CS27000: PROGRAMMING AND DATA STRUCTURES.

WEEK 13

SORTING ALGORITHMS (continued)

ANALYSIS OF QUICKSORT.

RECALL QUICK SORT :

(1) Select a pivot

(2) Partition using the pivot

(3) Recursively sort left and right parts.

Recurrence relation :

$$T(n) = T(i) + T(n-i-1) + cn$$

$i =$ size of one of the parts.

We have seen best case and worst case analysis.

AVERAGE CASE ANALYSIS OF QUICKSORT.

$$T(n) = T(i) + T(n-i-1) + cn$$

What are the diffrent values that i can take?

$i = 0, 1, 2 \ldots$

each of them is equally likely

AVERAGE CASE ANALYSIS OF QUICKSORT.

$$T(n) = T(i) + T(n-i-1) + cn$$

$$T(n) = \frac{1}{n} \sum_{i=0}^{n-1} \left[T(i) + T(n-i-1) \right] + cn$$

$$= \frac{2}{n} \left[T(0) + T(1) + T(2) + \dots + T(n-1) \right] + cn$$

AVERAGE CASE ANALYSIS OF QUICKSORT.

$$T(n) = T(i) + T(n-i-1) + cn$$

$$T(n) = \frac{1}{n} \sum_{i=0}^{n-1} \left[T(i) + T(n-i-1) \right] + cn$$

$$= \frac{2}{n} \left[T(0) + T(1) + T(2) + \dots + T(n-1) \right] + cn$$

$$\left.nT(n) = 2\left[T(0) + T(1) + \dots + T(n-1)\right] + cn^2 \right\} A$$

$$\left.(n-1)\,T(n-1) = 2\left[T(0) + T(1) + \dots + T(n-2)\right] + c(n-1)^2 \right\} B$$

AVERAGE CASE ANALYSIS OF QUICKSORT.

$$nT(n) - (n-1)\,T(n-1) = 2T(n-1) + 2cn - c$$
$$= 2T(n-1) + 2c'n$$

$$nT(n) = (n+1)\,T(n-1) + 2c'n$$

$$\frac{T(n)}{(n+1)} = \frac{T(n-1)}{n} + \frac{2c'}{(n+1)}$$

Telescoping $\frac{T(n)}{n+1} - \frac{T(n-1)}{n}$

AVERAGE CASE ANALYSIS OF QUICKSORT.

$$\frac{T(n)}{(n+1)} - \frac{T(n-1)}{n} = \frac{2c'}{(n+1)}$$

$$\frac{T(n-1)}{n} - \frac{T(n-2)}{n-1} = \frac{2c'}{n}$$

$$\vdots$$

$$\frac{T(n)}{(n+1)} - \frac{T(0)}{1} = \frac{2c'}{(n+1)} + \frac{2c'}{n} + \cdots + \frac{2c'}{2} =$$

$$= 2c'\left[\frac{1}{n+1} + \frac{1}{n} + \cdots \frac{1}{1}\right] \approx \underbrace{H_{n+1}}_{n^{th}\ \text{Harmonic}\ \#} \cdot 2c'$$

QUICK SORT : SUMMARY.

- A recursive sorting algorithm
- Running time depends on selection of pivot.
- Worst case $O(n^2)$

  Average case $O(n \log n)$

Has fast implementations in practice.

what is special about $O(n \log n)$?
w.r.t sorting algos.

Can we do better?
how do we know we have reached the limit?

