TRANSFORM & CONQUER

problem ⟶ simpler instance / alternate representation / different problem ⟶ solution

Red-Black Trees

[log n]

Root, leaves (NULL) ⟶ black

every red node ⟶ two children

route from root to NIL always has same no of black nodes (black depth)

shortest path ⟶ all black

longest path ⟶ alternate

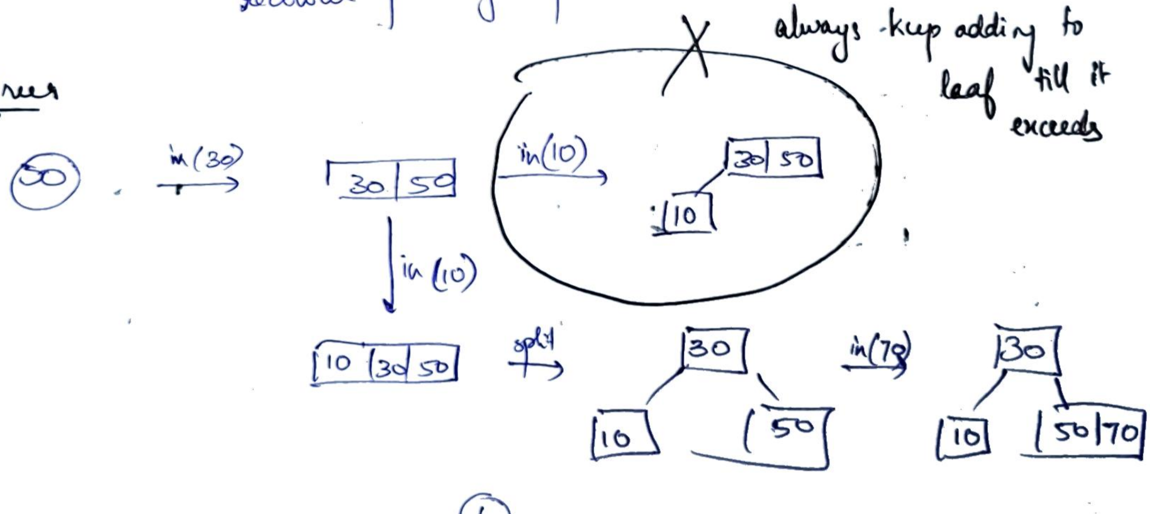
$\boxed{\text{longest} \le 2(\text{shortest})}$ $\boxed{h \le 2\log_2(n+1)}$

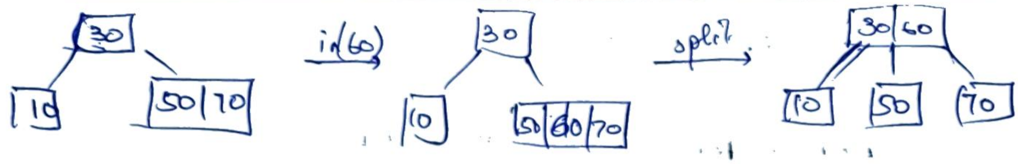
Insertion

1. insert and colour z red
2. if z is root: recolour | if z.uncle = red: recolour unc, parent, grandparent | if z.uncle = black (triangle): rotate z.parent

