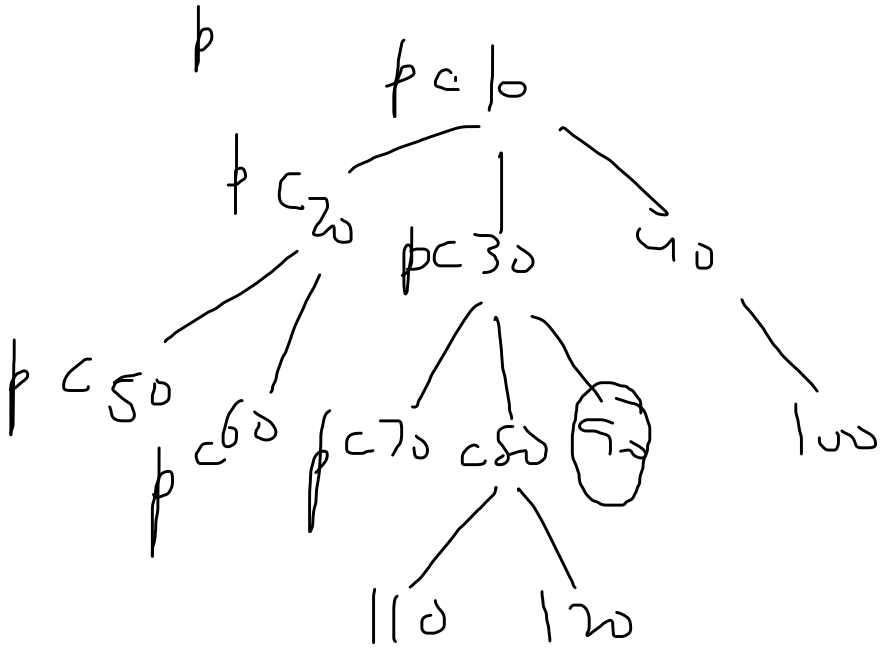
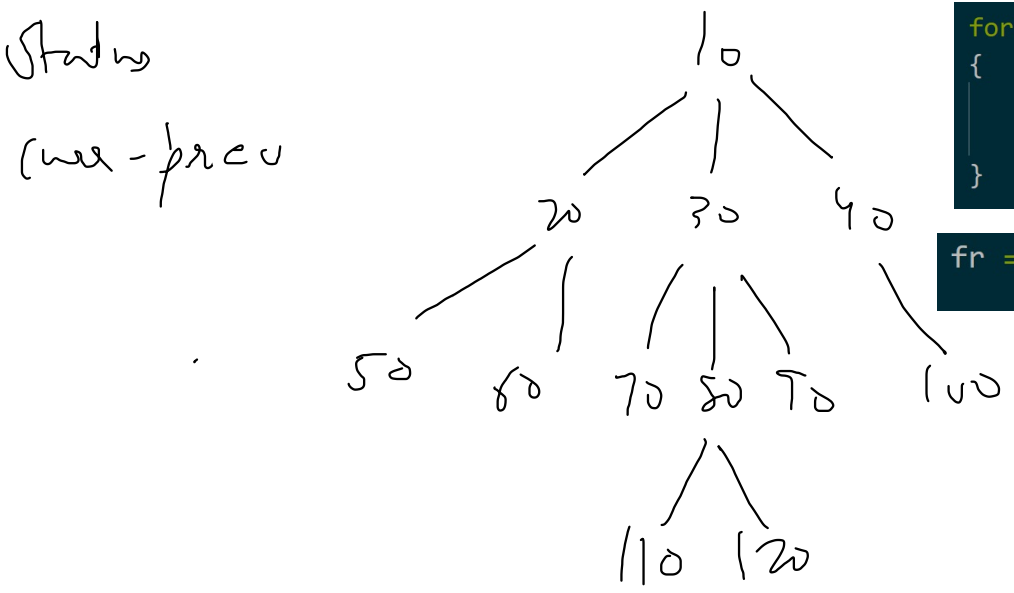


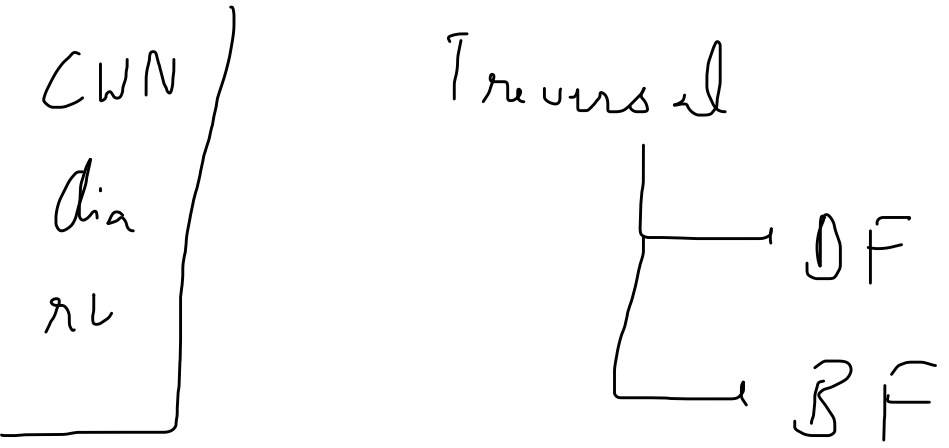
C. d $\left[p_{ind} = pv \right]$

p. v $\left[succ = cnt \right]$




```
for(int i = 0; i < k; i++)
{
    multisolverp(root, 0, sz, ht, mn, mx, cl, fr, res);
    res = fr;
}
```

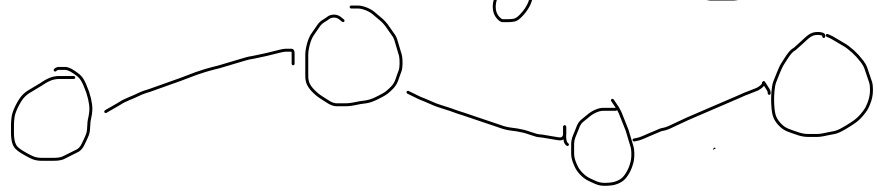
```
fr = root->data < key && root->data > fr? root->data: fr;
```


CWN
dia
RL
Transd
DF
BF

10
20
30
40
50
60
70
80
90
100
110
120

Array — get, set O(1), continuous, fixed

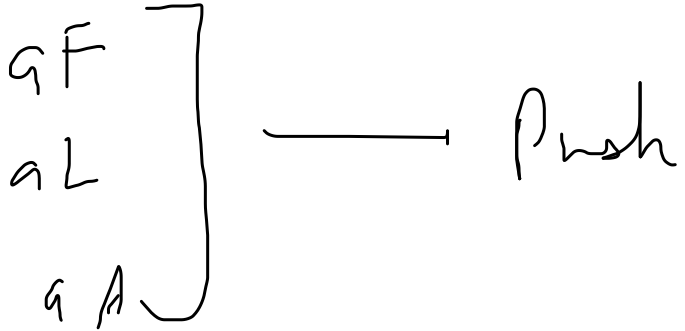
✓ Vector — dynamic array [Arr] [aF & rF

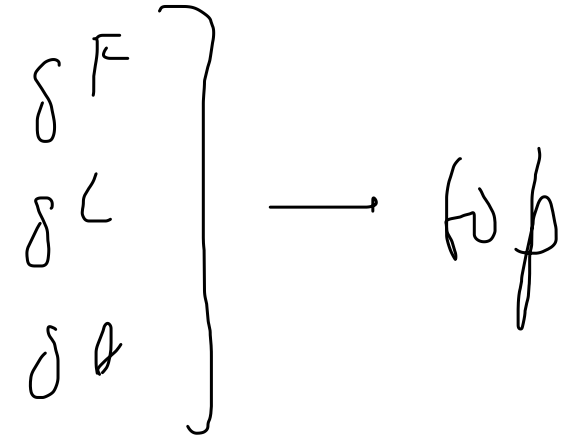
✓ LinkedList —  [rL]

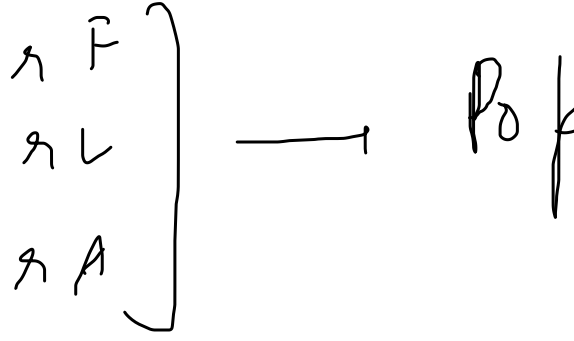
Stack — LIFO semantic

Queue — FIFO Semantic

Stack (LIFO)







Last - in first-out

Push 10

11 20

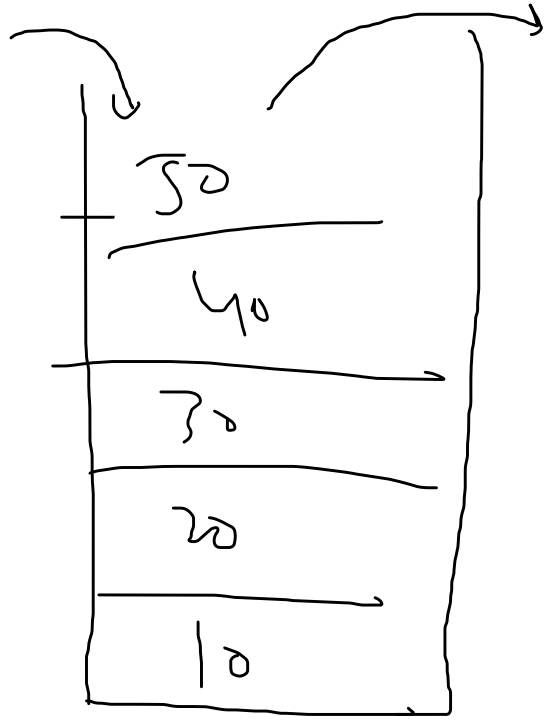
11 30

11 40

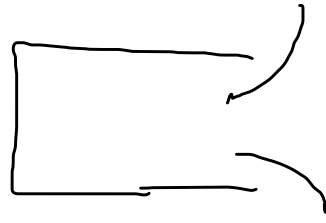
11 50

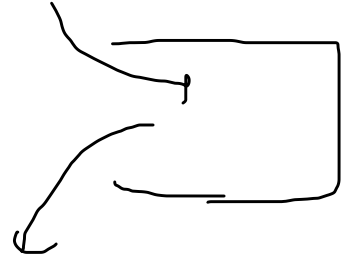
Top → 50

Pop



LIFO





Top → 50

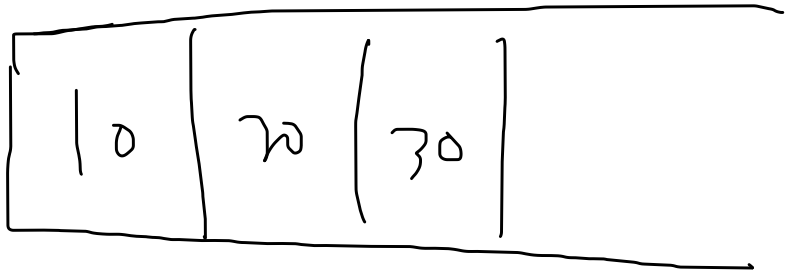
Pop

Abstract

Stack Implementations [Self, Adapt<V_LL>, Existing]

Vector	qFX	qFX	qL ✓	qL ✓
LinkedList	qF ✓	qF ✓	qL ✓	qL ✗

Vector as a stack [LIFO]



St	Vector
Push(10)	Push_back(10)
" (20)	" (20)
" (30)	" (30)
(40)	" (40)

Top → back() [40]

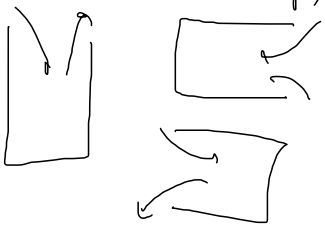
Pop → Pop_back()

Linked list as a stack

→ A, V, LL, S, O

→ V → Dynamic
→ LL → Memory
→ S → LIFO
→ O → FIFO

S Adapted (Push, Pop, Top) [LIFO]



S Imp → Self, Adapt, Existing

V LL

V as a Stack → aL, rL, gL

LL as a " → gF, rF, gF

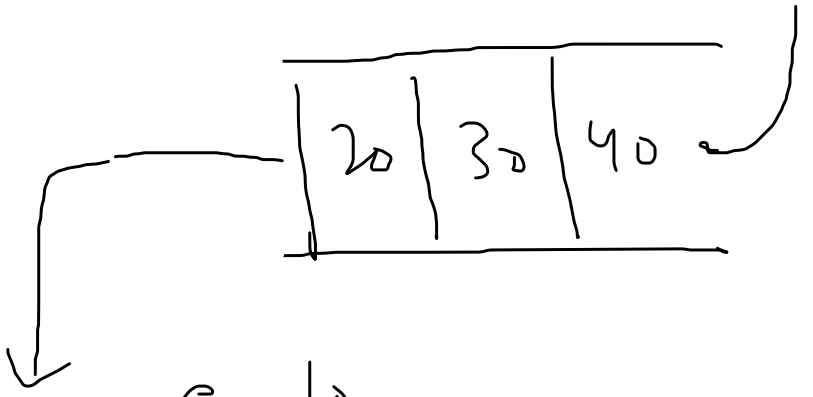
C++	Java
V → Push-back Pop-back back	AL → add() remove(s-1) get(s-1)
LL → Push-front Pop-front front	LL → aF() rF() gF()

