

Today's Content:

- { ✓ → Level order traversal without extra space
- ✓ → fill Right pointer
- ✓ → Level order Generalizability
- ✓ → De-Serializc
- Other doubts

Class Node { Q) print BT in level order without extra space

int data

= SC: O(1)

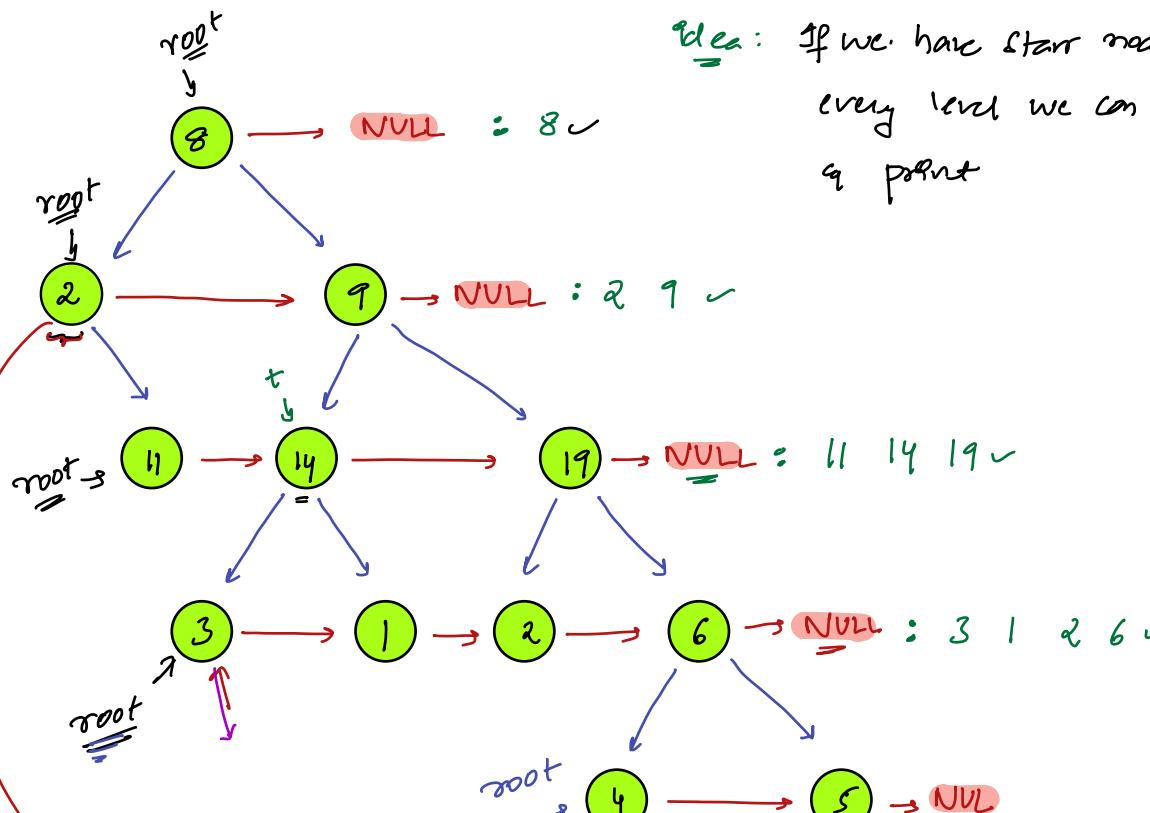
Node left

Node right

Node next

node points to next node in same level

level: 8 2 9 11 14 19 3 1 2 6 4 5



Idea: If we have start node at every level we can iterate q point

root = NextChild(2) : return 11

root = NextChild(3) : return 4

root = NextChild(4) : return NULL

```
void levelorder( Node root ) {
```

```
    while( root != NULL ) {
```

```
        Node temp = root
```

```
        while ( temp != NULL ) {
```

```
            print( temp.data )
```

```
            temp = temp.next
```

