \cdot n \cdot \log(n) \\ & = & b \cdot n \cdot \log n + a \cdot n \end{array}}$$

Merge Sort

merge 2 sorted lists of length n/2

input: sorted sequences

$(a(1), \dots, a(n))$

$(b(1)), \dots, b(m))$

with

$a(1) \leq \dots \leq a(n)$

$b(1) \leq \dots \leq b(nm)$

output: merged sequence $(c(1), \dots, c(n+m))$ sorted

$c(1) \leq \dots \leq c(n+m))$

$$merge((a(1),\dots,a(n)),(b(1),\dots b(m)))$$
$$= \begin{cases} a(1) \circ merge((a(2),\dots,a(n)),(b(1),\dots b(m))) & a(1) \leq b(1) \\ b(1) \circ merge((a(1),\dots,a(n)),(b(2),\dots b(m))) & a(1) > b(1) \end{cases}$$

$n + m - 1$ comparisons

$$sort((a(1), \dots, a(n)) = merge(sort(a(1) \dots a(n/2)), sort(a(n/2+1), \dots, a(n)))$$

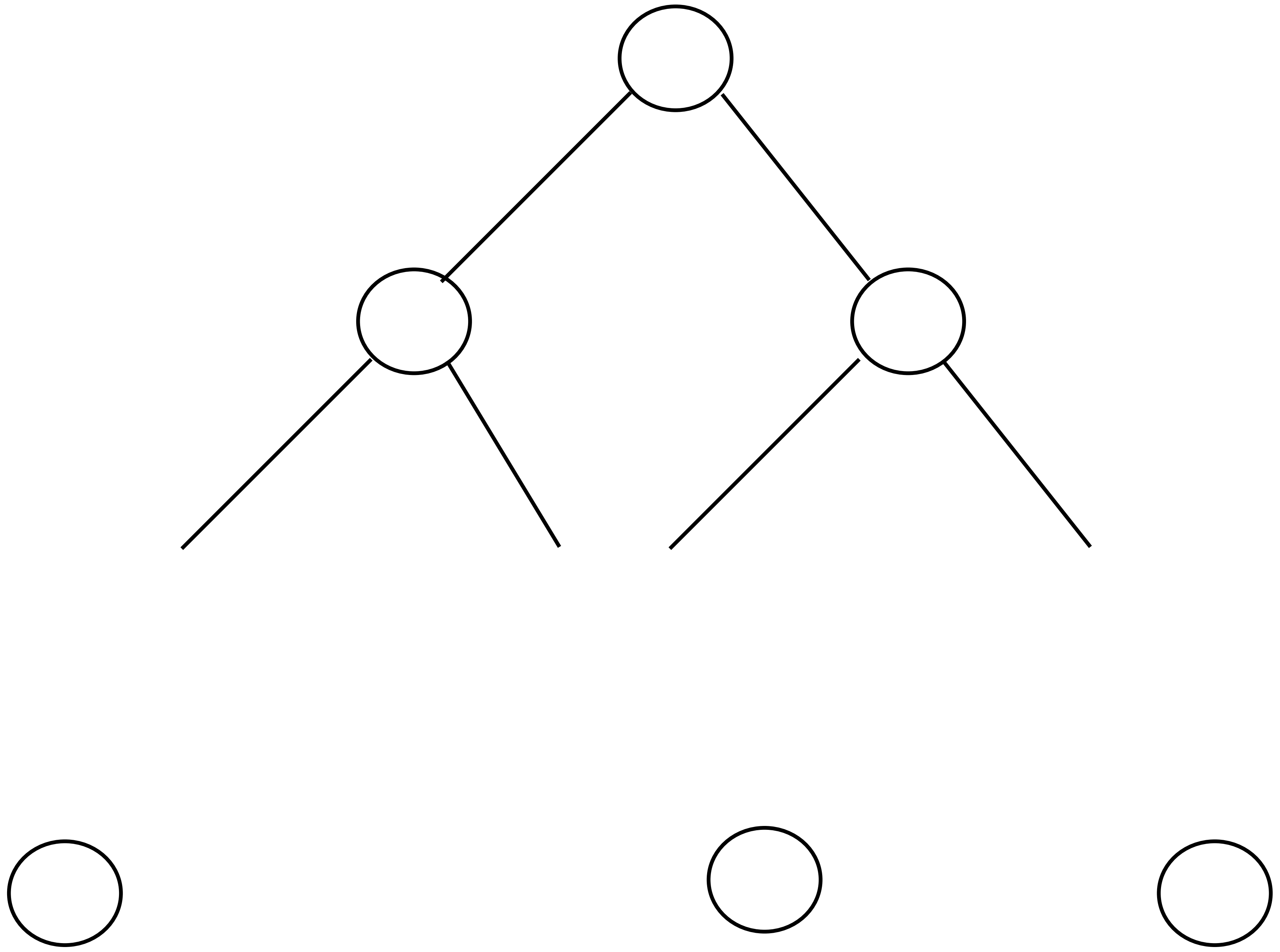
$$\begin{aligned} S(1) &= 0 \\ S(n) &< n/2 + n/2 + 2 \cdot S(n/2) \\ &= 2 \cdot S(n/2) + n \end{aligned}$$

$$\boxed{S(n) < n \cdot \log n}$$

can we sort asymptotically faster with comparisons?