Queue Abstract (FIFO)

aF, aL, aA] → Enqueue

rF, rL, rA] → dequeue

sF, sL, sA] → front



en 10

en 20

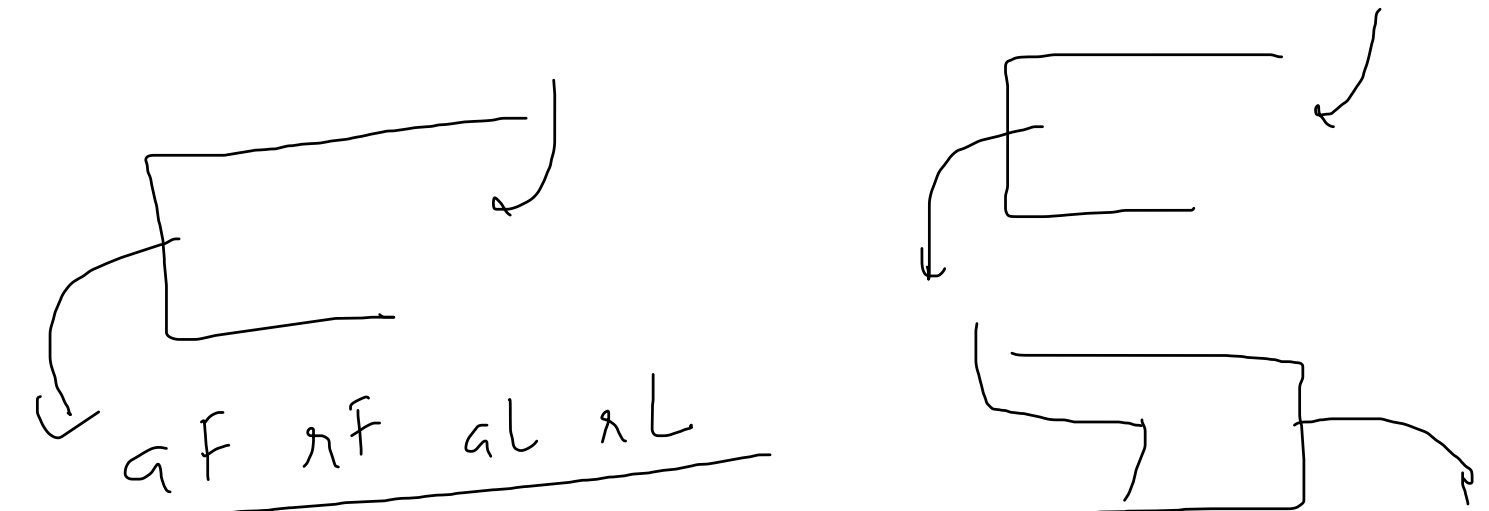
en 30

en 40

f (10)

de

Own Implmt



qF rL

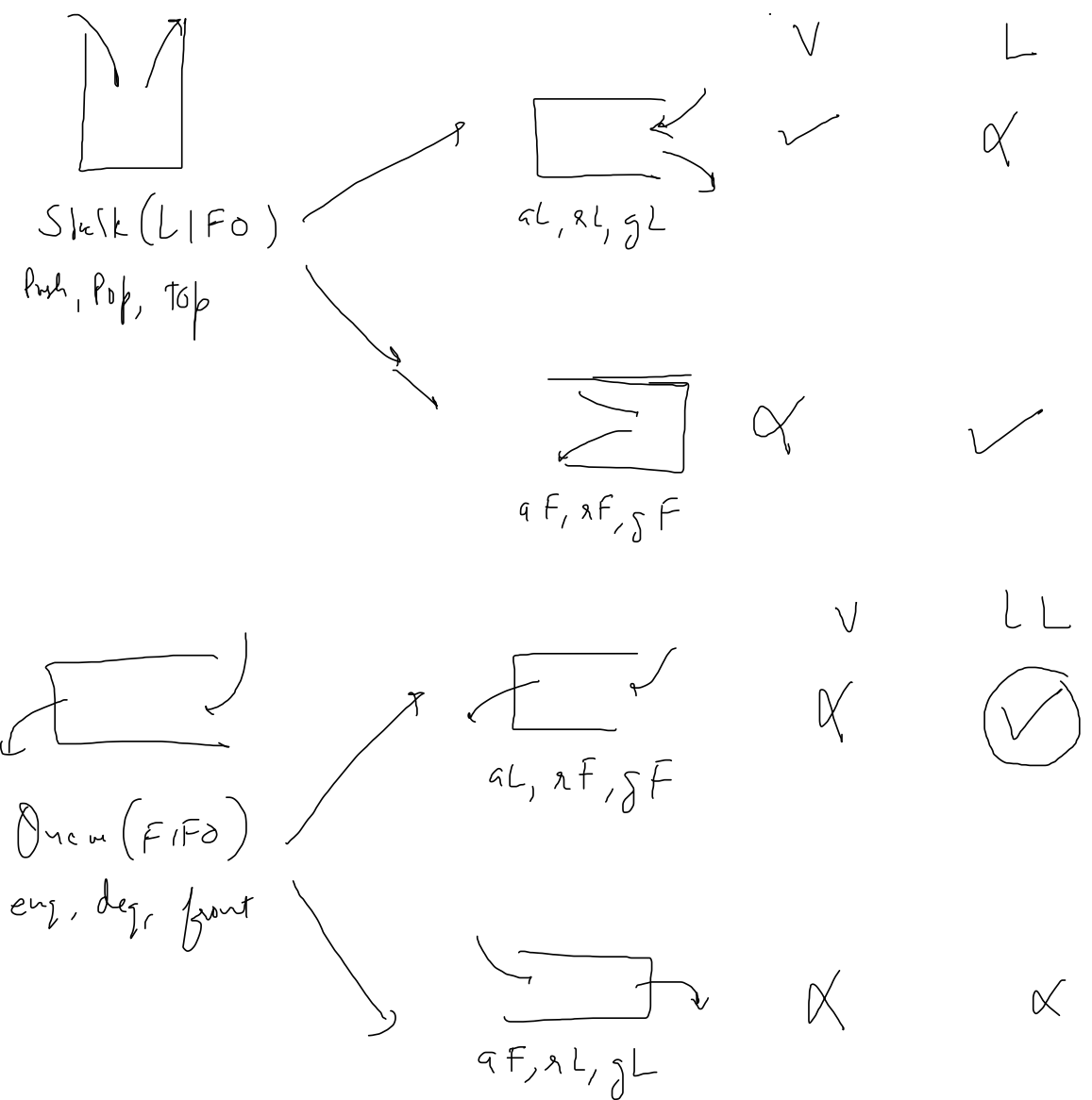
✗

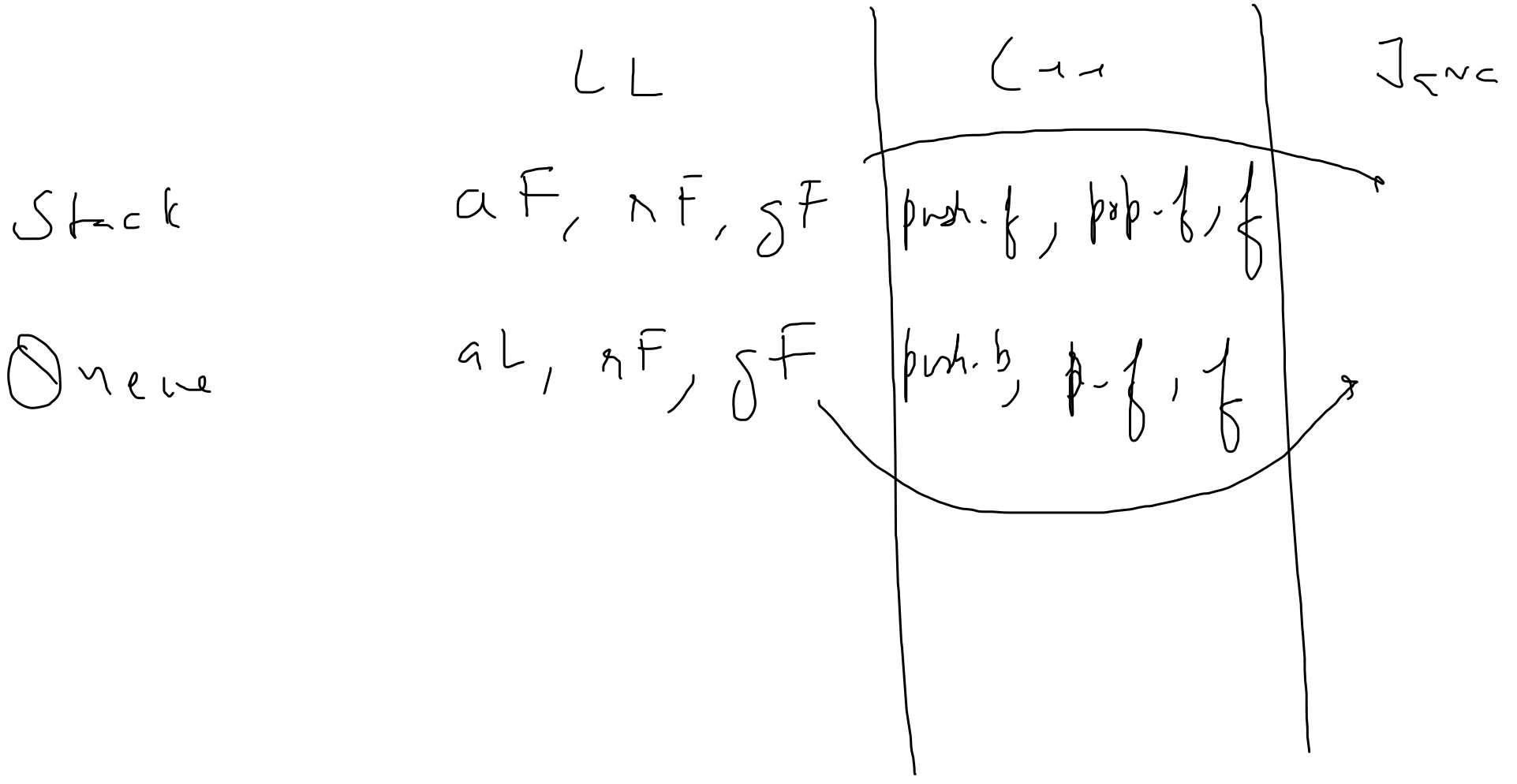
✗

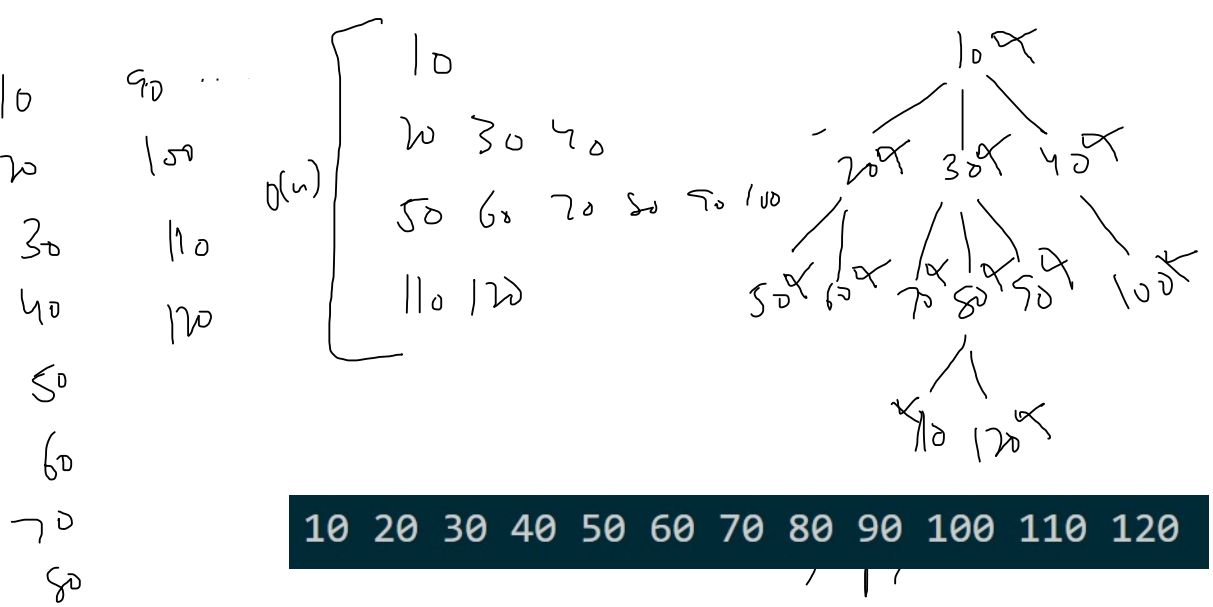
qL rF

✗ → vector

✓ → LL


V
L
Stack (LIFO)
Push, Pop, Top
aL, rL, gL
aF, rF, gF
V
LL
Queue (FIFO)
enq, deq, front
aL, rF, gF
aF, rL, gL


LL
C++
Func
Stack
aF, nF, sF
push_f, pop_f, f
Queue
aL, nF, sF
push_b, p_f, f



```
10 20 30 40 50 60 70 80 90 100 110 120
```