// we iterate all
all node i time

```
        if ( root.left != NULL ) {
```

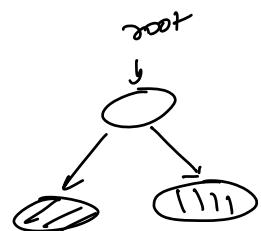
```
            root = root.left
```

```
        } else if ( root.right != NULL ) {
```

```
            root = root.right
```

```
        else,
```

```
            root = nextChild( root )
```



or

we iterate n
all node
1 times

```
root = nextChild( root )
```

TC: $O(N)$ SC: $O(1)$

```

Node NextChild (Node root) {
    while (root != NULL)
        if (root.left != NULL) { return root.left }

        if (root.right != NULL) { return root.right }

        root = root.next
    }

    return root
    ↳ probably pt1 NULL
}

```

28) Fill right Pointer for a given *B.T.* *Can be anything*

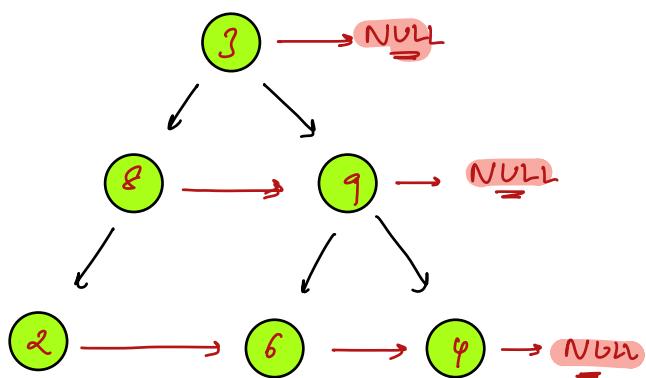
```

class Node {
    int data;
    Node left;
    Node right;
    Node next;

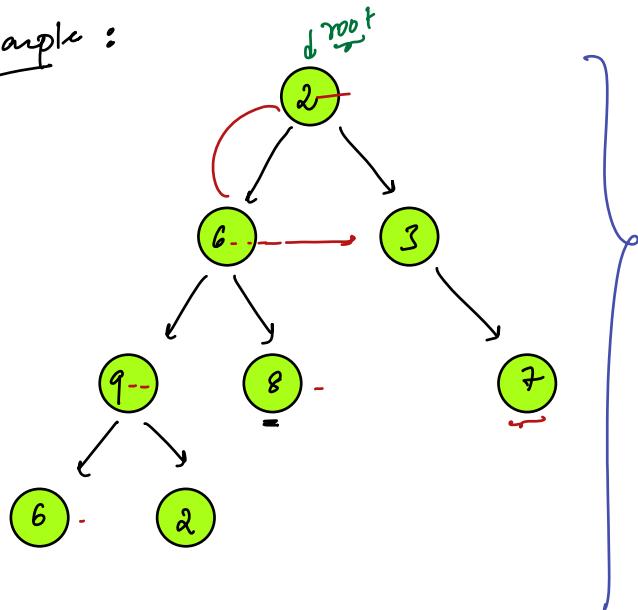
    Node (int n) {
        data = n;
        left = NULL;
        right = NULL;
        next = NULL;
    }
}

```

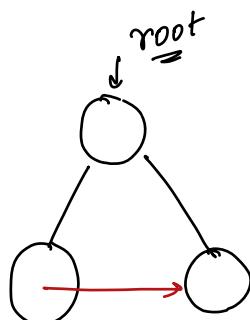
SC: O(1)
TC: O(N)



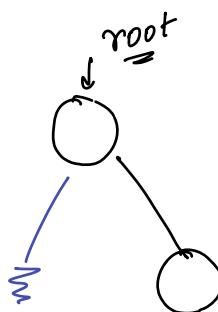
Example :