can we sort asymptotically faster with comparisons?

no

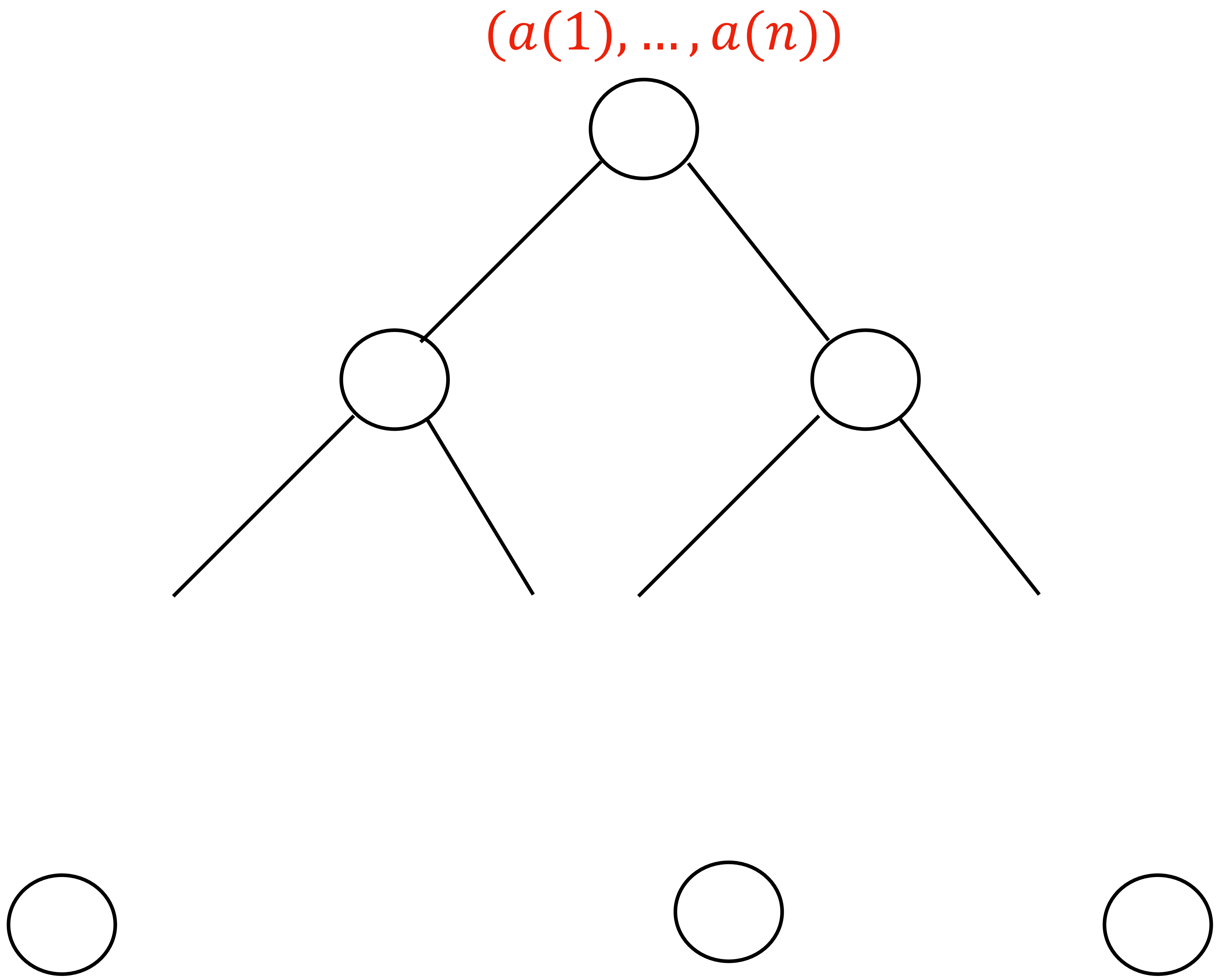


can represent any such algorithm as binary tree

- nodes: comparisons

can we sort asymptotically faster with comparisons?