DF vs BF

LO

S > C

O(n)

level 1

1

level 2

2 3 4

5 6 7 8 9 10

11 12

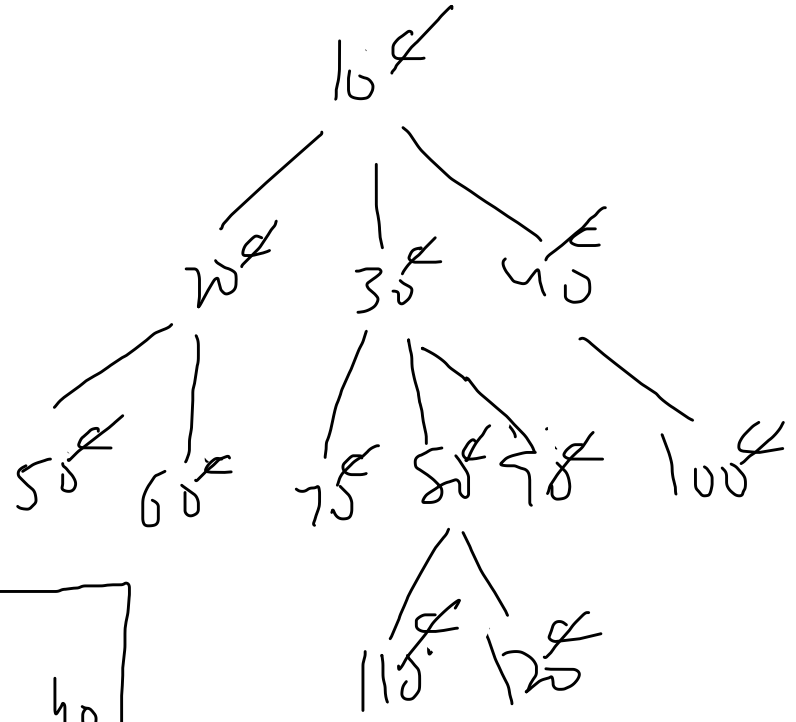
level 3

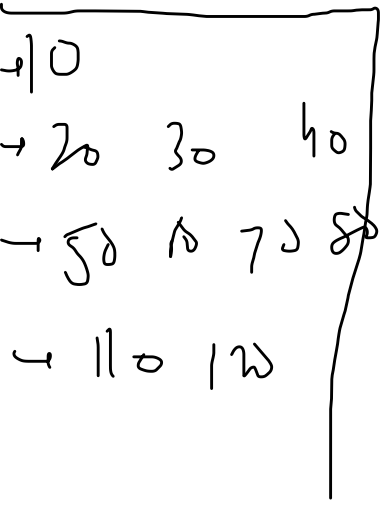
1

4 3 2

5 6 7 8 9 10

12 11





1. δ
2. R
3. P

C

4. a c in neat

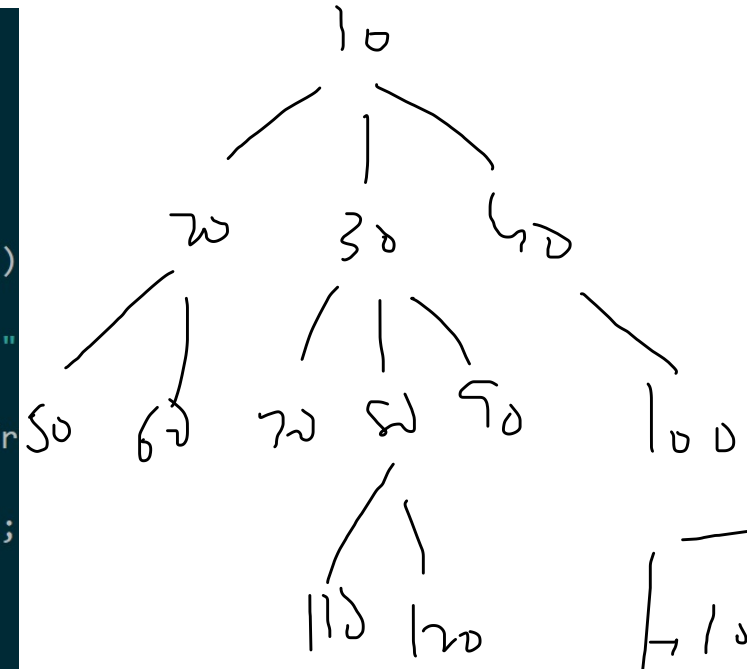
5. if C.S == 0

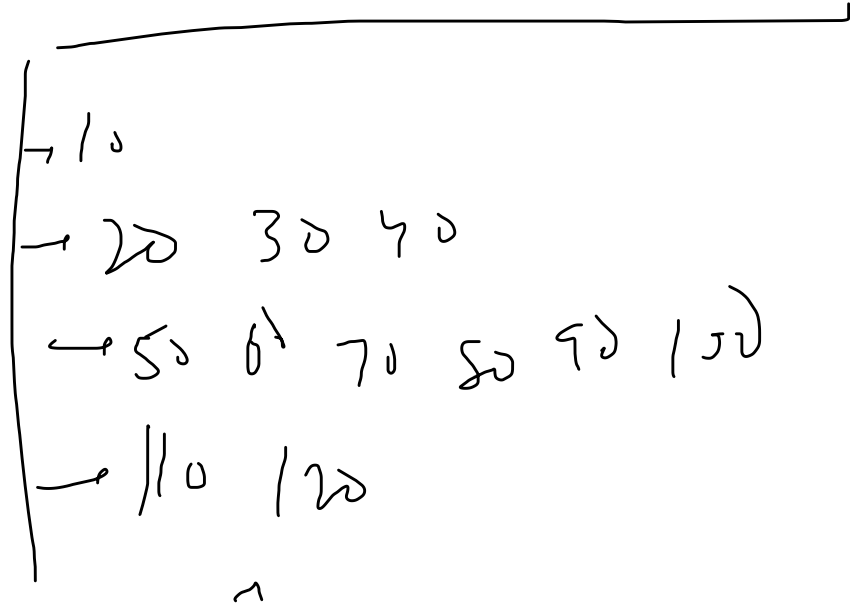
C = n

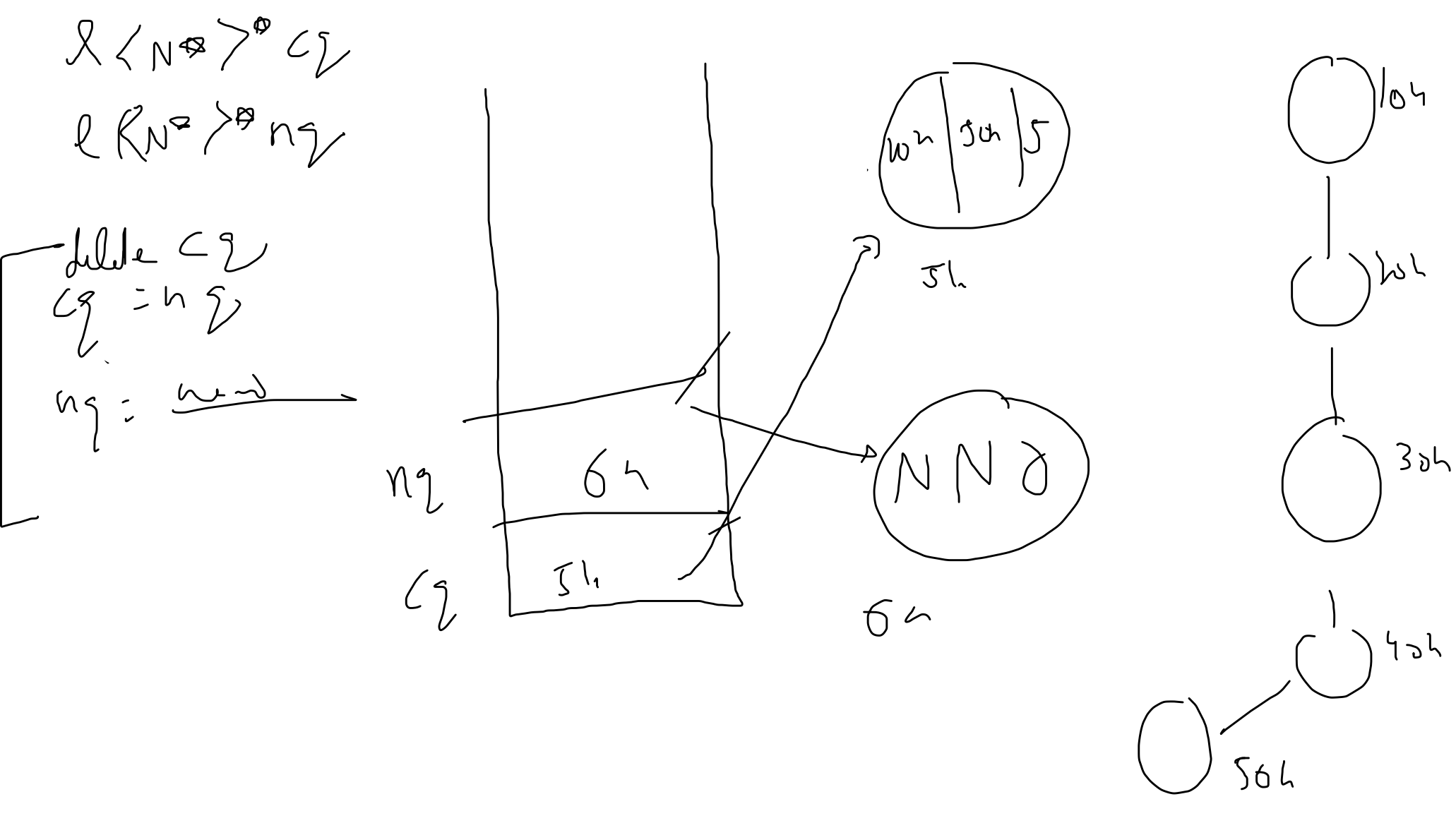
n = new

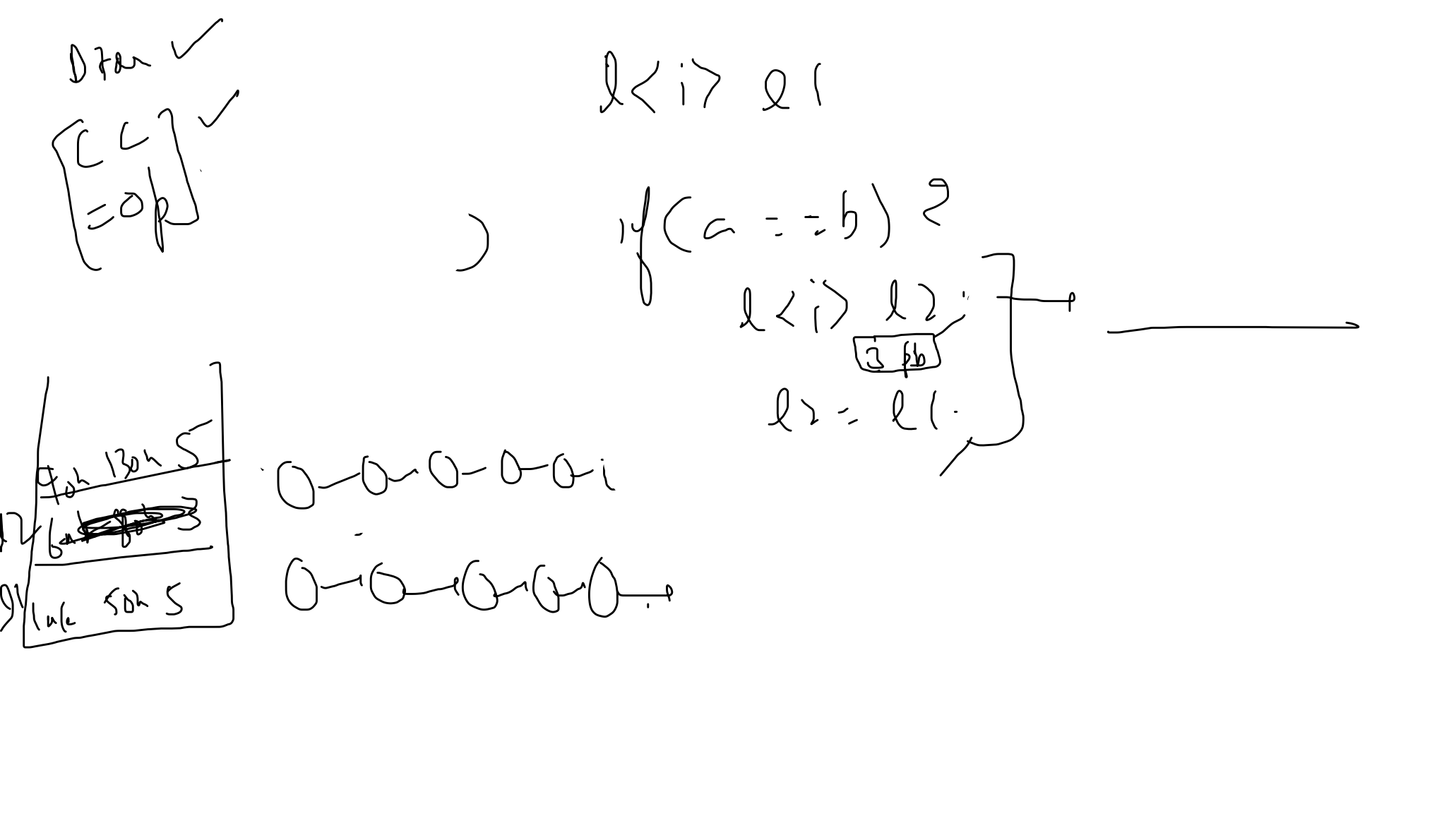
Sysо( )

```
while(cq.size() > 0)
{
    list<Node*> nq;
    while(cq.size() > 0)
    {
        Node* fr = cq.front()
        cq.pop_front();
        cout << fr->data << "

        for(int i = 0; i < fr
        {
            nq.push_back(fr);
        }
    }

    if(cq.size() == 0)
    {
        cq = nq;
        nq.clear();
        cout << endl;
    }
}
```