Idea : fill next pointers level by level

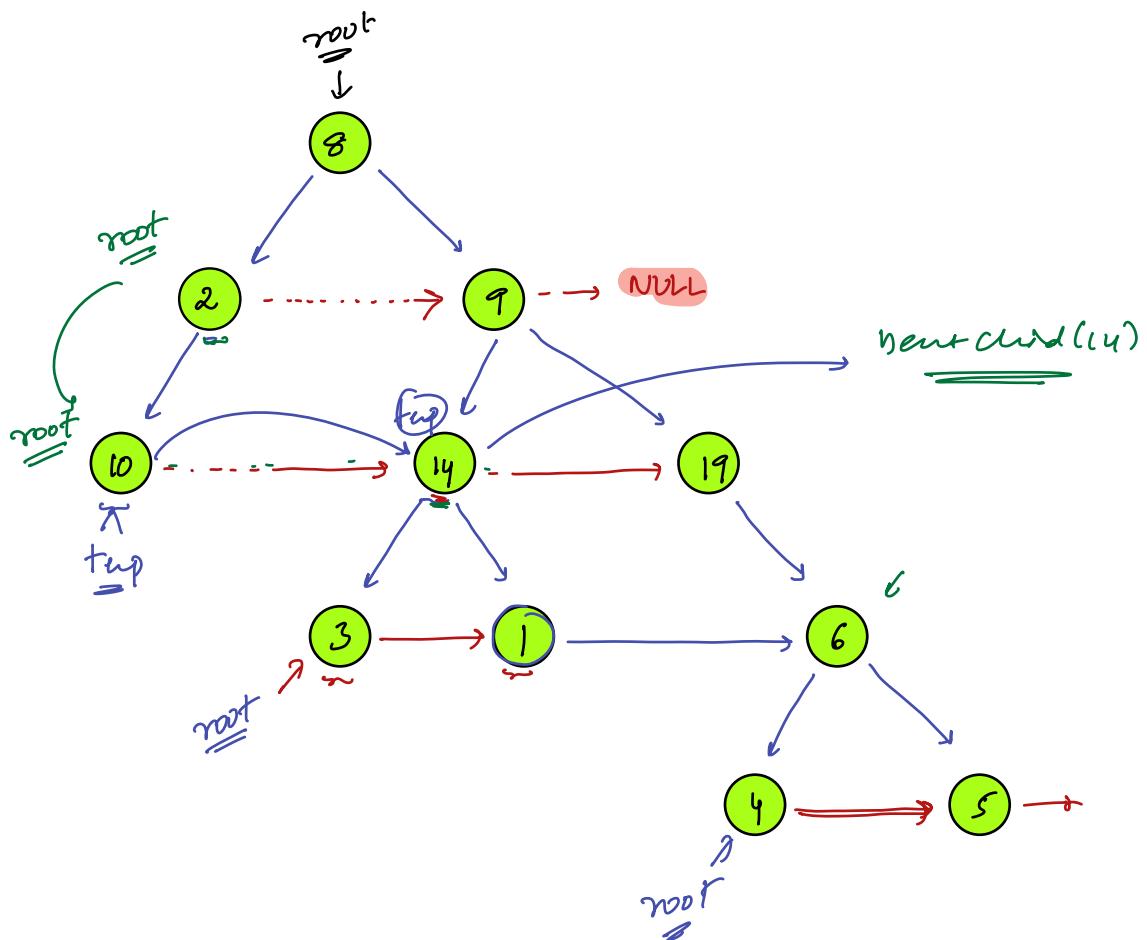


`root.left.next == root.right`

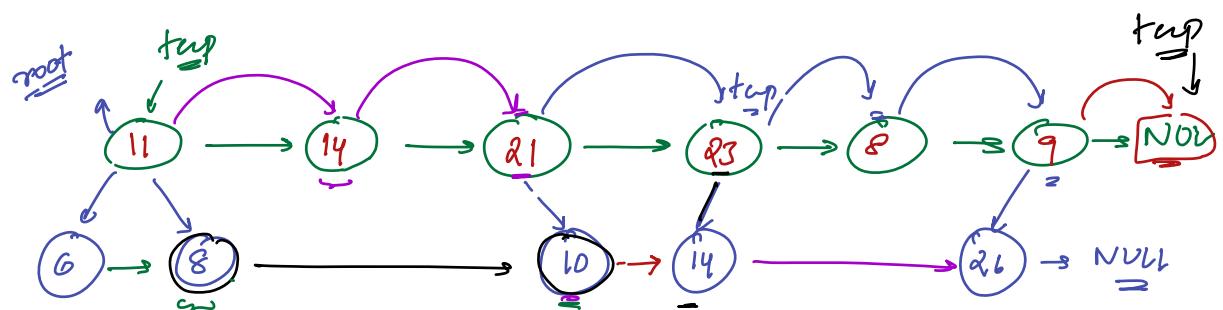


`// root.left.next == root.left`

root = 8



path:



next child (9, null)

next child (null)