no



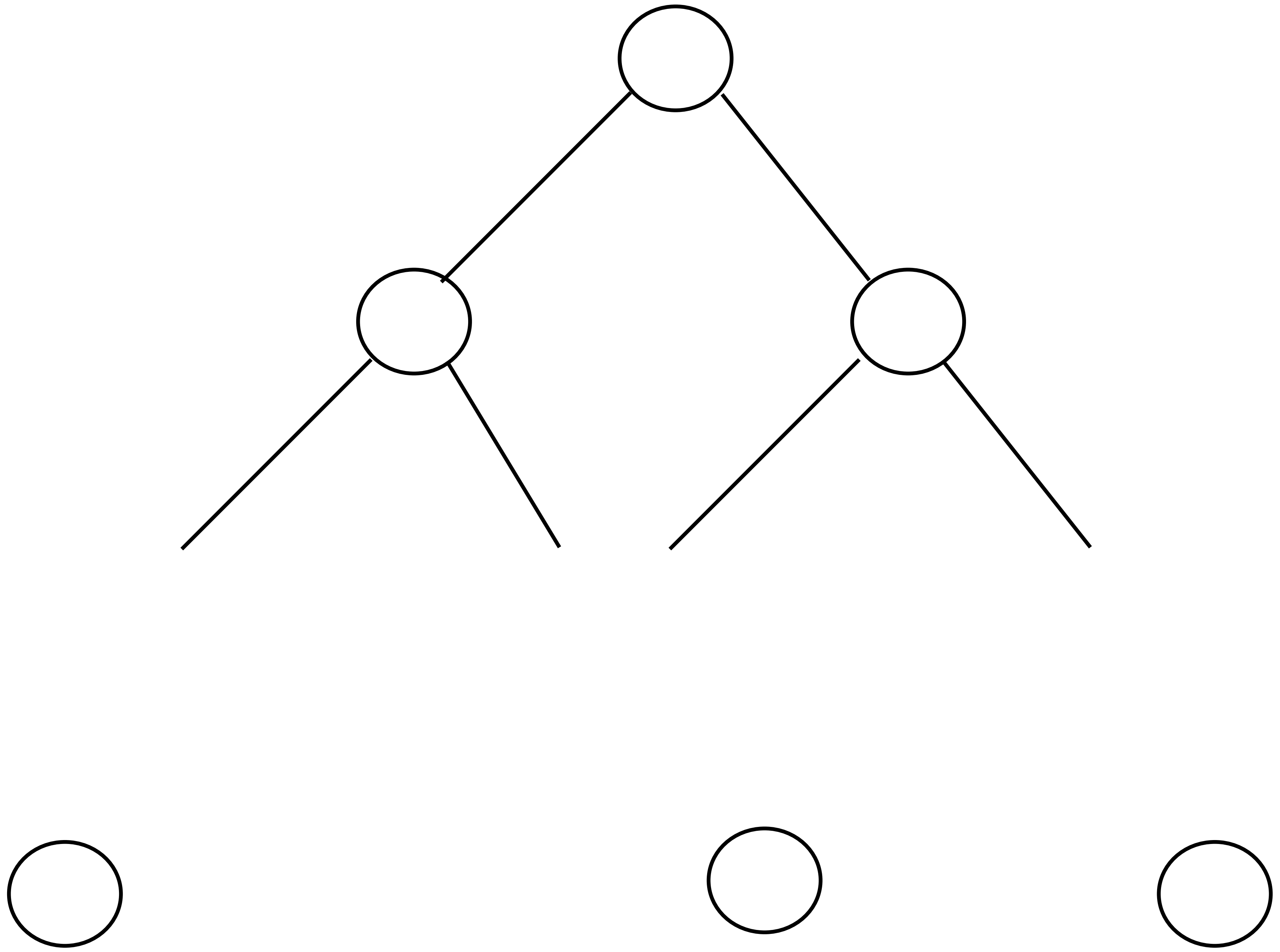
can represent any such algorithm as binary tree

- nodes: comparisons
- at root: input sequence or set to be sorted

can we sort asymptotically faster with comparisons?

no

$(a(1), \ldots, a(n))$



$(a(\pi(1)), \ldots, a(\pi(n)))$

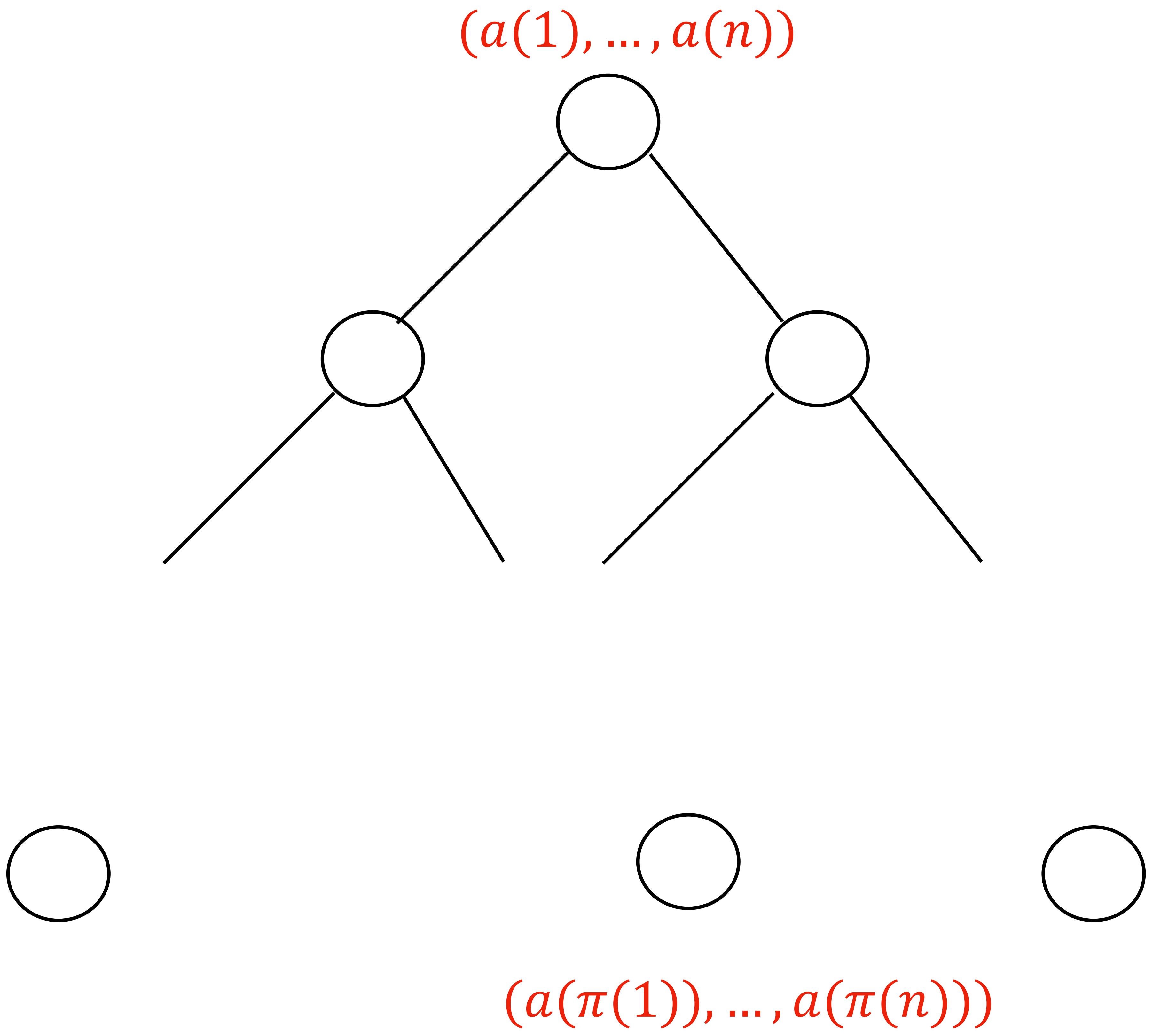
can represent any such algorithm as binary tree