if z.uncle = black (line):
rotate z.grandparent
recolour parent, grandparent

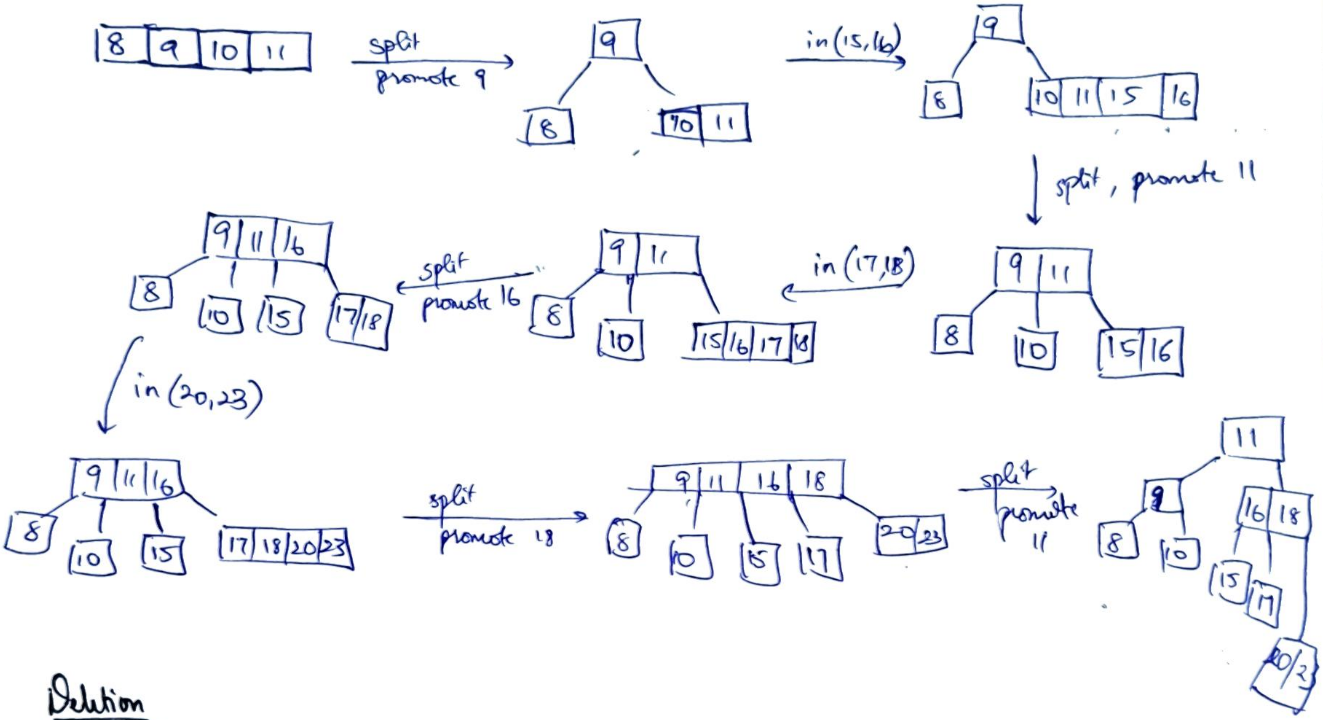
2-3 Trees



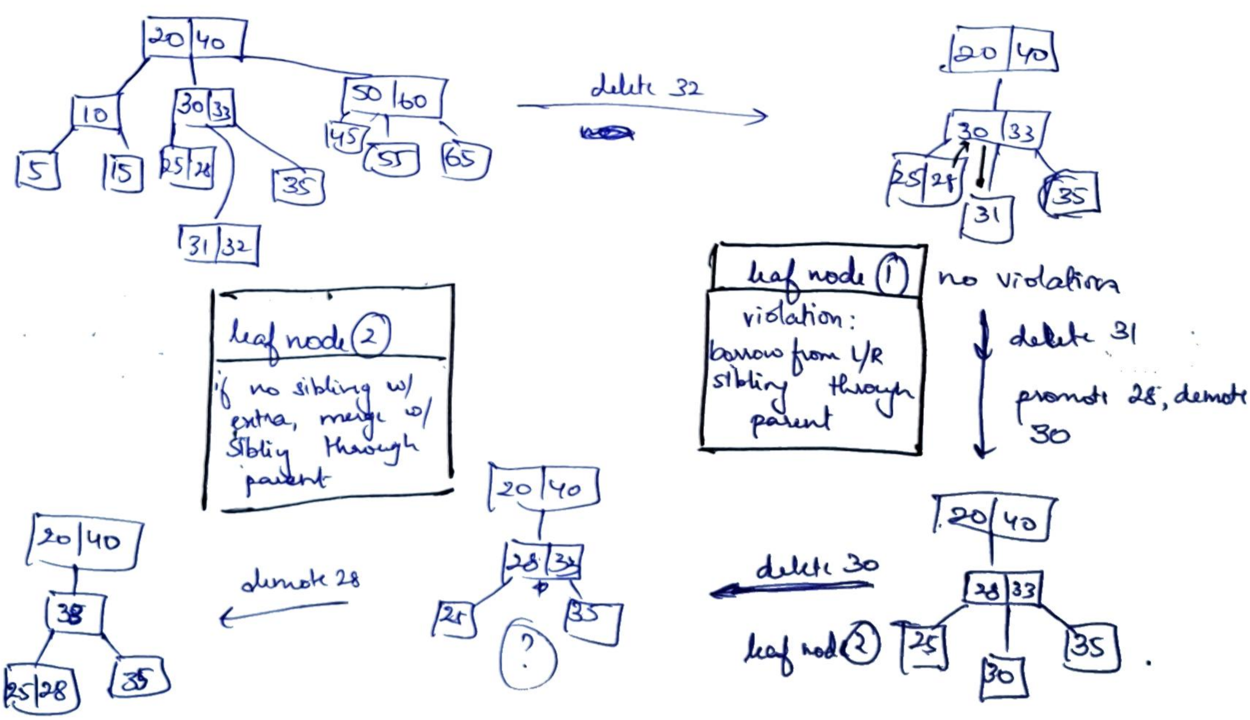


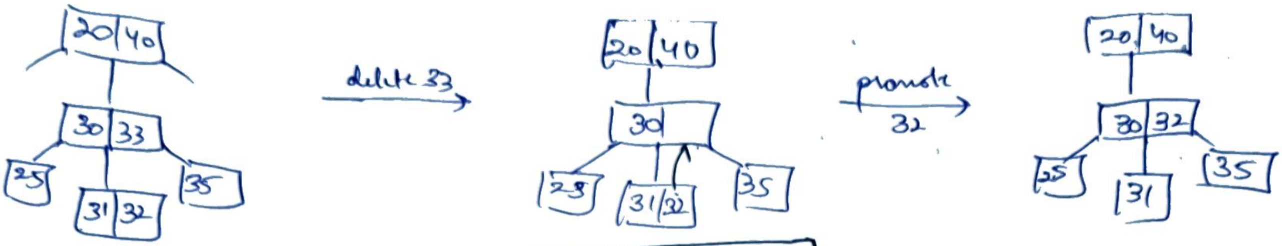
B-TREE

8, 9, 10, 11, 15, 16, 17, 18, 20, 23. Order = 4



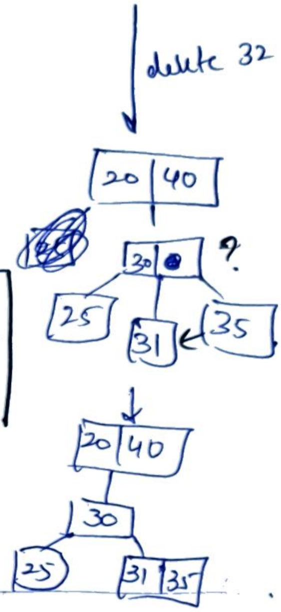
Deletion





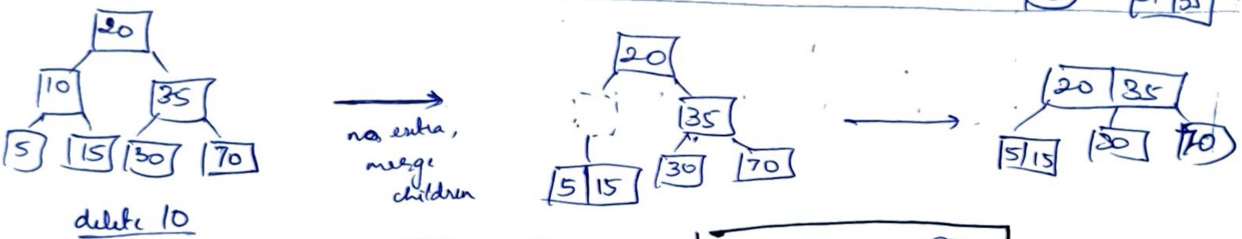
internal node 1

if children extra, replace w/ inorder pred/successor



internal node 2

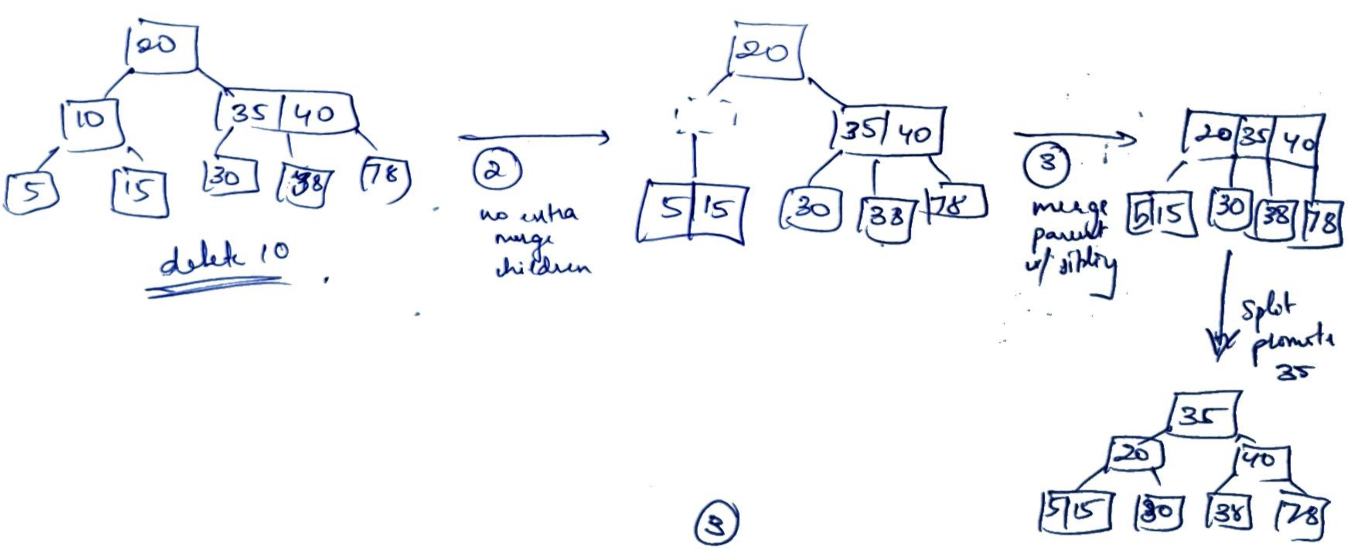
no extra, merge children



~~internal node 3 look for extra in sibling to replace~~

internal node 3

no extra in deleted. merge parent w/ sibling



2-3 TREES

Every node
- 1 value
  - leaf
  - 2 children
- 2 values
  - leaf
  - 3 children

(log n)

All leaf nodes at same level
like BST for search.

Insertion: find leaf to insert → if more than 2, split and promote middle.