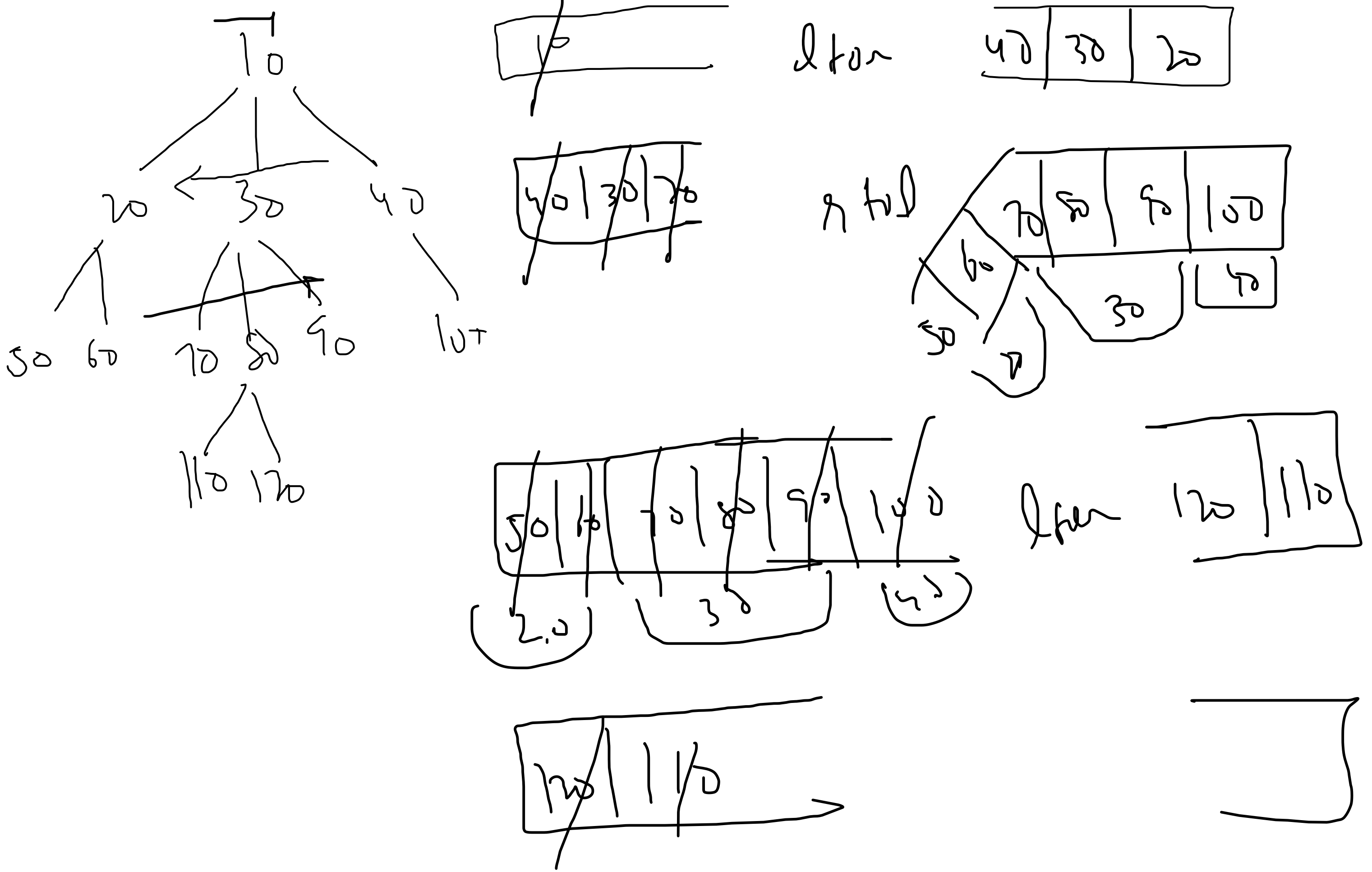


10

40 30 20

50 60 70 80 90 100

120 110



10

40 30 20

50 60 70 80 90 100

120 110

BFT(10)

S > C

Queue

10

20 30 40

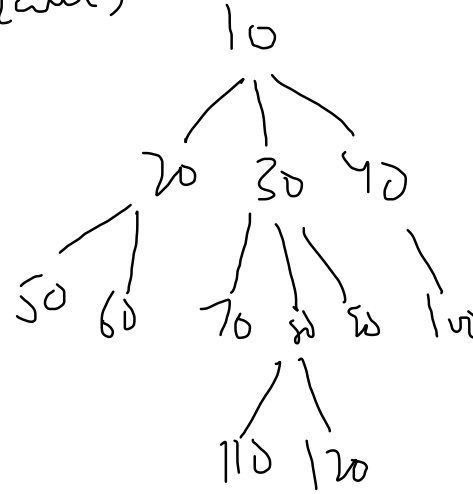
50 60 70 80 90 100

110 120

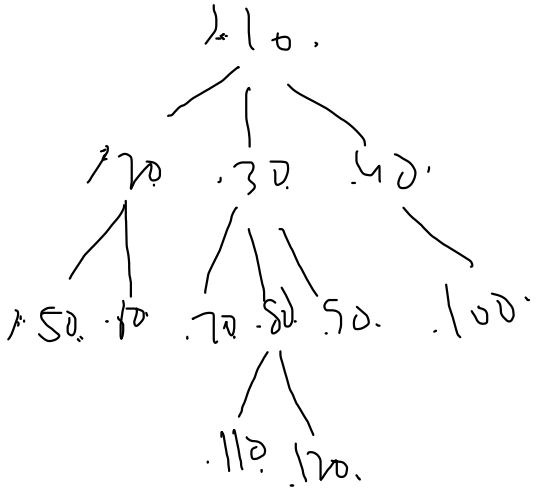
DFT(Pre0, Post0, In0) [Euler]

C > S

Stack



DFT (Pre, Post, In)



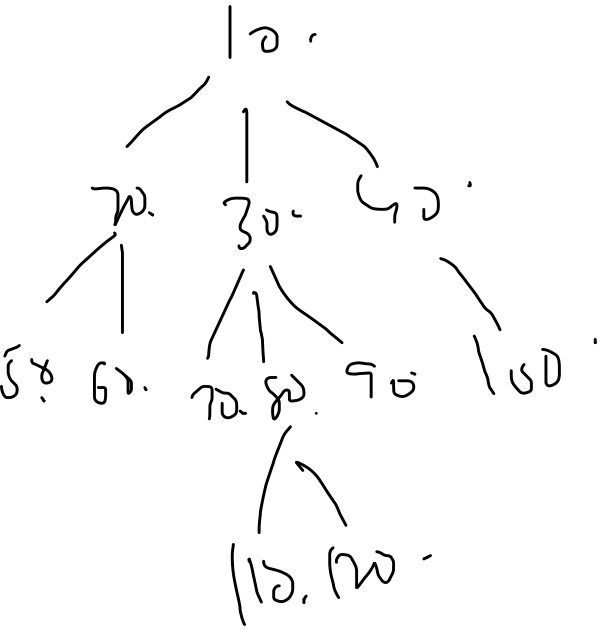
Pre

- Root is first
- Then left
- Recursion — on the way down

$n > C$

```
void preo(Node* root)
{
    // node pre
    cout << root->data << " ";
    for(int i = 0; i < root->children.size(); i++)
    {
        // edge pre
        preo(root->children[i]);
        // edge post
    }

    // node post
}
```

DFT (Pre, Post, Gn)



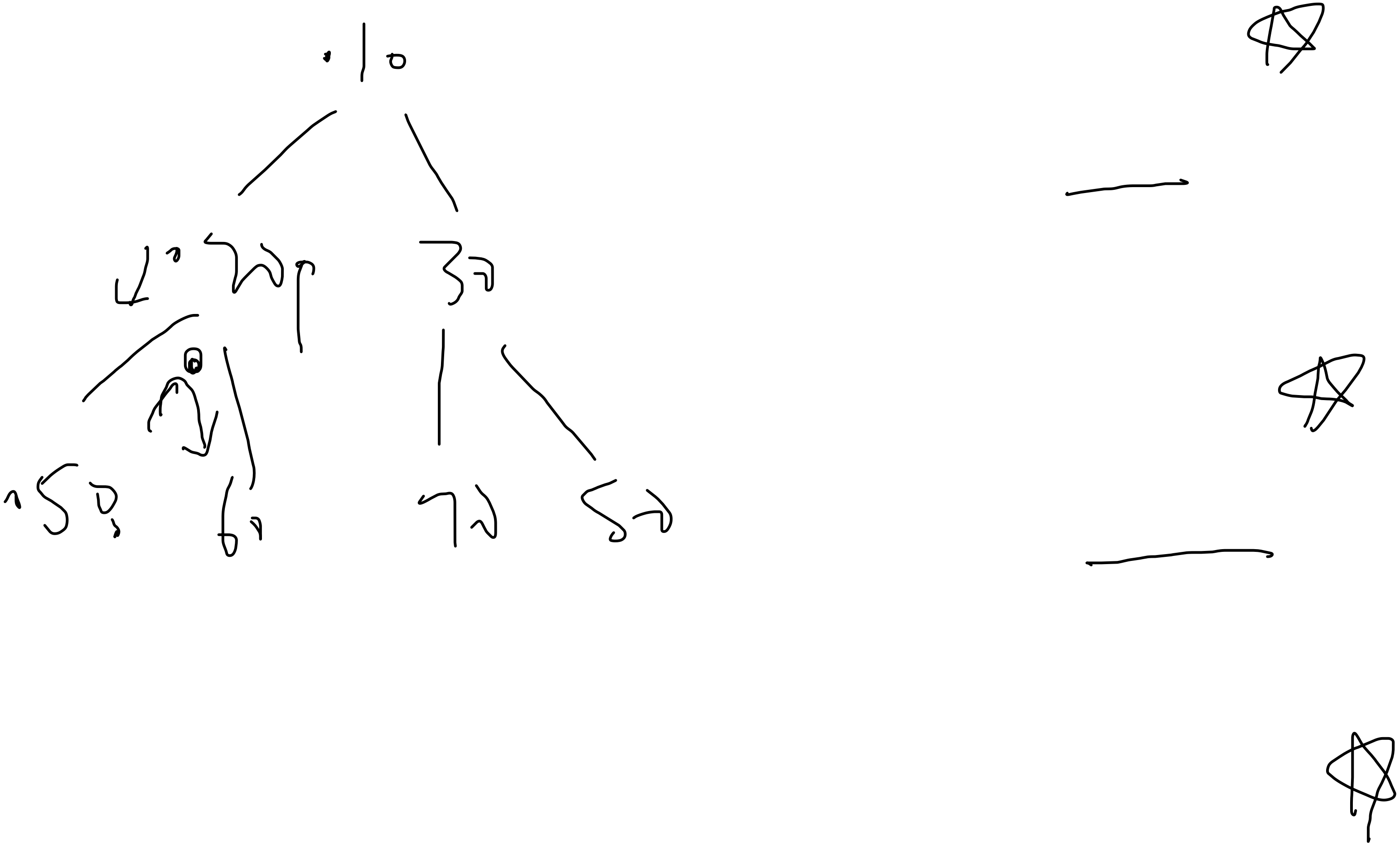
O(n)

```
void posto(Node* root)
{
    // node pre

    for(int i = 0; i < root->children.size(); i++)
    {
        // edge pre
        posto(root->children[i]);
        // edge post
    }

    cout << root->data << " ";
    // node post
}
```