What is the limit?
– lower bounds allow to address this

A general lower bound on sorting

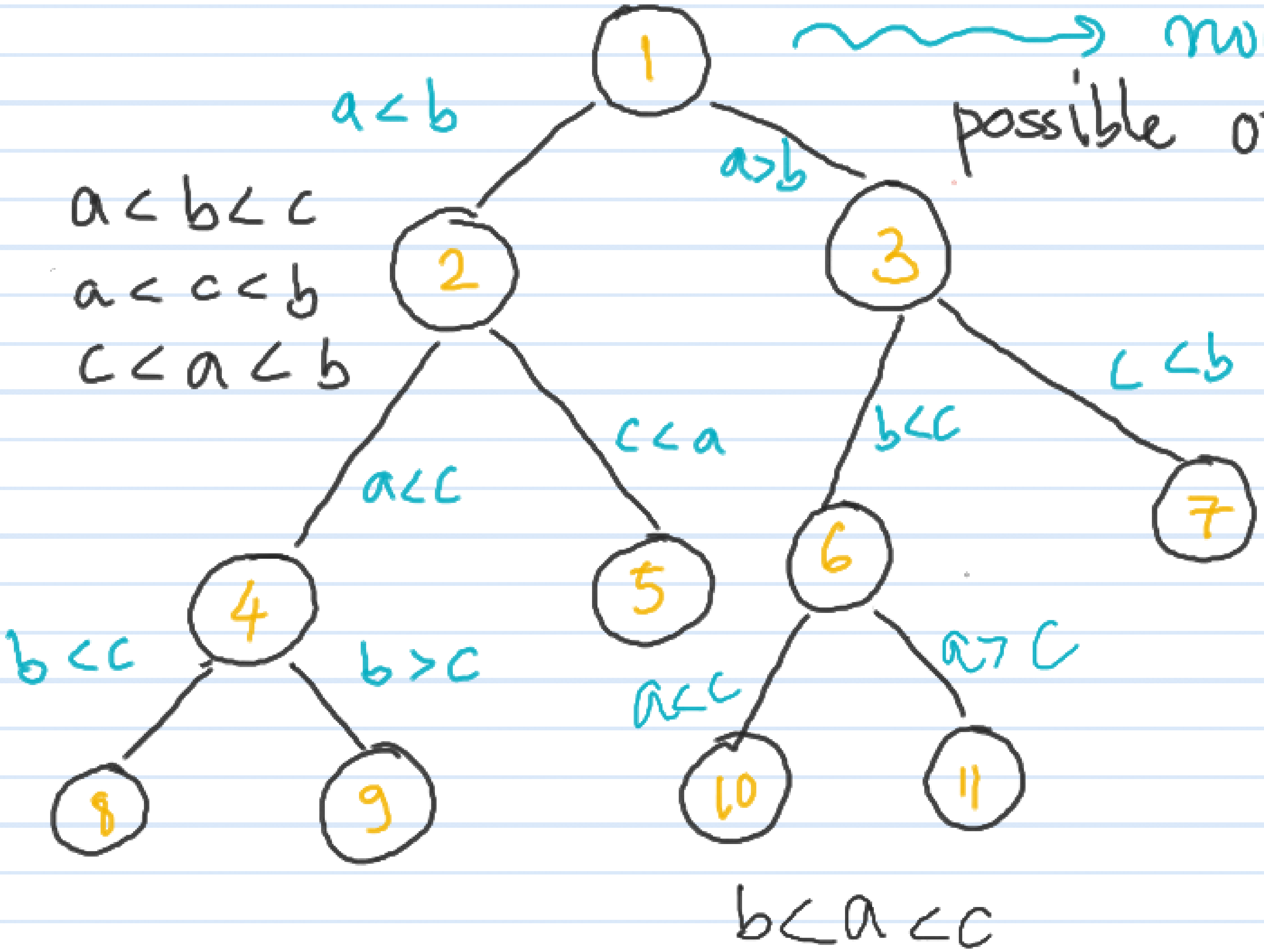
Claim : Any sorting algo that uses comparisons only requires $\lceil \log N! \rceil$ comparisons in the worst case

- Such algos are called comparison based sorting algos

selection sort, mergesort, quicksort, bubble sort heap sort all fall under this category

Decision tree : An abstraction for proving lower bounds.

(Binary)