B-TREE / M-way height balanced tree

General version of 2-3 tree.

2-3 tree → B tree of order 3.

for order n,

Any node can have max n-1 values and n children

Each node except root $\xrightarrow{\text{at least}}$ n/2 children
$\xrightarrow{\text{at most}}$ n children

Root will have at least 1 value, 2 children.

P.T.O

All leaves at same level, i.e., same height/depth

SPACE - TIME TRADEOFFS

Input Enhancement: additional space to preprocess input to reduce overall time.

- Counting sort
  - Comparison
  - Distribution
- String matching
  - Horspool's
  - Boyer-Moore

Comparison Count Sort

for every element, count no. of elements smaller than it.
→ gives position

go through arr. → +1 the ith element for every element smaller

+1 the loop element if bigger than ith

Distribution Count Sort

Say you know the elements of a list ∈ finite domain

- get frequency of each element
- Construct distribution array with cumulative freq for array elements in order.
- go from n-1 to zero → A[i] - l becomes index in D

↓

D[j]-1 (zero based index) becomes index for element in sorted array

Subtract 1 from D stored freq. ←

$\boxed{O(n)}$

Input Enhancement

Comparison Count Sort

Count ele smaller
↓
ascending

" " greater
↓
descending

A	62	31	84	96	19	47	
Count (initial)	0	0	0	0	0	0	
$i=0$	3	0	1	1	0	0	
$i=1$	3	1	2	2	0	1	
$i=2$	3	1	4	3	0	1	
$i=3$	3	1	4	5	0	1	
$i=4$	3	1	4	5	0	2	} Final state.

Sorted array: | 19 | 31 | 47 | 62 | 84 | 96 |

$$\boxed{\begin{matrix} n^2 \\ \left[\frac{n(n-1)}{2}\right] \end{matrix}}$$

Distribution Count Sort

you know values come from finite domain, you know order of finite domain.

$D[0 \ldots (u-l)]$

$D[j] \leftarrow D[j-1] + D[j]$.