- nodes: comparisons
- at root: input sequence or set to be sorted
- at leaves: sorted sequence
- one leaf for each permutation π

depth of tree = length of longest path = worst case number of comparisons

can we sort asymptotically faster with comparisons?

no



can represent any such algorithm as binary tree
- nodes: comparisons
- at root: input sequence or set to be sorted
- at leaves: sorted sequence
- one leaf for each permutation

π

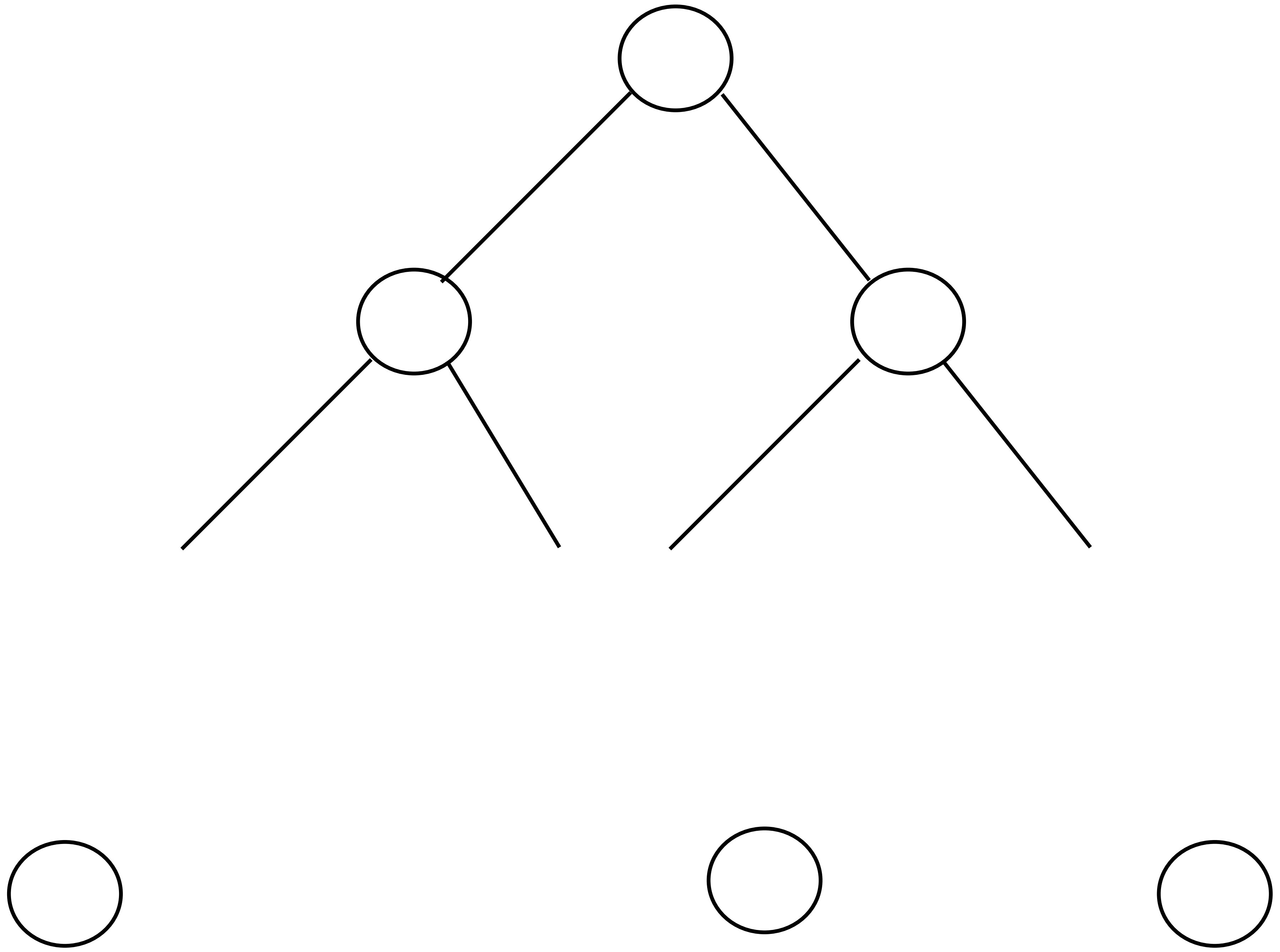
d

depth of tree = length of longest path = worst case number of comparisons

easy induction: binary tree with depth *d* has at most 2^d leaves

can we sort asymptotically faster with comparisons?