{ next child (21, null)

next child (23) = 14

{ next child (23, null)

next child (8) = 26

```
void FillRightPointer( Node* root) {
```

```
    while( root != NULL) {
```

```
        Node* temp = root;
```

```
        while( temp != NULL) {
```

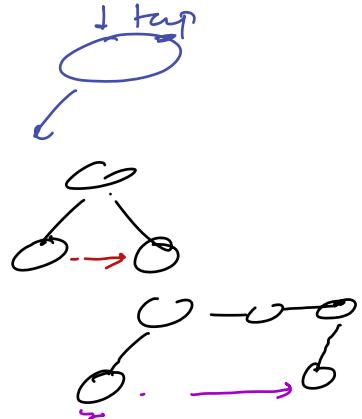
```
            if( temp->left != NULL) {
```

```
                if( temp->right != NULL) {
```

```
                    temp->left->next = temp->right
```

```
                else {
```

```
                    temp->left->next = nextChild(temp->right)
```



```
            if( temp->right != NULL) {
```

```
                temp->right->next = nextChild(temp->right)
```

```
            temp = temp->next
```

```
root = nextChild(root) //
```

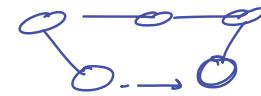
root is going to first
node of new list

TC: $O(n)$

SC: $O(1)$

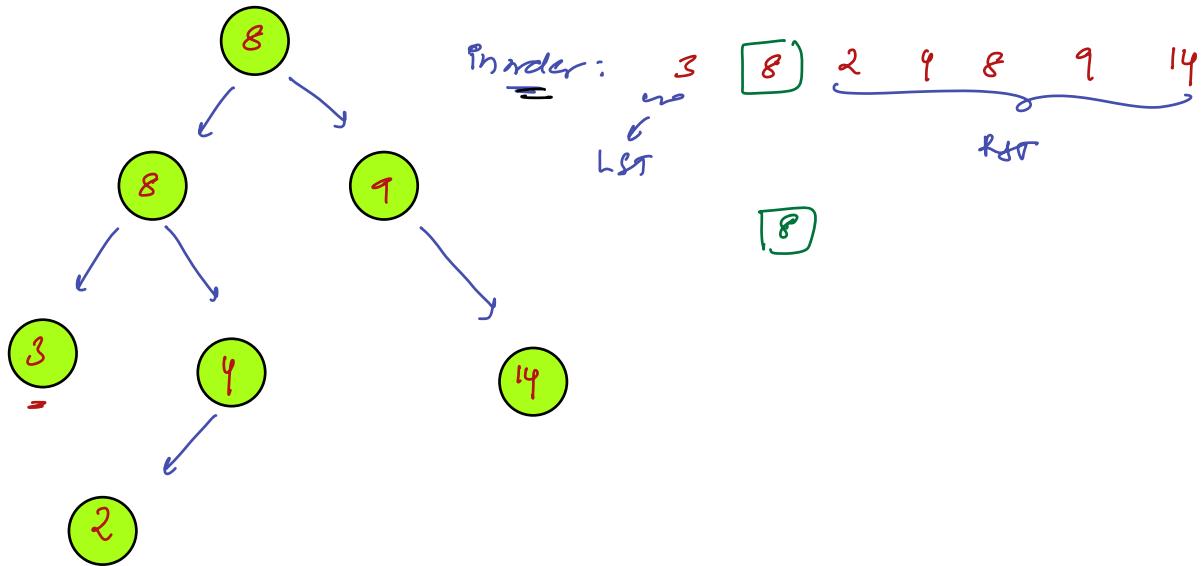
] 10:40pm

breadth-



// duplicacy:

preorder: 8 8 3 4 2 9 14



inorder: 3 8 2 9 8 9 14

LST

RST

[8]