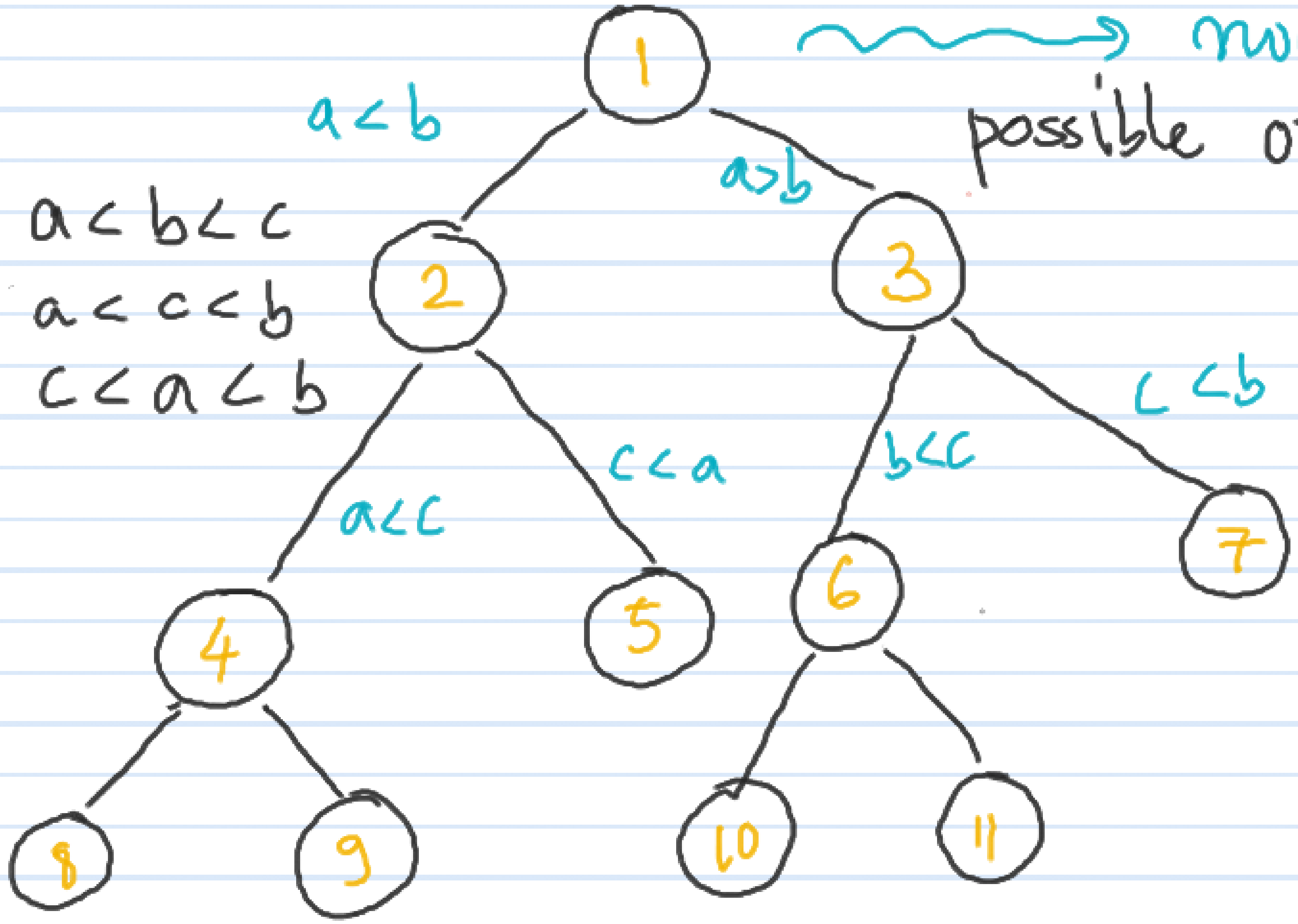


node in a decision tree

possible orderings of elements at that node

(Binary)

Decision tree : An abstraction for proving lower bounds.



node in a decision tree

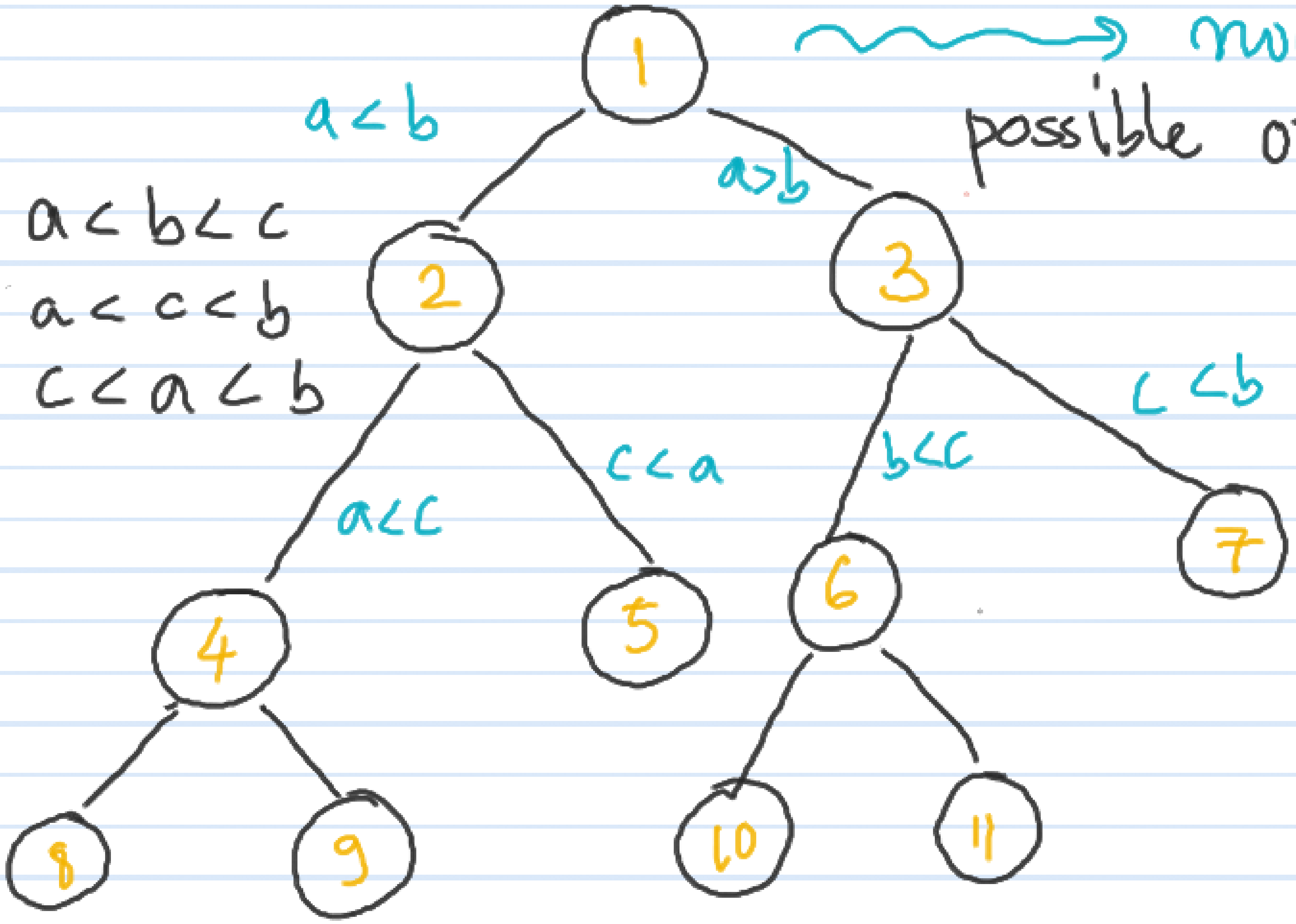
possible orderings of elements at that node