no



can represent any such algorithm as binary tree

- nodes: comparisons
- at root: input sequence or set to be sorted
- at leaves: sorted sequence
- one leaf for each permutation π

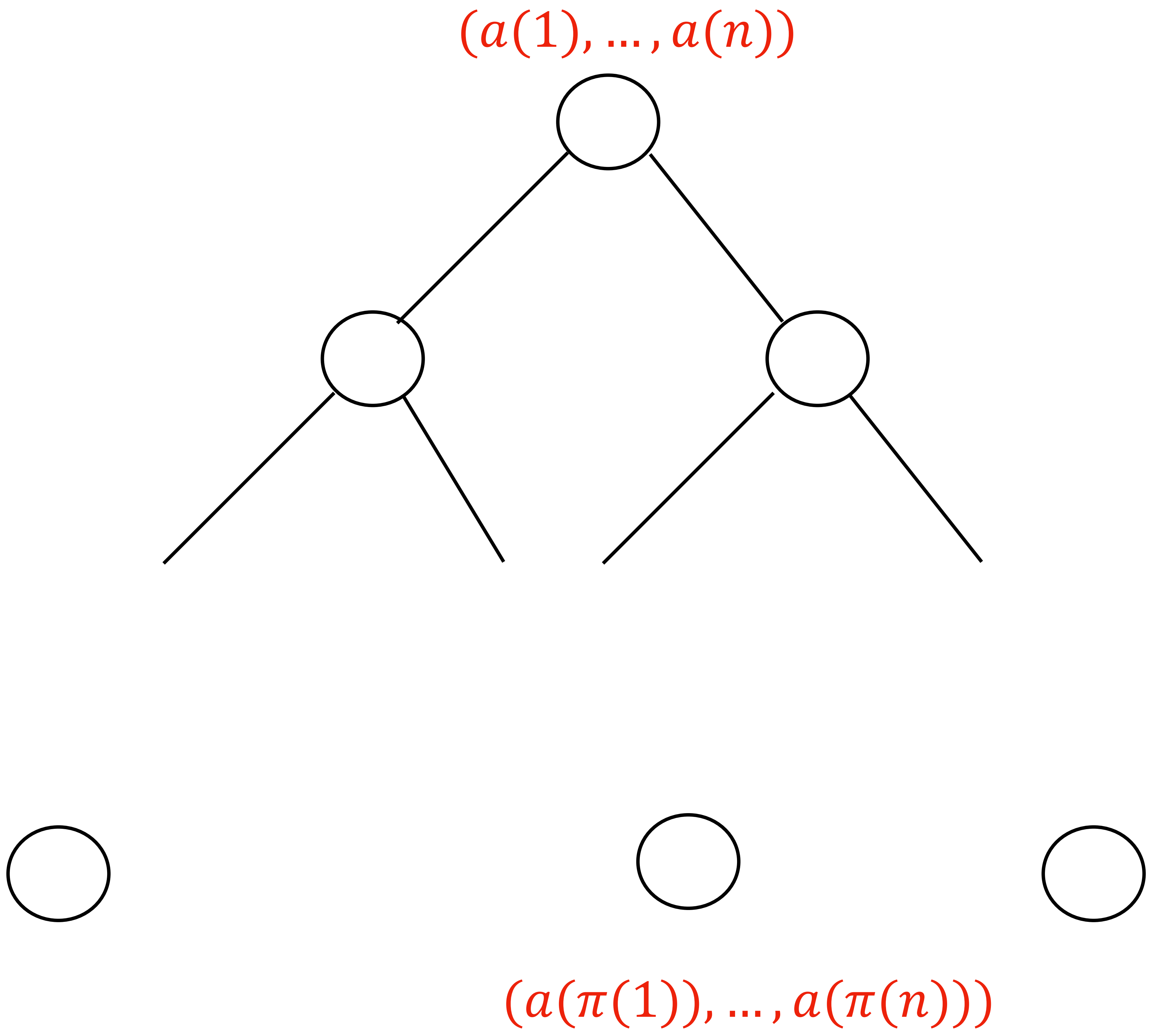
d depth of tree = length of longest path = worst case number of comparisons

easy induction: binary tree with depth *d* has at most 2^d leaves

def: $n!$ number of permutations of n numbers

can we sort asymptotically faster with comparisons?