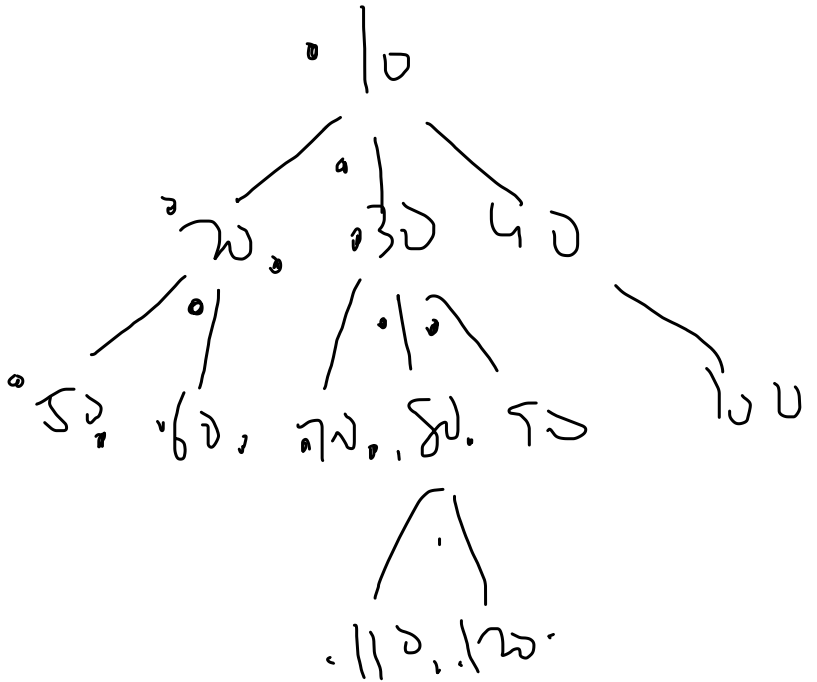
Post

Root is last

Recursion



DFT Iteratively



10 Pre
20 Pre
50 Pre
50 Post
20 in
60 Pre
60 Post
30 Post
10 in
30 Pre

70 Pre
40 Post
30 in
80 Pre
110 Pre
110 Post
80 in
90 Pre
120 Post
80 Post
30 in2

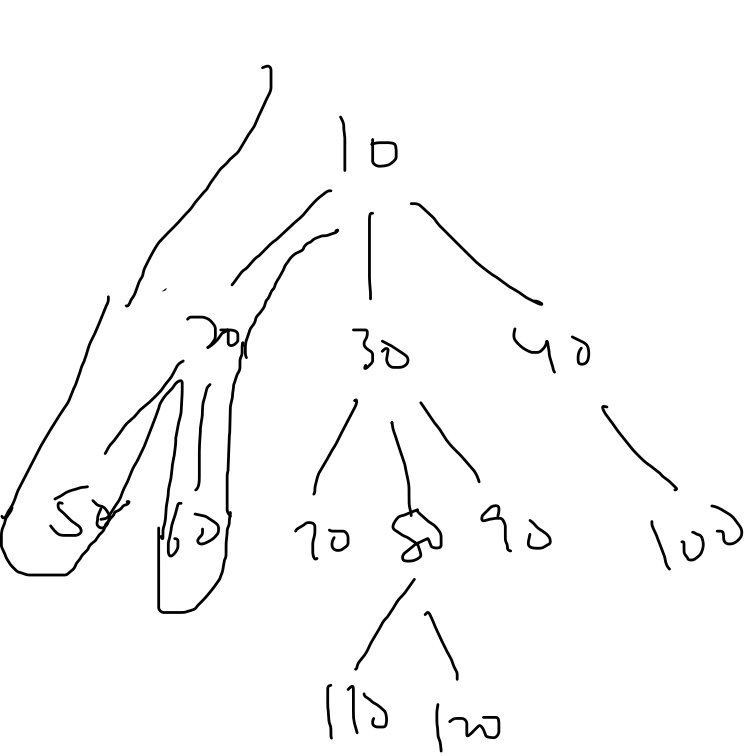
→ dL

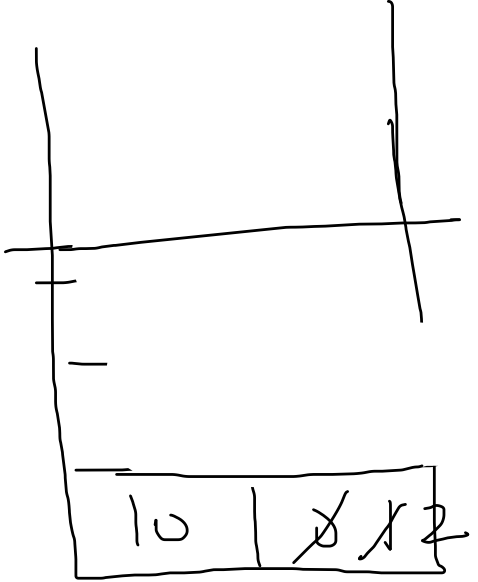
→ dia

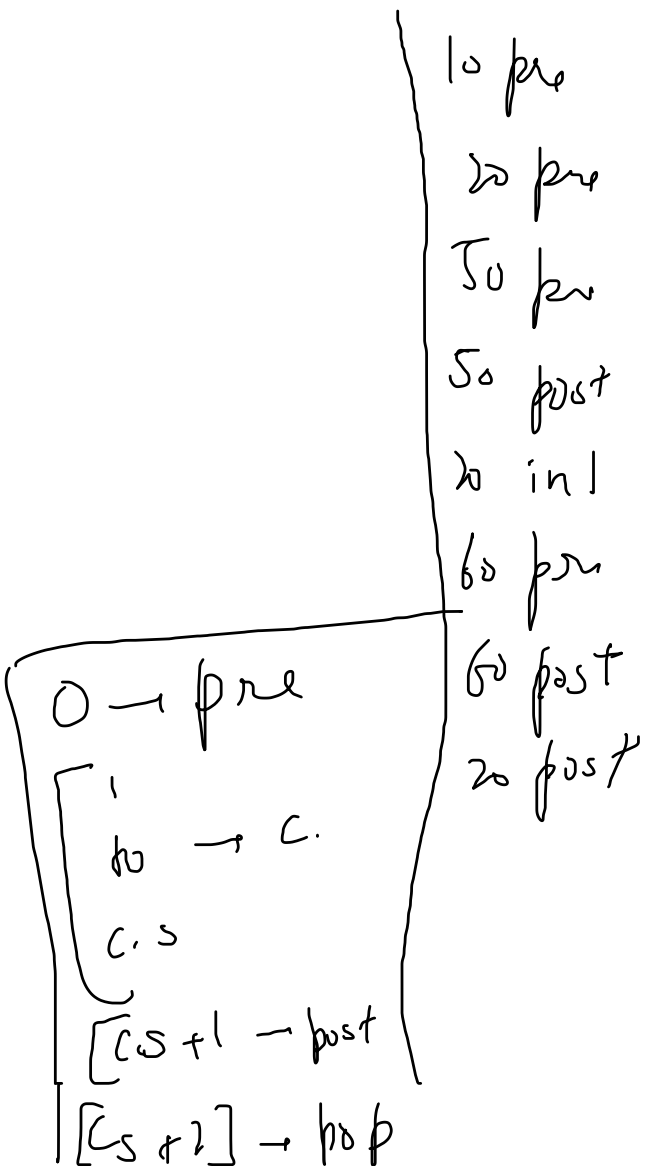
→ cWeird

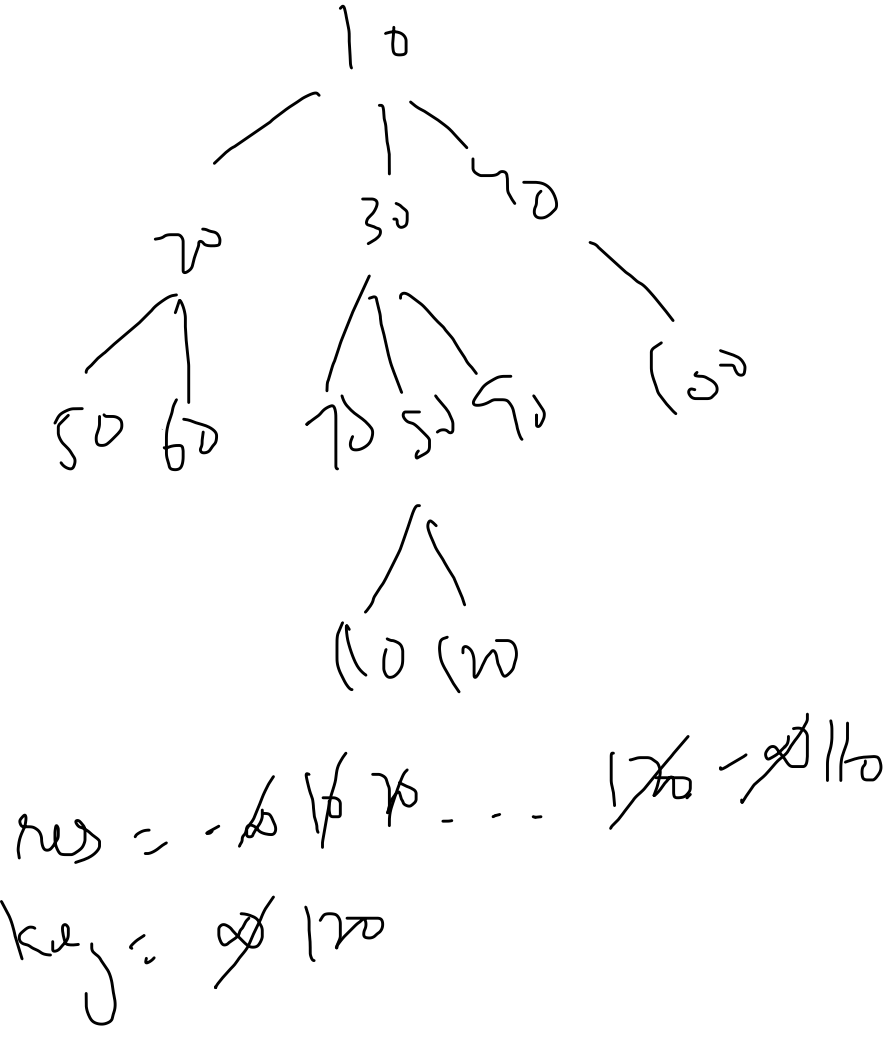
→ max nblsum

→ " lblsum


10
20
30
40
50
60
70
80
90
100
110
120


10


10 pre
20 pre
30 pre
50 post
40 in
60 pre
60 post
20 post
0 → pre
1
40 → c.
c.s
[cs+1] → post
[cs+2] → pop



```
void floorforKthLargest(Node* root, int& fr, int key)
{
    fr = root->data < key && root->data > fr? root->data: fr;
    for(int i = 0; i < root->children.size(); i++)
    {
        floorforKthLargest(root->children[i], fr, key);
    }
}
```

```
int kthLargest(Node* root, int k)
{
    int res = INT_MIN;
    int key = INT_MAX;
    for(int i = 0; i < k; i++)
    {
        floorforKthLargest(root, res, key);
        key = res;
    }
```

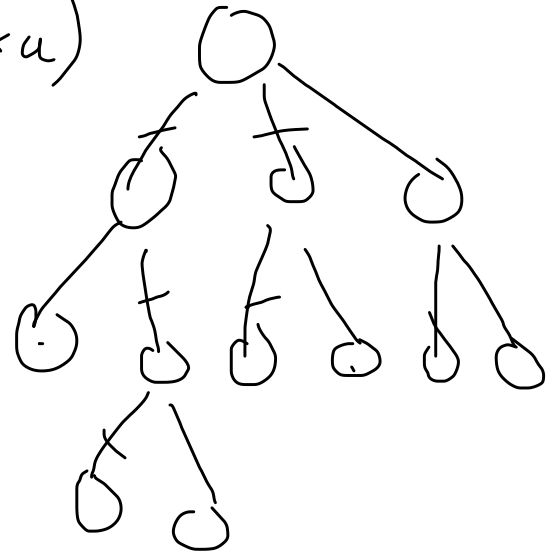
→ CountNodes (Nodes with subtree higher than that of parent)

→ diameter [On, no space]