1. freq. array
2. dist. array → cumulative freq.
3. from $n-1$ to 0:
   - $A[i] = x$
   - x - lowerbound = index for dist array d_i
   - $S[D[d_i]-1] = A[i]$.
   - $D[d_i] -= 1$

$$\boxed{O(n)}$$

$\{11, 12, 13\}$

13	11	12	13	12	12

	11	12	13
Freq:	1	3	2
Dist:	1	4	6

	Dist								
	1	4	6						
A[5]=12	1	3	6				12		
A[4]=12	1	2	6			12	12		
A[3]=13	1	2	5			12	12		13
A[2]=12	1	1	5		12	12	12		13
A[1]=11	0	1	5	11	12	12	12		13
A[0]=13	0	1	4	11	12	12	12	13	13

Horspool's Algorithm

Shift table

Shift = val (a) $\begin{cases} \text{length of pattern } m & \text{if } a \notin P \\ \text{distance from rightmost occurrence of 'a' to end} & \text{if } a \in P \end{cases}$

$\boxed{\begin{matrix}\Theta(n) \\ O(mn)\end{matrix}}$

Always makes a shift based on rightmost character c that was compared to P

Suffix matching

$0 \to m-2$: Table $[P[j]] \leftarrow m-1-j$

P: B A R B E R

Shift table

A	B	E	R	...
4	2	1	3	6

S: THIS_IS_A_BARBIE_BARBERSHOP

BARBER

BARBER

BARBER

BARBER

BARBER ✓

BOYER-MOORE STRING MATCHING

Two tables → bad symbol shift → same as Horspool

→ good suffix shift

k from 1 to m-1. take k length suffix and check for ~~another~~ rightmost occurrence that isnt preceded by the same character (preceding suffix)

d_1 = Horspool shift - no of chars matched

d_2 = good suffix shift

shift value = max (d_1, d_2)

only when some characters matched. If no characters match, shift value is only d1

CABABA

$d_2(1) = 4$

WOWWOW

$d_2(3) = 3$

$d_2(2) = 5$

$\therefore d_2(1) = 5$

P: BAOBAB

Bad symbol shift

B	A	O	...
2	1	3	6

good suffix shift

k	suffix	d_2
1	BAOBAB	2
2	BAOBAB	~~2~~ 5
3	BAOBAB	~~2~~ 5
4	BAOBAB	~~2~~ 5
5	BAOBAB	~~2~~ 5

B E S S _ KNEW_ ABOUT_ BAOBABS

B A O B A B

$d_1 = 6, d_2 = 2$

B A O B A B

$d_1 = 6-2 = 4, d_2 = 2$

B A O B A B

$d_1 = 6, d_2 = 2$

B A O B A B

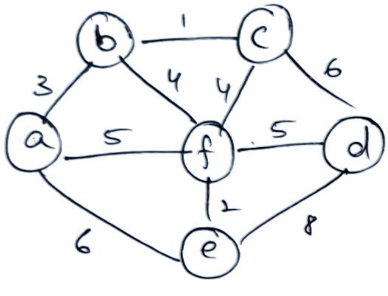
GREEDY ALGOS

On every step:

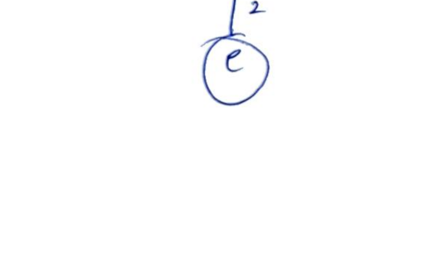
- feasible
- locally optimal
- irrevocable (no backtracking)

Prim's Algo (MST)

Start from one vertex and continue from there.



Vertex	Remaining vertices
a(-,-)	b(3,a), e(6,a), f(5,a), c(∞,-), d(∞,-)
b(3,a)	e(6,a), f(4,b), c(1,b), d(∞,-)
c(1,b)	e(6,a), f(4,b), d(6,c)
f(4,b)	e(2,f), d(5,f)
e(2,f)	d(5,f)
d(5,f)	



$|V|^2$

$|E| \log |V|$

Kruskal's → sort edges by weight

Min edge that does not add a cycle.