no



can represent any such algorithm as binary tree

- nodes: comparisons
- at root: input sequence or set to be sorted
- at leaves: sorted sequence
- one leaf for each permutation π

d depth of tree = length of longest path = worst case number of comparisons

easy induction: binary tree with depth *d* has at most 2^d leaves

def: $n!$ number of permutations of n numbers

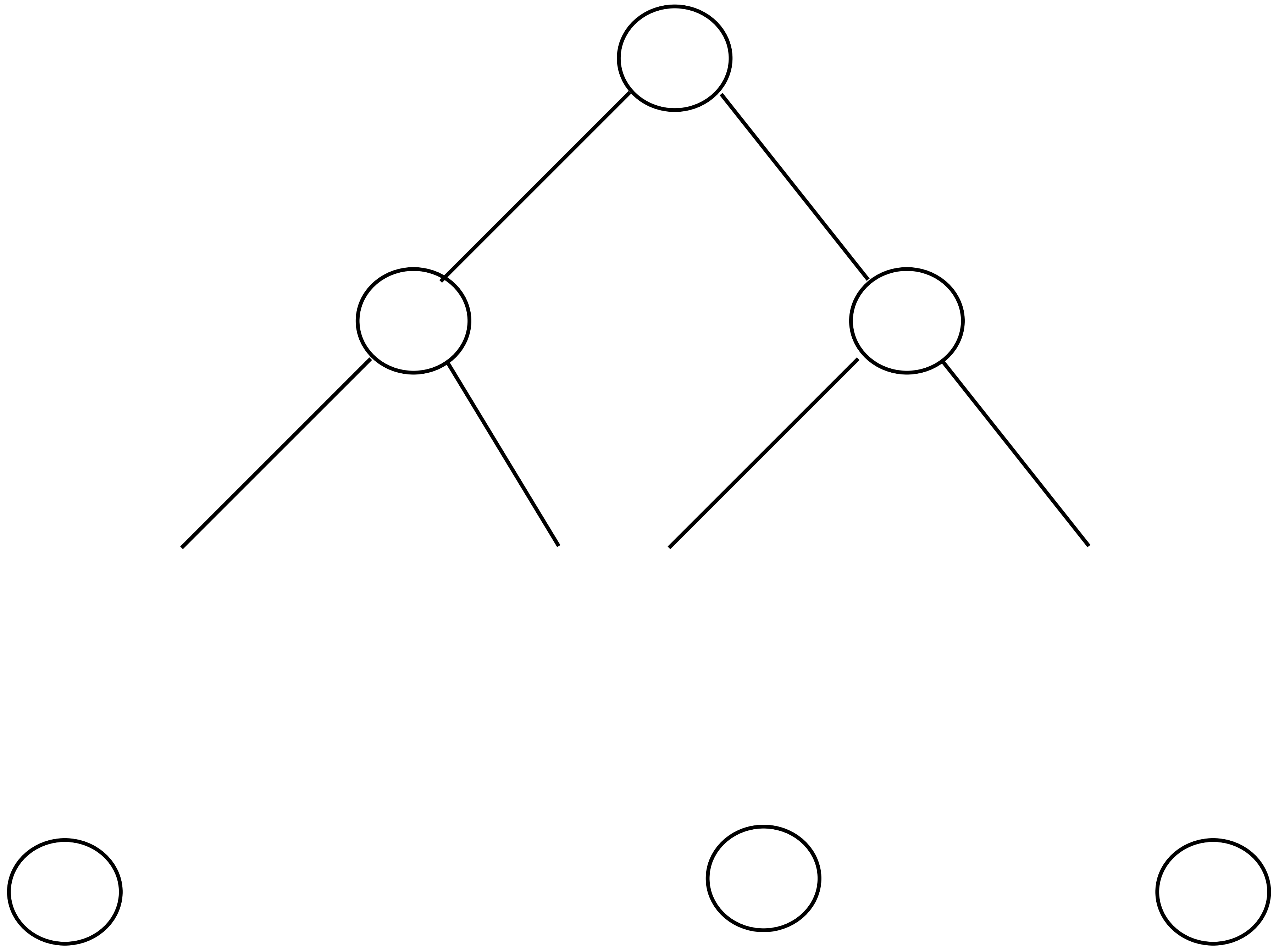
$1! = 1$ $\quad$ $n! = n \cdot (n-1)!$

choices of first element $\quad$ possible permutations of remaining elements

can we sort asymptotically faster with comparisons?

no

$(a(1), \dots, a(n))$



$(a(\pi(1)), \dots, a(\pi(n)))$

can represent any such algorithm as binary tree

- nodes: comparisons
- at root: input sequence or set to be sorted
- at leaves: sorted sequence
- one leaf for each permutation