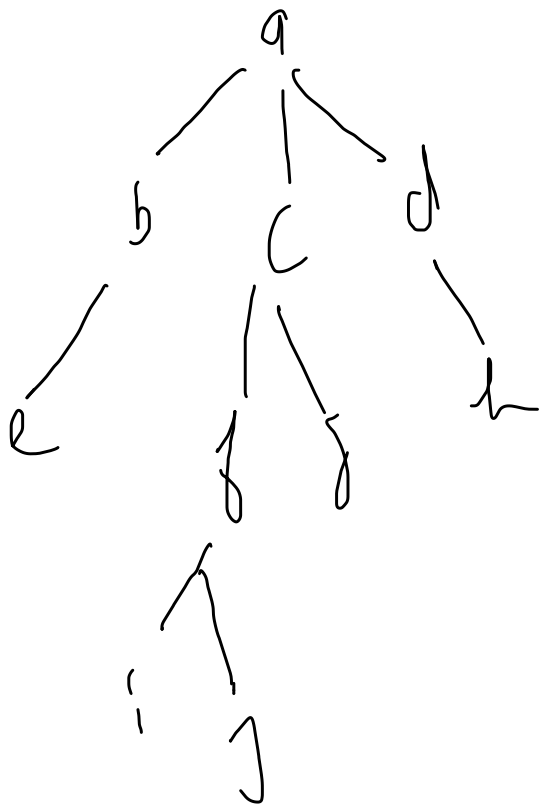
→ max node to leaf sum

→ max leaf to leaf sum

→ RL

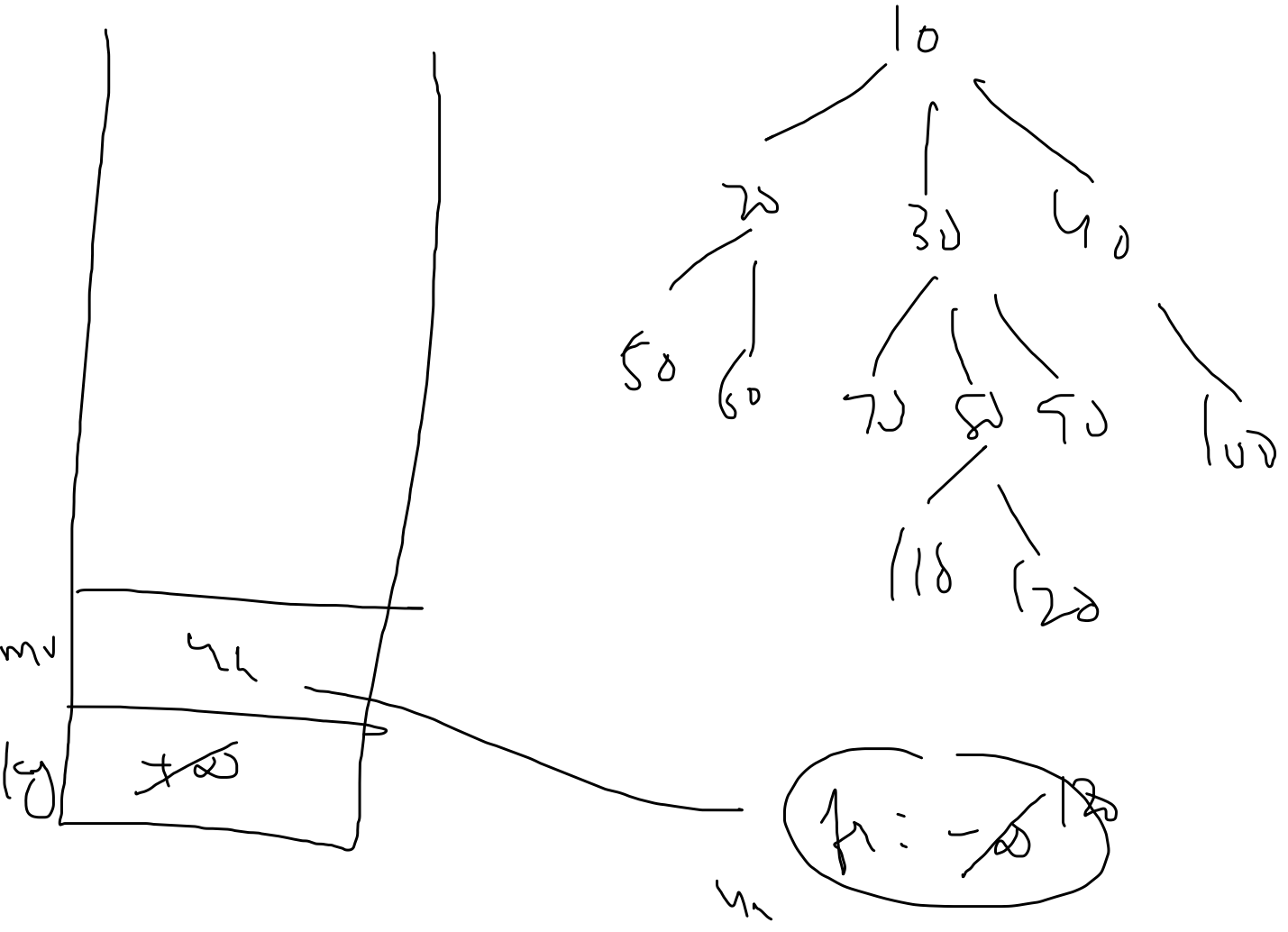


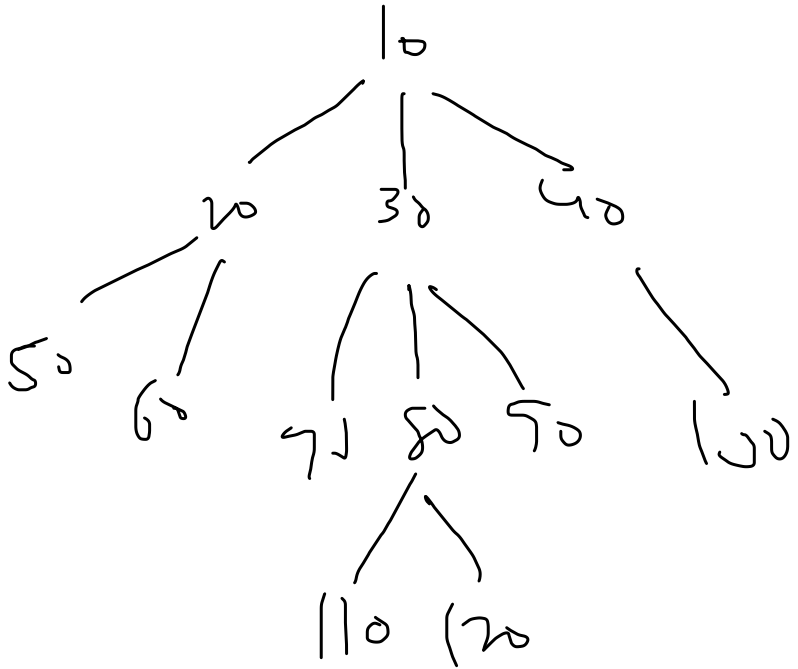
$^{n}C_{2}$



a b e	b e	e	
a c f i	c f i	f i	i
a c f j	c f j	f j	j
a c g	c g	g	
a d h	d h	h	

e b a c f i	e b a d h	i f c a d h	g c a d h
e b a c f j	i f g	j f c g	
e b a c g	i f c g	j f c a d h	





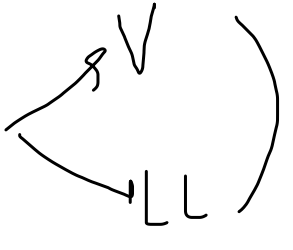
Level Order

10 20 30 40 50 60 70 80 90 100 110 120

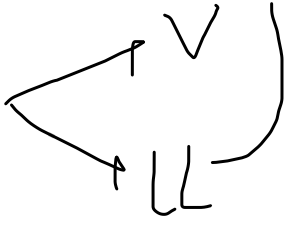
→ Array, V|AL, LL, S, O

→ S Abstract

→ O "

→ S Implement [Self, Pre-existing, Adapt 

)

→ O " [", " , Adapt 

]

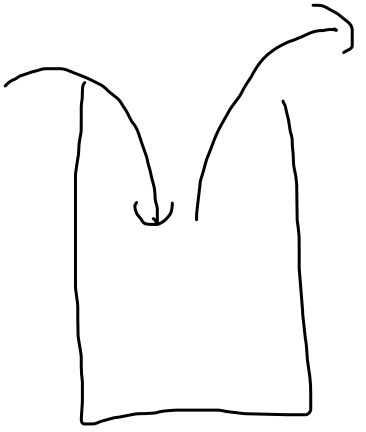
Array [Contiguous, Fixed → Access O1

V | AL → Dynamic Array [Sdc] [Access O1]

LL → Memory Optimised [Access On]

	AF	AL	RF	RL	SF	SL
LL	O1	O1	O1	On	O1	O1
V	On	O1	On	O1	O1	O1

Stack (Abstract) → LIFO Semantic



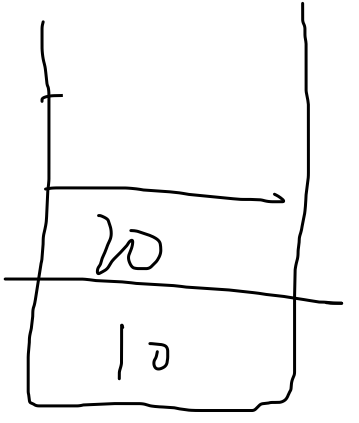
aF
aL
aAt
→ Push

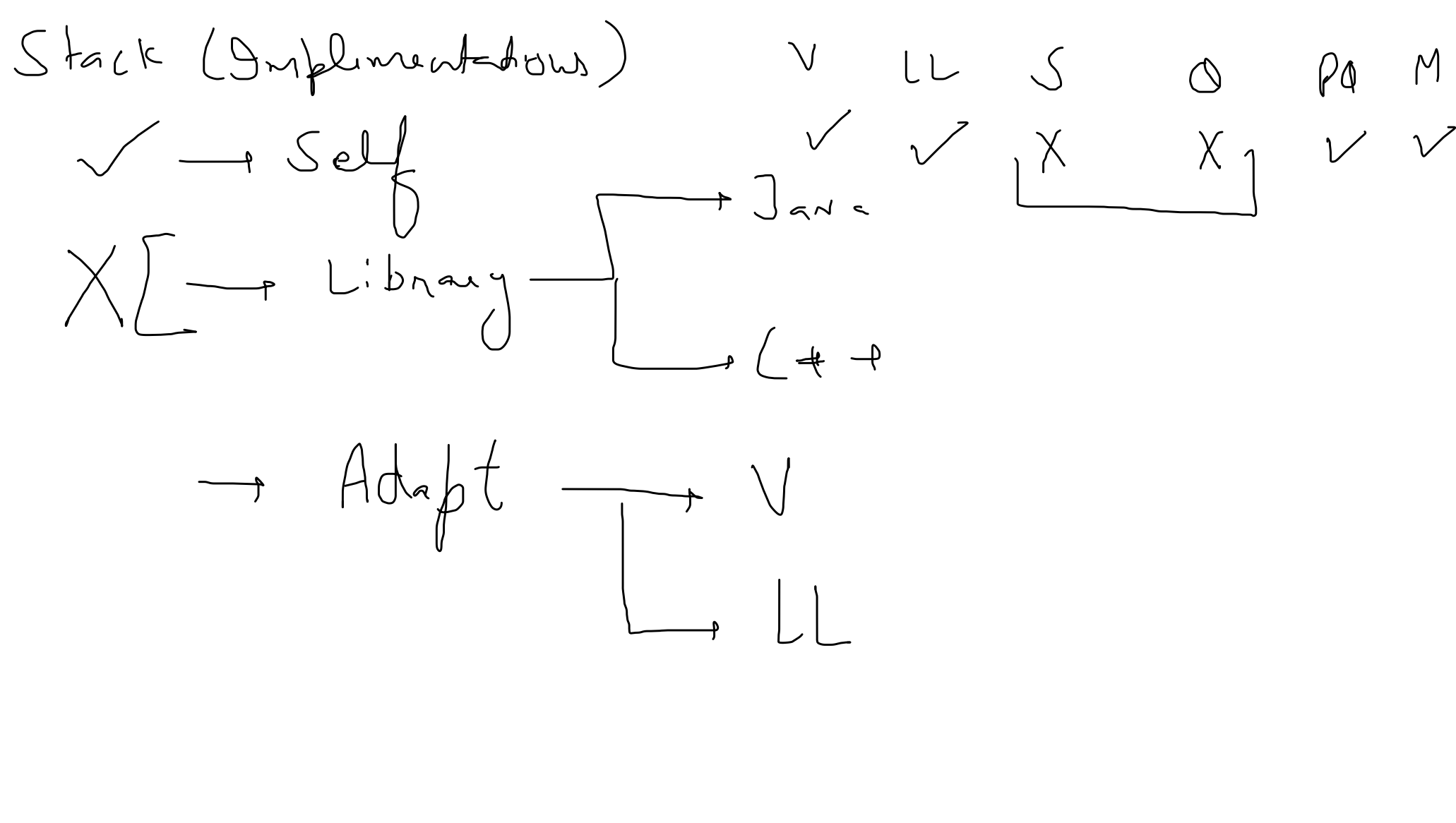
ΛF
ΛL
ΛA
→ Pop

δF
δL
δA
→ Top

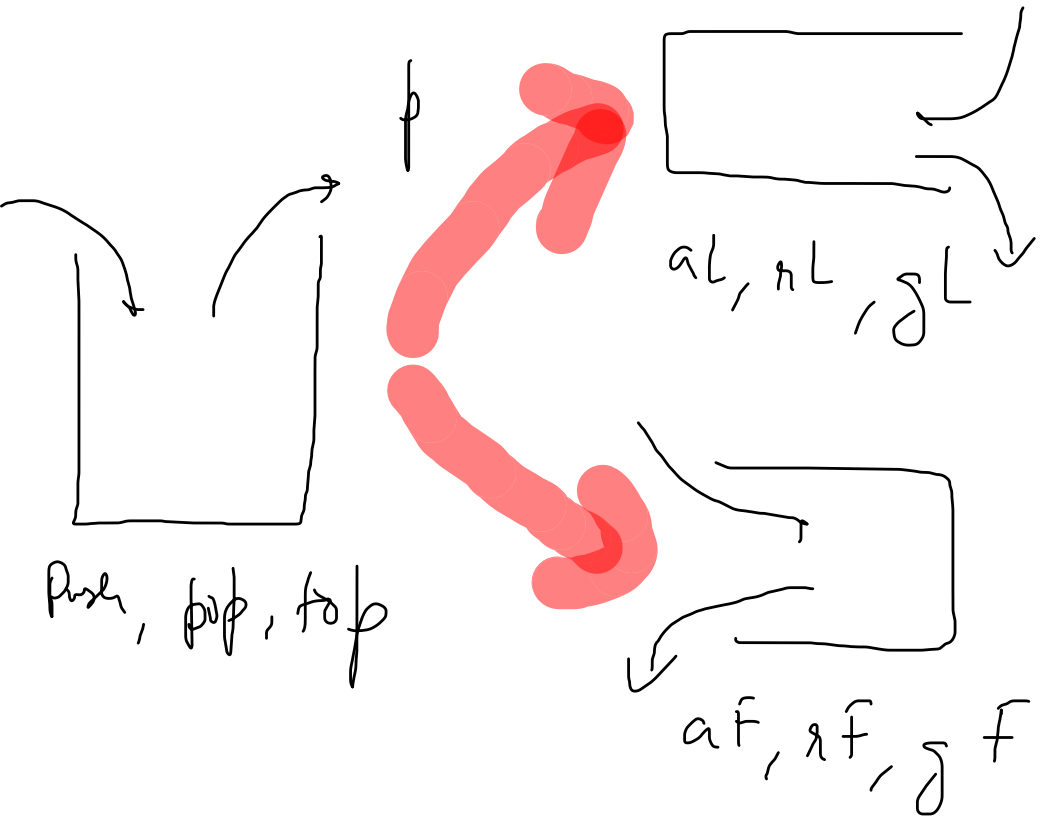
Push 10
Push 20
Push 30
Push 40

Top → 40
Pop
Top → 30
Pop




Stack (Implementations)
V
LL
S
Q
PQ
M
✓
✓
X
X
✓
✓
✓ → Self
X → Library
Java
C++
→ Adapt
V
LL

Stack Adaptor (Adapter Design Pattern)