At root all orderings are possible

Once some comparisons are done some orderings are eliminated

Decision tree : An abstraction for proving lower bounds.

(Binary)



node in a decision tree

possible orderings of elements at that node

Execution of an algo

≈ decision tree

. How many comparisons does the algo need?

Proving lower bound using DT.

Claim 1: Let T be a binary tree of depth d.

\# of leaves in T is $\leq \underline{2^d}$

Claim 2: A binary tree with L leaves has depth at least $\underline{\lceil \log L \rceil}$

Claim 3: A decision tree to sort N elements has $N!$ leaves.

How does freq based algo (counting sort) sort in linear time?

Let B denote the freq array of size M

- The sorting algo uses the following operation
  B[A[i]]++;
- This operation is more powerful since it performs an M-way comparision in unit time
- does not contradict the $\Omega(n \log n)$ lower bound.

of Inversions in input

34, 8, 64, 51, 32, 21 } # of inversions = 9

8 34 64 51 32 21 } # of inversions = 8

What is an inversion ?

– a pair (i, j) s.t.

$i \leq j$ and $A[i] \geq A[j]$

How many inversions are present in above example?

Sorted Array : # of inversions !

of Inversions in input

34, 8, 64, 51, 32, 21 } # of inversions = 9

8 34 64 51 32 21 } # of inversions = 8

Inversions : (34, 8) (34,32) (34,21) (64,51) (64,32) (64,21)
(51, 32) (51,21) (32,21)

In blue array : all above inversions except (34,8) remain as it is.

(obtained by swapping adjacent elements

21, 8, 64, 51, 32, 34

Inversions : <u>(21, 8)</u> (64,51) (64,32) <u>(64,34)</u>
(51, 32) <u>(51,34)</u>

} obtained by swapping non adjacent elements

blue underlined ones are new inversions.

Average # of Inversions in input

34, 8, 64, 51, 32, 21

How many permutations? $P_1, P_2, P_3 \; .. \; P_k$

$$\left.\begin{array}{l}\text{average}\\ \text{\# of inversions}\\ \text{is}\end{array}\right\} = \frac{1}{k}\left[I(P_1) + I(P_2) + \cdots + I(P_k) \right]$$

How to determine $I(P_j)$?

Average # of Inversions in input

1	2	3	4	5	6	
34,	8,	64,	51,	32,	21	P_j
21,	32,	51,	64,	8,	34	P_j'

For any 2 elements say (34, 51)

They are inverted in either P_j or P_j'

Between P_j and P_j' # of inversions is ______

Average # of Inversions in input

34, 8, 64, 51, 32, 21

How many permutations? $P_1, P_2, P_3 \dots P_k$

find its pair

average # of inversions is $\Big\} = \frac{1}{\cancel{k}\ n!} [I(P_1) + I(P_2) + \dots + I(P_k)]$

$$= \frac{1}{n!}\left[\frac{n(n-1)}{2} \times \frac{n!}{2}\right]$$

$$= n(n-1)/4.$$