π

d depth of tree = length of longest path = worst case number of comparisons

easy induction: binary tree with depth *d* has at most 2^d leaves

def: $n!$ number of permutations of n numbers

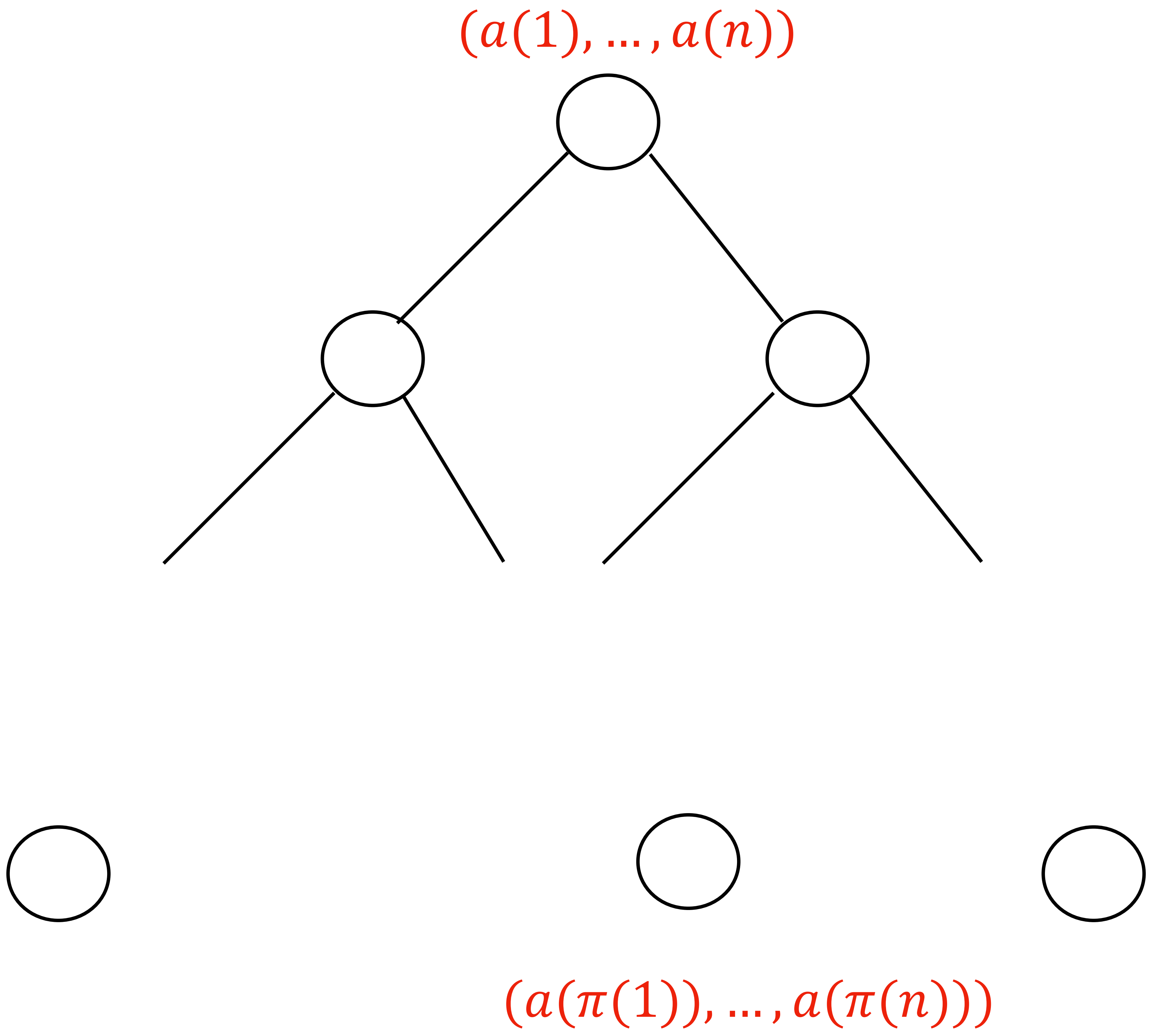
$1! = 1 \qquad n! = n \cdot (n-1)!$

choices of first element possible permutations of remaining elements

$$2^d \geq n! > n \cdot (n-1) \cdot \dots \cdot n/2 > (n/2)^{n/2}$$

can we sort asymptotically faster with comparisons?

no



can represent any such algorithm as binary tree

- nodes: comparisons
- at root: input sequence or set to be sorted
- at leaves: sorted sequence
- one leaf for each permutation π

d depth of tree = length of longest path = worst case number of comparisons

easy induction: binary tree with depth *d* has at most 2^d leaves

def: $n!$ number of permutations of n numbers

$1! = 1$ $\quad$ $n! = n \cdot (n-1)!$

choices of first element $\quad$ possible permutations of remaining elements

$$2^d \geq n! > n \cdot (n-1) \cdot \ldots \cdot n/2 > (n/2)^{n/2}$$

$$\boxed{d > (n/2) \cdot \log(n/2)}$$

Probabilistic Analysis of Algorithms

based on ‚random events‘
whatever randomness is…

more in ‚theoretical computer science‘
or - time permitting - here

Probabilistic Analysis of Algorithms

based on ‚random events‘
whatever randomness is…

more in ‚theoretical computer science‘
or - time permitting - here

two flavors

- random input
  - modeling: our lack of understanding of what future inputs we will see
  - bad: what if an adversary constructs the input - when it matters?

- the algorithm generates - and branches on - random data
  - good: this is under the programmer’s control
  - usual implementation: by a pseudo random number generator

Probabilistic Analysis of Algorithms

based on ‚random events‘
whatever randomness is…

more in ‚theoretical computer science‘
or - time permitting - here

two flavors

- random input
    - modeling: our lack of understanding of what future inputs we will see
    - bad: what if an adversary constructs the input - when it matters?

- the algorithm generates - and branches on - random data
    - good: this is under the programmer's control
    - usual implementation: by a pseudo random number generator
        - generated data are completely predictable (run the generator)
        - whatever randomness is… this is not it!

Probabilistic Analysis of Algorithms

based on ‚random events'
whatever randomness is…

more in ‚theoretical computer science'
or - time permitting - here

two flavors

- random input
  - modeling: our lack of understanding of what future inputs we will see
  - bad: what if an adversary constructs the input - when it matters?