V

LL

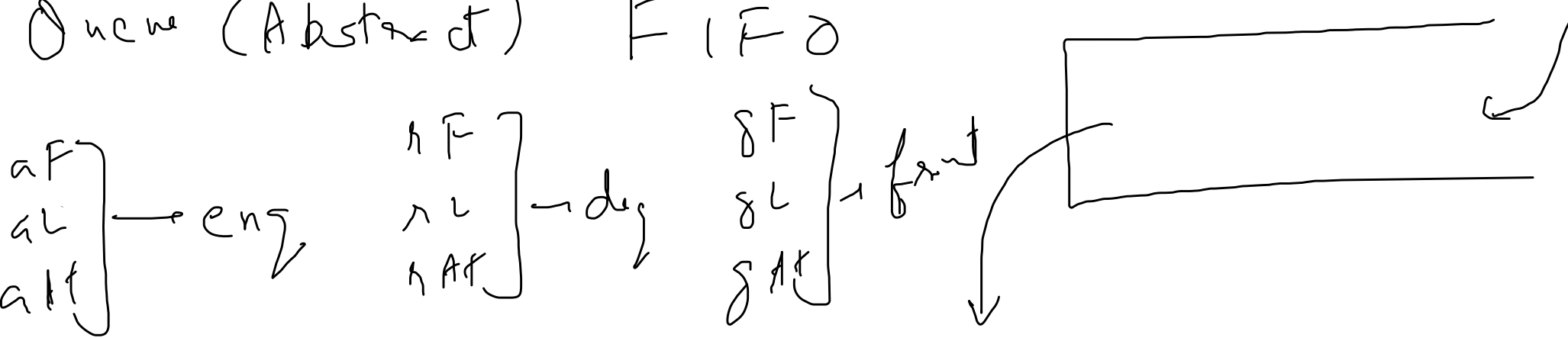
C++	Java
ps_b	add
po_b	r(s-1)
back	g(s-1)

X

X

C++	Java
ps_f	aF
po_f	rF
front	gF

Queue (Abstract) FIFO



eng/to

— 20

— 30

— 40

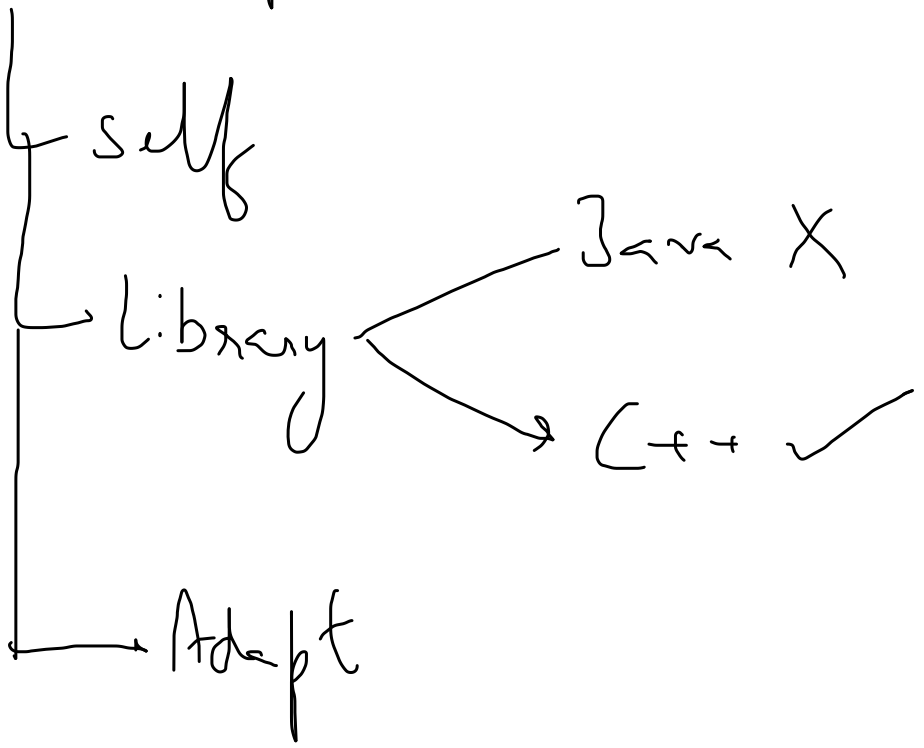
front 10

deg

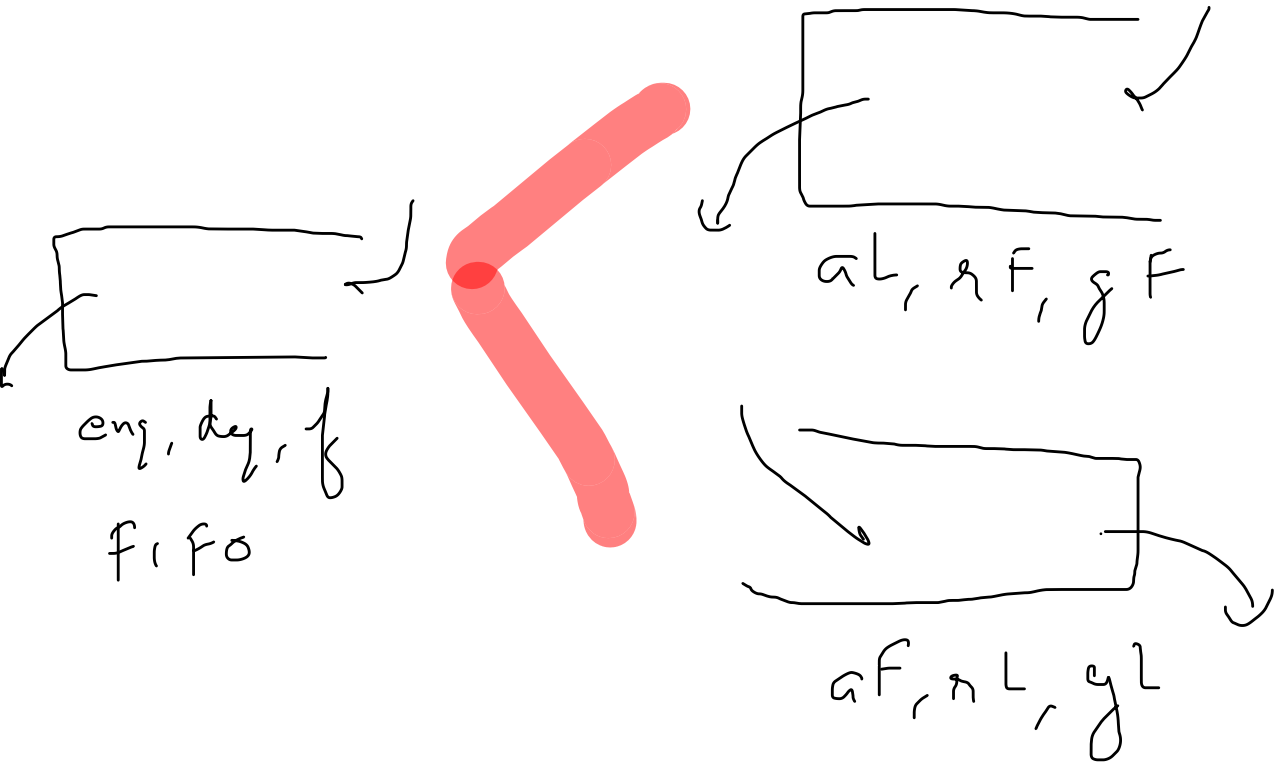
front 20

30	40

Owen Implementation



Queue Adaptions.



V	LL	
X	✓	
	C++	Java
	Ps_b	a
	Po_f	rF
	front	gF
X	X	

Stack

push, pop, top

Vector

aL, rL, gL

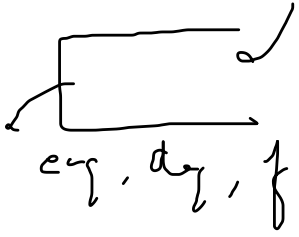
C++	Java
ps-b	add
po-b	r(s-1)
back	g(s-1)