// Serializable:

{ → we need to store in such a manner that
 we need to construct back original Tree

Deserialization

// Serializatim:

preorder

postorder

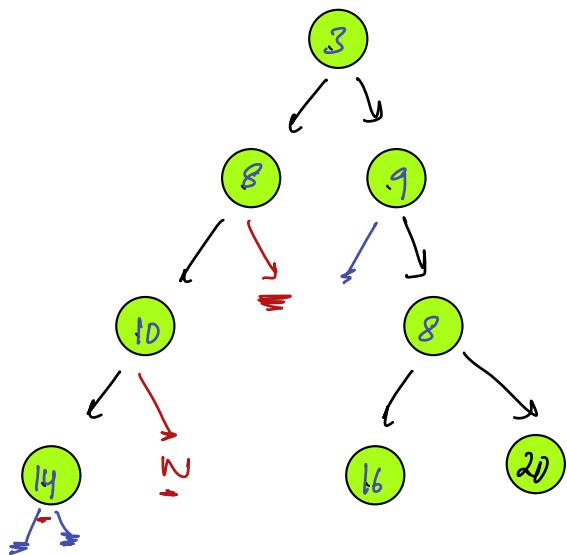
levelorder

(inorder)

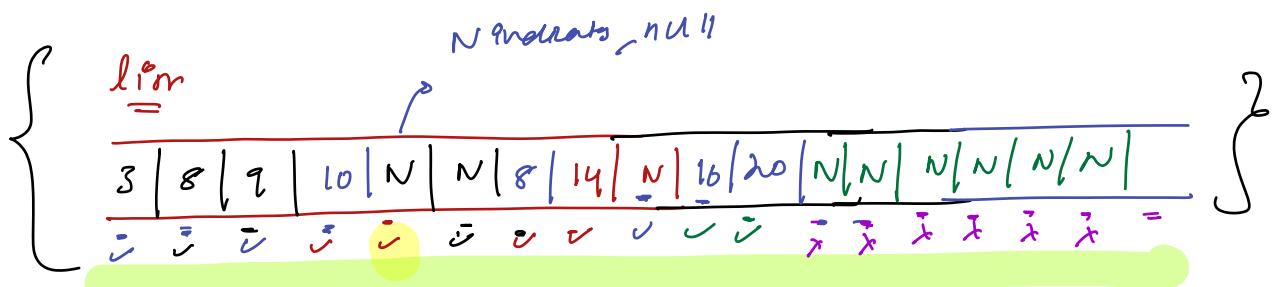
postorder

AisPre

// Serializatim using level order: $\rightarrow [C + \underline{\underline{VC}} \text{ data}]$ (Array + VC data)



Ideas:
 Simply go level by level
 & push child of every
 node.
 //only change: in a list



Serialization(Node root) {

(Prints list; \Rightarrow use queue to store nodes & value in list)
queue of nodes q;
q.push(root);
while (q.size() > 0)
 Node t = q.front(); q.pop();
 if (t != NULL) { // If, add(t.data);
 do { q.push(-1); } $\xrightarrow{-1 \text{ indicates NULL}}$
 if (t != NULL) { // If t == NULL, then can
 q.push(t.left); } $\xrightarrow{\text{we won't push NULL}}$
 q.push(t.right); }
 }
}

return li;