- the algorithm generates - and branches on - random data
  - good: this is under the programmer's control
  - usual implementation: by a pseudo random number generator
    - generated data are completely predictable (run the generator)
    - whatever randomness is… this is not it!

the miracle

- you program with a pseudo random number generator
- you analyze as if the generated data was truly random
- the measured run time (usually) matches the analysis

gentle reminder at this place:

mathematics is an experimental science

repairs just happen to be particularly rare

gentle reminder at this place:

mathematics is an experimental science

repairs just happen to be particularly rare

1903: Russel paradox

gentle reminder at this place:

mathematics is an experimental science

repairs just happen to be particularly rare

1903: Russel paradox

why does probabilistic modeling
of a completely deterministic -provably not random - algorithm
give the right results??

the miracle

- you program with a pseudo random number generator
- you analyze as if the generated data was truly random
- the measured run time (usually) matches the analysis

Quicksort

here: the algorithm does the random experiments

input: $(a(1), \ldots, a(n))$ or set $A = \{a(1), \ldots, a(n)\}$

Quicksort

here: the algorithm does the random experiments

input: $(a(1), \dots, a(n))$ or set $A = \{a(1), \dots, a(n)\}$

random experiment: choose ‚splitter‘ $s \in \{a(1), \dots, a(n)\}$

Quicksort

here: the algorithm does the random experiments

input: $(a(1), \ldots, a(n))$ or set $A = \{a(1), \ldots, a(n)\}$

random experiment: choose ‚splitter' $s \in \{a(1), \ldots, a(n)\}$

all *n* splitters equally likely

Quicksort

here: the algorithm does the random experiments

input: $(a(1), \ldots, a(n))$ or set $A = \{a(1), \ldots, a(n)\}$

random experiment: choose ‚splitter‘ $s \in \{a(1), \ldots, a(n)\}$

all n splitters equally likely

$$A_< = \{a \in A | a < s\}$$

$$A_> = \{a \in A | a > s\}$$

Quicksort

here: the algorithm does the random experiments

input: $(a(1), \dots, a(n))$ or set $A = \{a(1), \dots, a(n)\}$

we assume here: a(i) mutually distinct

random experiment: choose ‚splitter' $s \in \{a(1), \dots, a(n)\}$

all *n* splitters equally likely

$$A_< = \{a \in A | a < s\}$$

$$A_> = \{a \in A | a > s\}$$

$$sort(A) = sort(A_<) \circ s \circ sort(A_>)$$

Quicksort. 1960, Tony Hoare

here: the algorithm does the random experiments

input: $(a(1), \dots, a(n))$ or set $A = \{a(1), \dots, a(n)\}$

we assume here: a(i) mutually distinct

random experiment: choose ‚splitter‘ $s \in \{a(1), \dots, a(n)\}$

all n splitters equally likely

$$A_< = \{a \in A | a < s\}$$

$$A_> = \{a \in A | a > s\}$$

$$sort(A) = sort(A_<) \circ s \circ sort(A_>)$$

2013

now computer science textbooks continue ….. since decades

here: the algorithm does the random experiments

input: $(a(1), \dots, a(n))$ or set $A = \{a(1), \dots, a(n)\}$ we assume here: a(i) mutually distinct

random experiment: choose ‚splitter‘ $s \in \{a(1), \dots, a(n)\}$

all *n* splitters equally likely

$A_< = \{a \in A | a < s\}$

$A_> = \{a \in A | a > s\}$

n-1 comparisons

$sort(A) = sort(A_<) \circ s \circ sort(A_>)$

$T(n)$ expected run time for sorting n elements

$$\boxed{T(n) \leq n + (1/n) \cdot \Sigma_{i=1}^{n} (T(i-1) + T(n-i))}$$

now computer science textbooks continue ….. since decades

here: the algorithm does the random experiments

input: $(a(1), \dots, a(n))$ or set $A = \{a(1), \dots, a(n)\}$

we assume here: a(i) mutually distinct

random experiment: choose ‚splitter‘ $s \in \{a(1), \dots, a(n)\}$

all *n* splitters equally likely

$$A_< = \{a \in A | a < s\}$$

$$A_> = \{a \in A | a > s\}$$

n-1 comparisons

$$sort(A) = sort(A_<) \circ s \circ sort(A_>)$$

$T(n)$ expected run time for sorting n elements

in mathematics expected runtime has a definition

$$\boxed{T(n) \leq n + (1/n) \cdot \Sigma_{i=1}^{n} (T(i-1) + T(n-i))}$$

now computer science textbooks continue ….. since decades

here: the algorithm does the random experiments

input: $(a(1), \dots, a(n))$ or set $A = \{a(1), \dots, a(n)\}$ we assume here: a(i) mutually distinct

random experiment: choose ‚splitter' $s \in \{a(1), \dots, a(n)\}$

all *n* splitters equally likely

$$A_< = \{a \in A | a < s\}$$

$$A_> = \{a \in A | a > s\}$$

n-1 comparisons

$$sort(A) = sort(A_<) \circ s \circ sort(A_>)$$

$T(n)$ expected run time for sorting n elements

in mathematics expected runtime has a definition

$$\boxed{T(n) \leq n + (1/n) \cdot \Sigma_{i=1}^{n} (T(i-1) + T(n-i))}$$

this should have a proof!