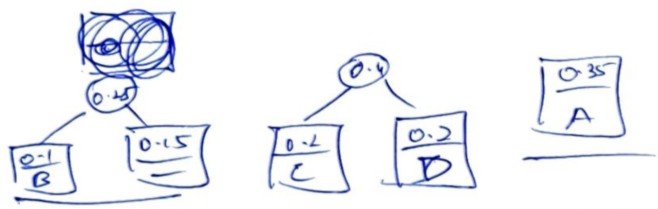
$\Theta(|E| \log |E|)$

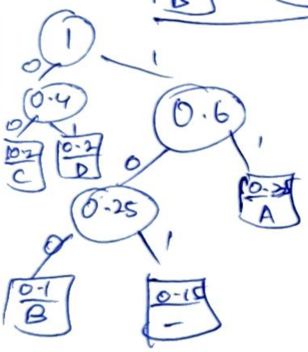
Huffman coding

Variable encoding → prefix-free code.

1. n-1 node tree
2. pick two w/ smallest freq, combine and repeat from ①.

sym	A	B	C	D	—
freq.	0.35	0.1	0.2	0.2	0.15





left → 0
right → 1

Huffman code

Sym	code
A	11
B	100
C	00
D	01
—	101

Avg length $= 2\times0.35 + 3\times0.1 + 2\times0.2 + 2\times0.2 + 0.15\times3$

$= 2.20$.

Compression ratio $= \dfrac{l_{fixed} - l_{var}}{l_{fixed}} \quad \dfrac{3 - 2.2}{3}$

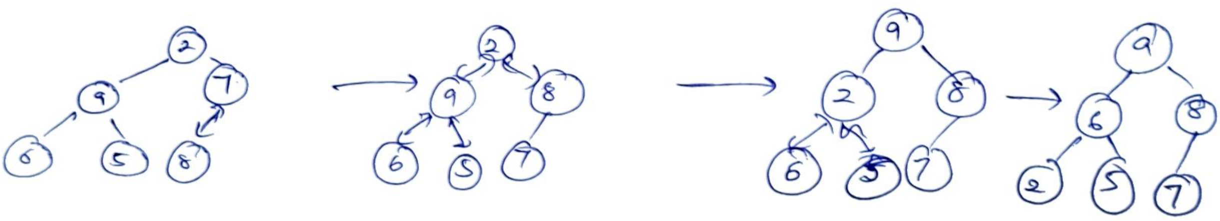
$= 26.6\%$

Heaps

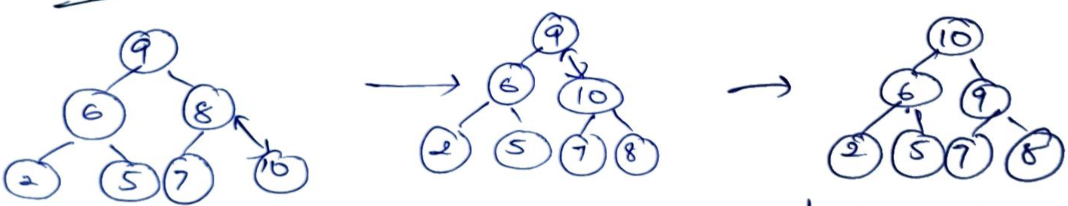
Bottom-up heap:

input: A [0...n-1].

```
for i ← ⌊n/2⌋ to 1:
    p_ind ← i; og_par ← A[i];
    heap ← false
    while not heap and (2*p_ind ≤ n):      children exist
        child_ind ← 2* p_ind.
        if A[child_ind+1] > A[child_ind]
                child_ind ← child_ind+1
        if A[par_ind] > A[child_ind]
                heap = true
        else:
            A[par_ind] = A[child_ind]
            par_ind ← child_ind
    A[par_ind] ← og_par
```



Top-down.



Deletion.

1. Swap node to delete w/ last key
2. Remove key
3. Heapify.

Heap sort.

1. Construct heap
2. Delete max node. n-1 times

Heap

2 9 7 6 5 8

2 9 8 6 5 7

2 9 8 6 5 7

9 2 8 6 5 7

9 6 8 2 5 7

max deletions

9 6 8 2 5 7

7 6 8 2 5 | 9

8 6 7 2 5

~~6 7 2 5 | 8~~

~~7 6 2 5~~

~~6 5 6 2 | 7~~

5 6 7 2 | 8

7 6 5 2

2 6 5 | 7

6 2 5

5 2 | 6

2 | 5

2