TC: $\Theta(N)$

SC: $\Theta(\max \text{ Nodes in a level}) = \Theta(N)$

$\frac{N}{2}$

Deserialization:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671	672	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687	688	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703	704	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735	736	737	738	739	740	741	742	743	744	745	746	747	748	749	750	751	752	753	754	755	756	757	758	759	760	761	762	763	764	765	766	767	768	769	770	771	772	773	774	775	776	777	778	779	780	781	782	783	784	785	786	787	788	789	790	791	792	793	794	795	796	797	798	799	800	801	802	803	804	805	806	807	808	809	810	811	812	813	814	815	816	817	818	819	820	821	822	823	824	825	826	827	828	829	830	831	832	833	834	835	836	837	838	839	840	841	842	843	844	845	846	847	848	849	850	851	852	853	854	855	856	857	858	859	860	861	862	863	864	865	866	867	868	869	870	871	872	873	874	875	876	877	878	879	880	881	882	883	884	885	886	887	888	889	890	891	892	893	894	895	896	897	898	899	900	901	902	903	904	905	906	907	908	909	910	911	912	913	914	915	916	917	918	919	920	921	922	923	924	925	926	927	928	929	930	931	932	933	934	935	936	937	938	939	940	941	942	943	944	945	946	947	948	949	950	951	952	953	954	955	956	957	958	959	960	961	962	963	964	965	966	967	968	969	970	971	972	973	974	975	976	977	978	979	980	981	982	983	984	985	986	987	988	989	990	991	992	993	994	995	996	997	998	999	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009	10010	10011	10012	10013	10014	10015	10016	10017	10018	10019	10020	10021	10022	10023	10024	10025	10026	10027	10028	10029	10030	10031	10032	10033	10034	10035	10036	10037	10038	10039	10040	10041	10042	10043	10044	10045	10046	10047	10048	10049	10050	10051	10052	10053	10054	10055	10056	10057	10058	10059	10060	10061	10062	10063	10064	10065	10066	10067	10068	10069	10070	10071	10072	10073	10074	10075	10076	10077	10078	10079	10080	10081	10082	10083	10084	10085	10086	10087	10088	10089	10090	10091	10092	10093	10094	10095	10096	10097	10098	10099	100100	100101	100102	100103	100104	100105	100106	100107	100108	100109	100110	100111	100112	100113	100114	100115	100116	100117	100118	100119	100120	100121	100122	100123	100124	100125	100126	100127	100128	100129	100130	100131	100132	100133	100134	100135	100136	100137	100138	100139	100140	100141	100142	100143	100144	100145	100146	100147	100148	100149	100150	100151	100152	100153	100154	100155	100156	100157	100158	100159	100160	100161	100162	100163	100164	100165	100166	100167	100168	100169	100170	100171	100172	100173	100174	100175	100176	100177	100178	100179	100180	100181	100182	100183	100184	100185	100186	100187	100188	100189	100190	100191	100192	100193	100194	100195	100196	100197	100198	100199	100200	100201	100202	100203	100204	100205	100206	100207	100208	100209	100210	100211	100212	100213	100214	100215	100216	100217	100218	100219	100220	100221	100222	100223	100224	100225	100226	100227	100228	100229	100230	100231	100232	100233	100234	100235	100236	100237	100238	100239	100240	100241	100242	100243	100244	100245	100246	100247	100248	100249	100250	100251	100252	100253	100254	100255	100256	100257	100258	100259	100260	100261	100262	100263	100264	100265	100266	100267	100268	100269	100270	100271	100272	100273	100274	100275	100276	100277	100278	100279	100280	100281	100282	100283	100284	100285	100286	100287	100288	100289	100290	100291	100292	100293	100294	100295	100296	100297	100298	100299	100300	100301	100302	100303	100304	100305	100306	100307	100308	100309	100310	100311	100312	100313	100314	100315	1003

root deserialization(int ar[], int n) {

 Node root = new Node(ar[0])

 Queue^{Node} q; q.push(root);

$i = 1;$

 while (q.size() > 0) {

 Node t = q.front();

q.pop();

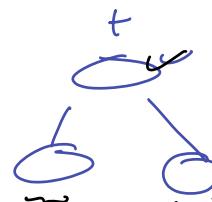
 // t → left child at index i

 → right child at index $i+1$

 if (ar[i] != -1) {

t.left = new node(ar[i])

q.push(t.left)



 if (ar[i+1] != -1) {

t.right = new node(ar[i+1])

q.push(t.right)

$i = i + 2$

 return root;