LL

aF, rF, gF

C++	Java
ps-f	aF
po-f	rF
f	gF

Queue

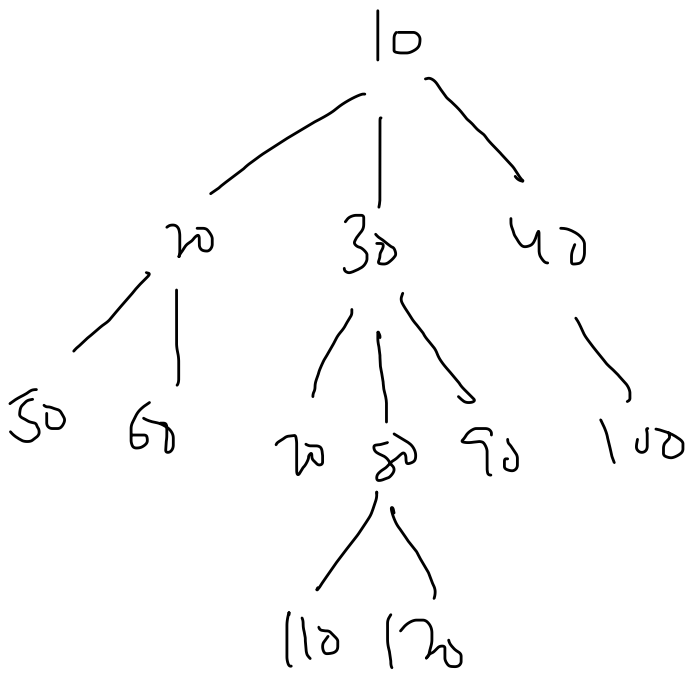


eq, dq, f

X

aL, rF, gF

C++	Java
ps-f	aL
po-f	rF
front	gF



Level order	Breadth First
10 20 30 40 50 60 70 80 90 100 110 120	

S > C	Queue

~~10^0~~	~~20^1~~	30^1	40^1	50^2	60^2

10 20

$[g, r, p, ac] \neq 0$

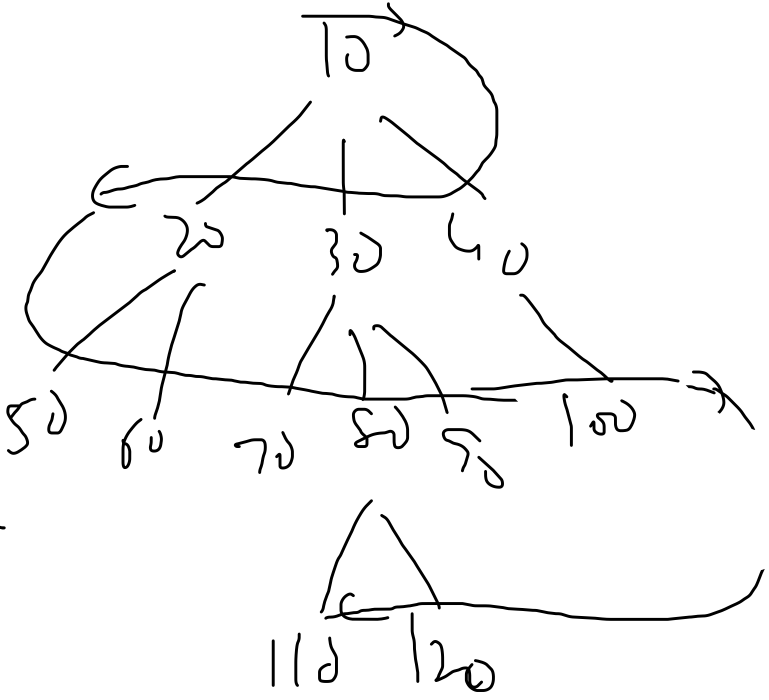
lolw

10

20 30 40

50 60 70 80 90 100

110 120



---

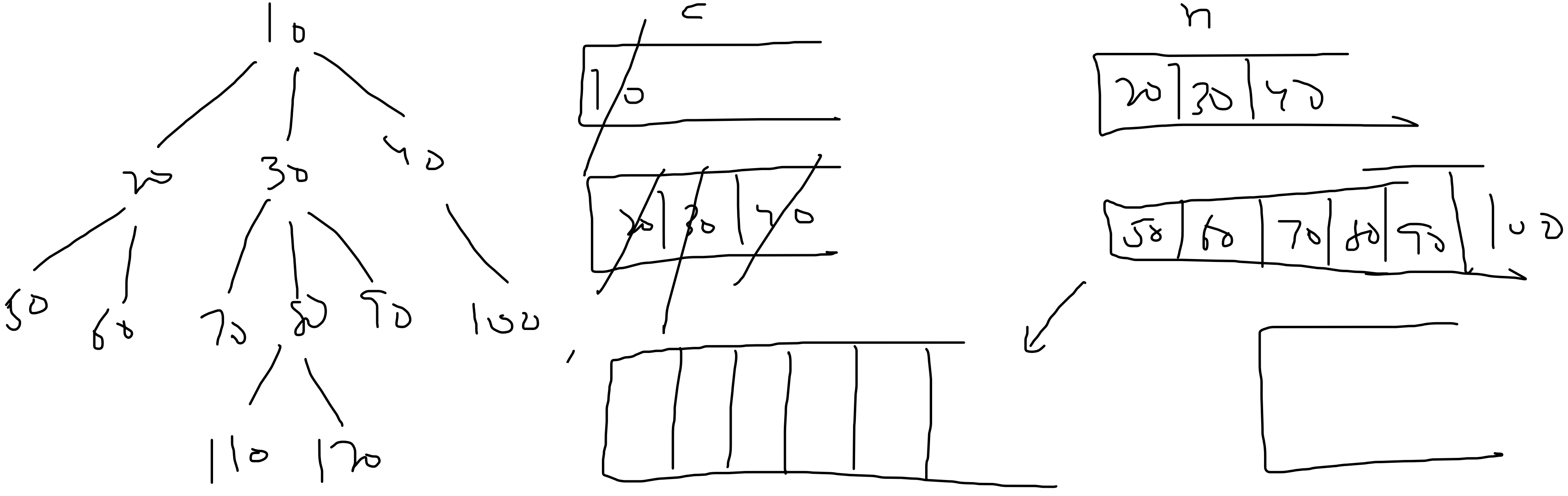
lolw22

10

40 30 20

50 60 70 80 90 100

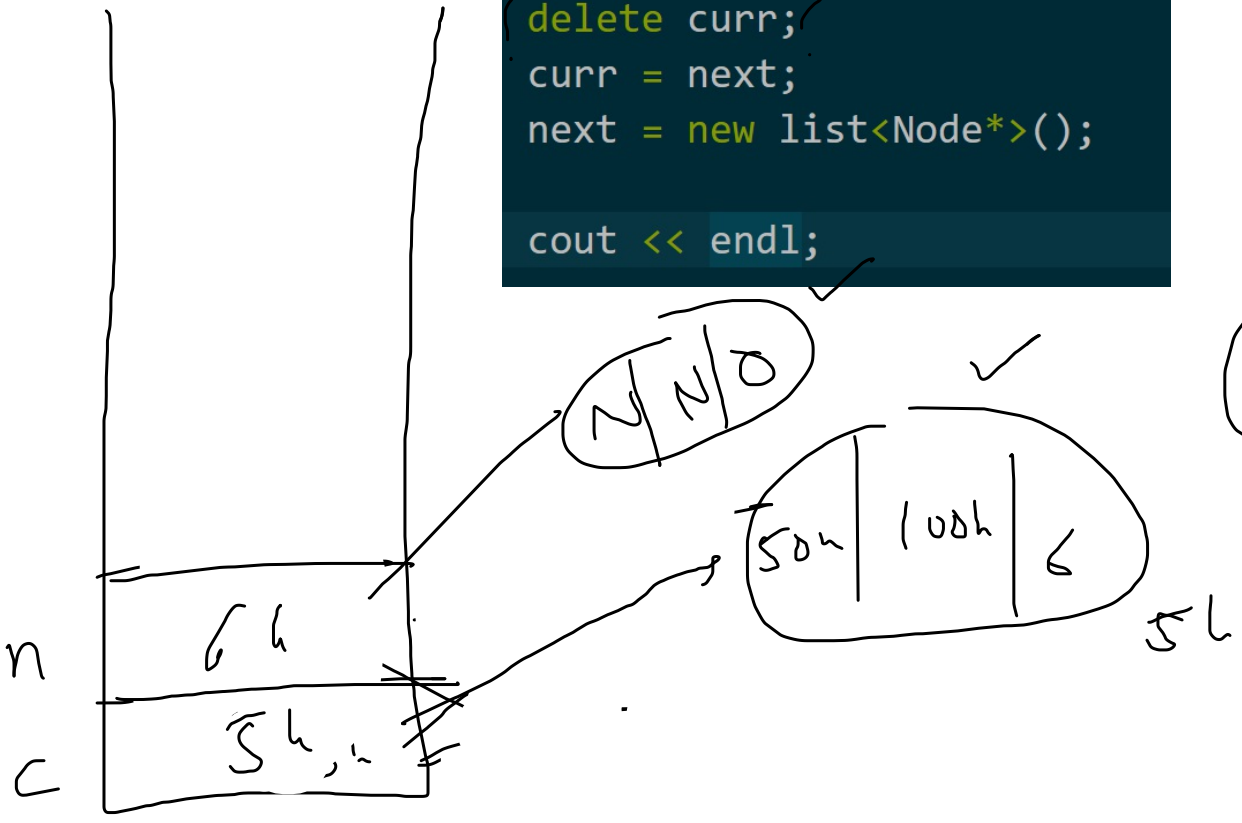
120 110

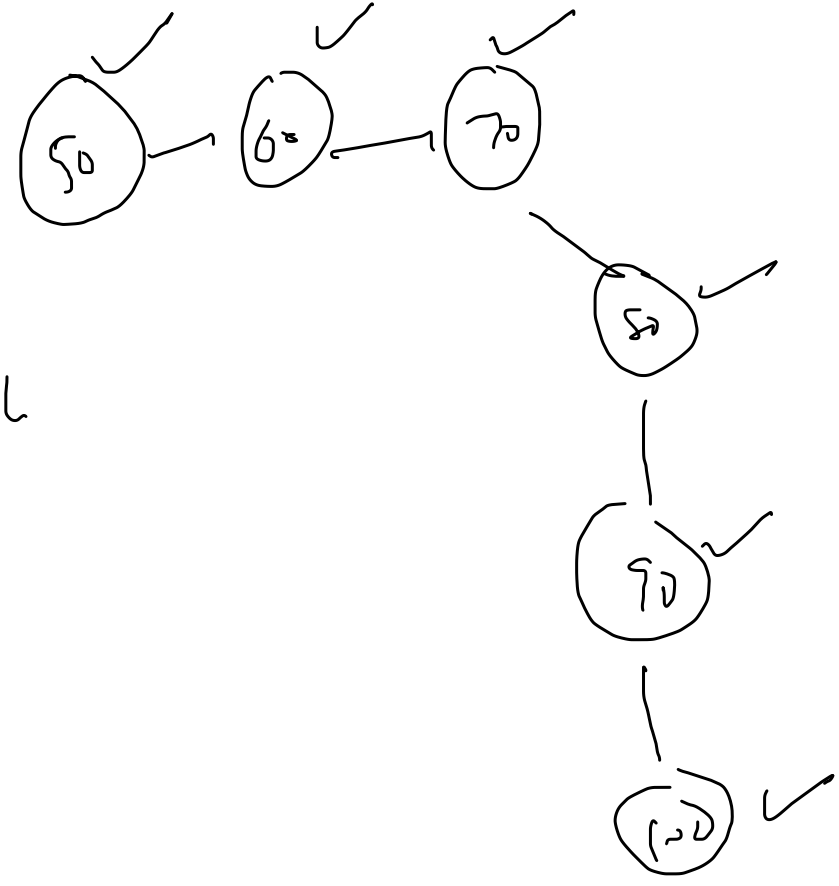


g, r, p, ac*
+
[c=n, n=c, d)

→ 10
→ 20 30 40
→ 50 60 70 80 90 100
→ 110 120

```
delete curr;
curr = next;
next = new list<Node*>();

cout << endl;
```

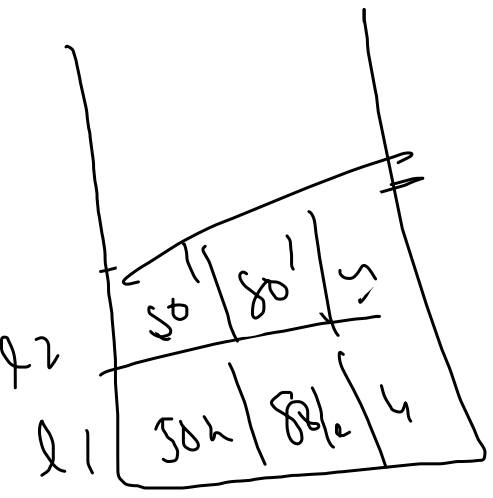


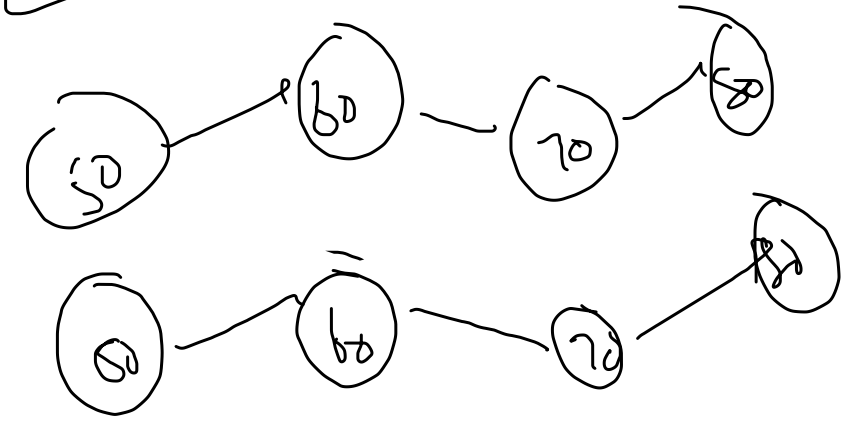


curr = next

next.clear

Rof Three → [destructor ✓
copy constructor ✓
op = ✓





LL<1> l2 = l1

LL<i> l2;

l2.pb(10)

l2.pb(20)

l2 = l1

destructor

CC X

op = X

L<i> l1;

l1.pb(10)

—.pb(20)

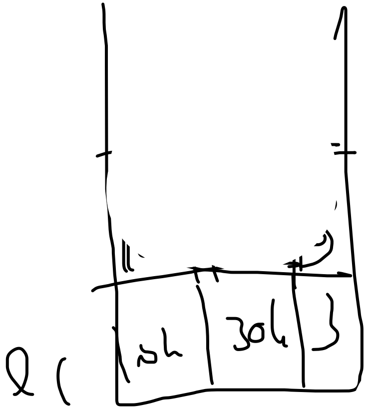
—.pb(30)

if(——) {

L<i> l2 = l1 ;

}

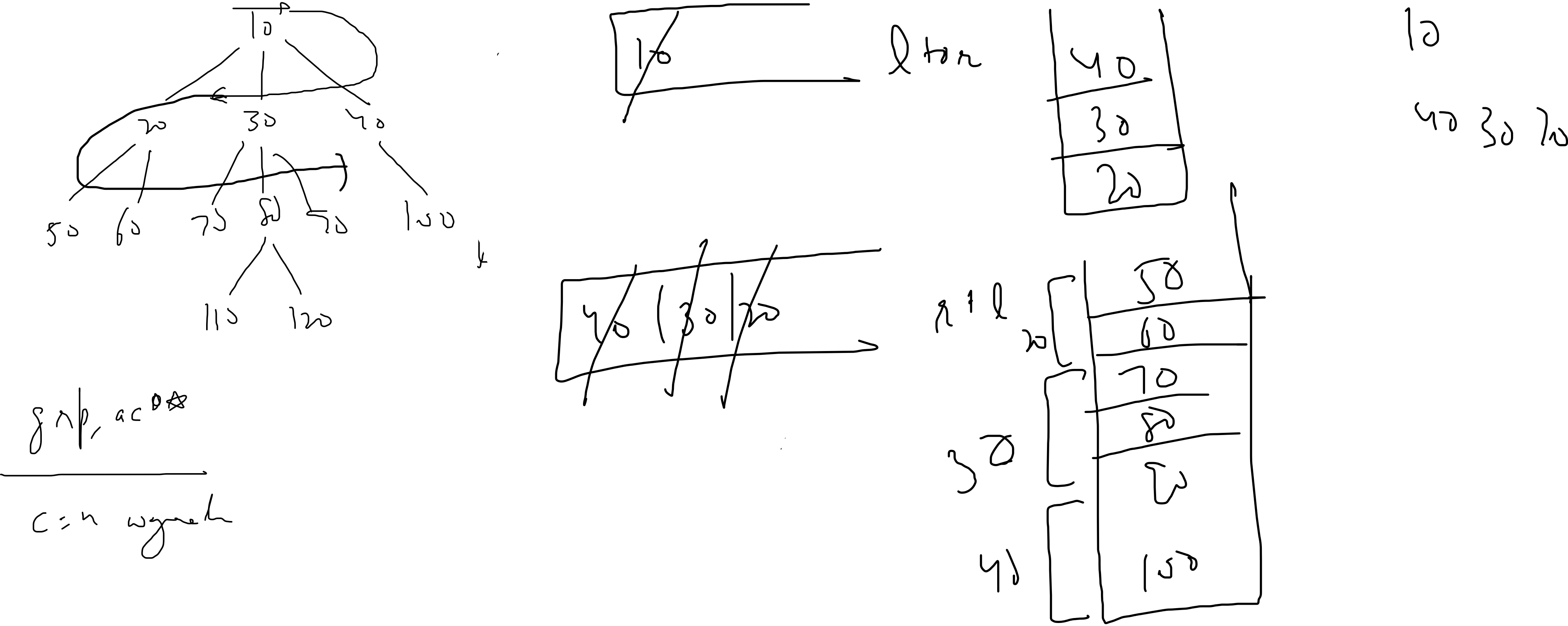
cout << l1[0];

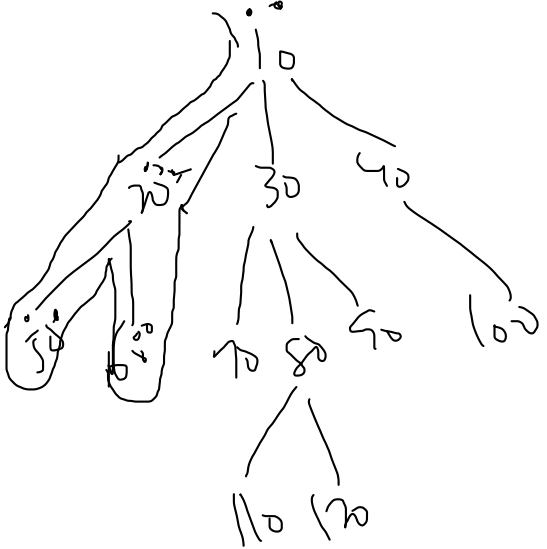


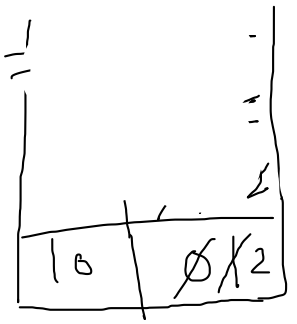
1o pre
2o pre
5o pre
5o post
2o in 1
6o pre
6o post
2o post
1o in 1

3o pre
7o pre
7o post
3o in 1
5o pre
11o pre
11o post
8o in 1
12o pre
12o post

8o post
3o in 2


10
20 30 40
50 60 70 80 90 100
110 120
40 30 20
50 60 70 80 90 100
10
40 30 20





10 pre
20 pre
30 pre
50 post
20 in
60 pre
60 post
20 post

```
S = 0   [pre, S++]
S = 1 to C.S (child[S-1], Sel)
S = C.S+1   (post, Sel)
S = C.S+2   pop
```