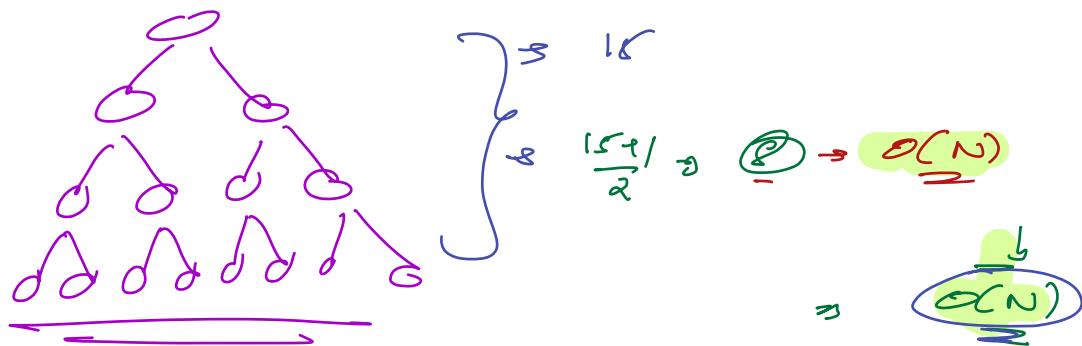
Tc: O(n) sc: O(max node in level)

C Data) \rightarrow push on

↳ number = String \Rightarrow int

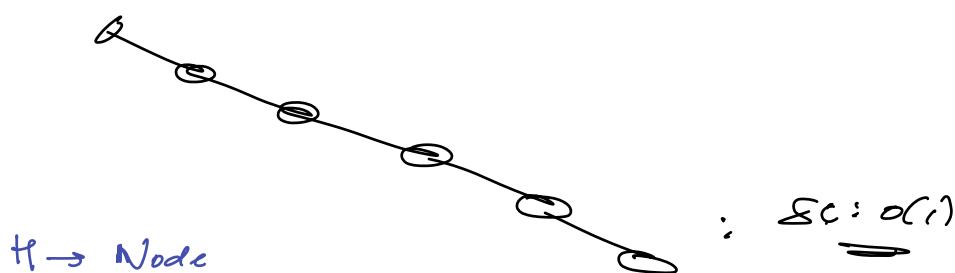
↳ we can use char to indicate tree end is NULL

$\Rightarrow : O(n/2)$



\hookrightarrow (Equal Tree Partition)

\rightarrow Tree: Chase if then enough a subtree with 8n?



$h = N : Skewed \Rightarrow SC: O(\underline{\underline{1}})$

$h = \log(N) : Balance \Rightarrow SC: O(\underline{\underline{N}})$

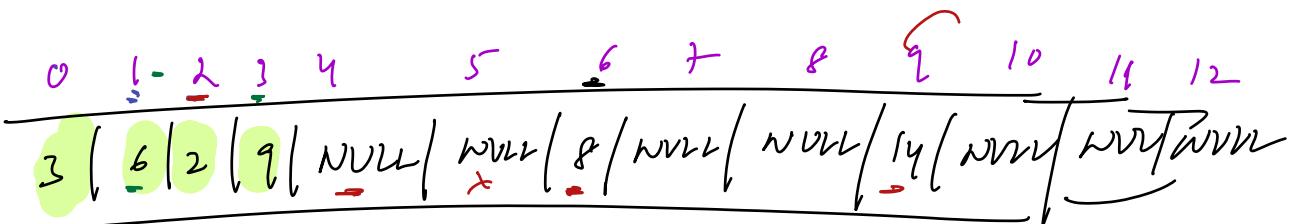
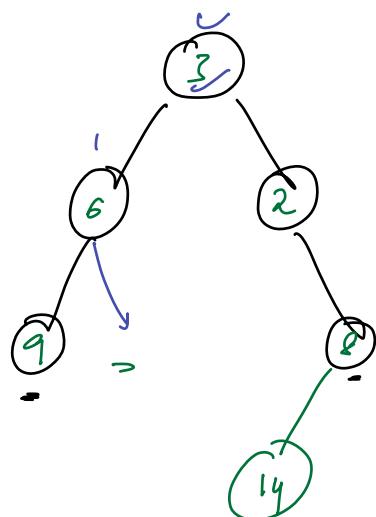
- LCA with no path on
↳ node which return both True is your LCA

Optimal Complexity

CBT → Complete Binary Tree :

→ Nodes level by level
→ nodes / 2 nodes

step 3 → }



Period of String

$\hookrightarrow (N - L_{PS}())$

3

