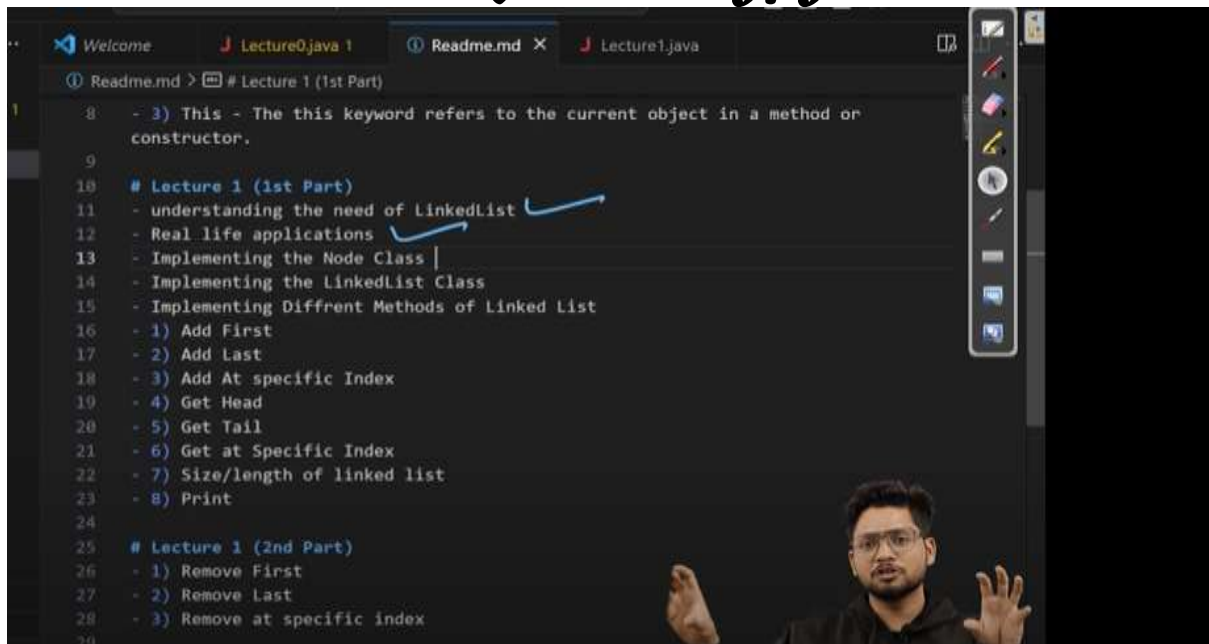
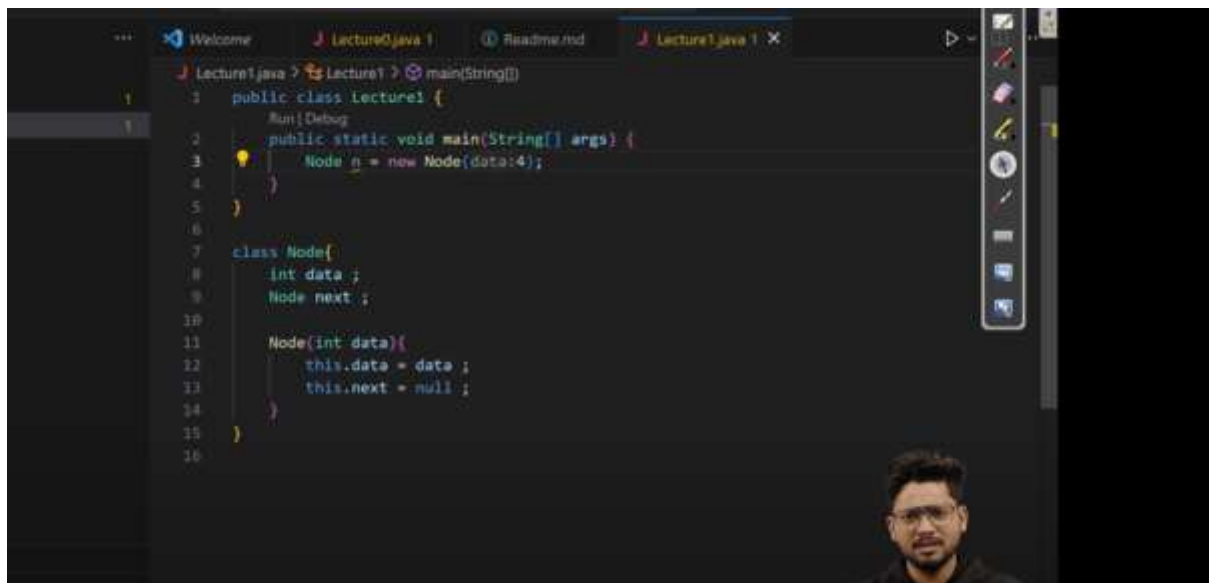


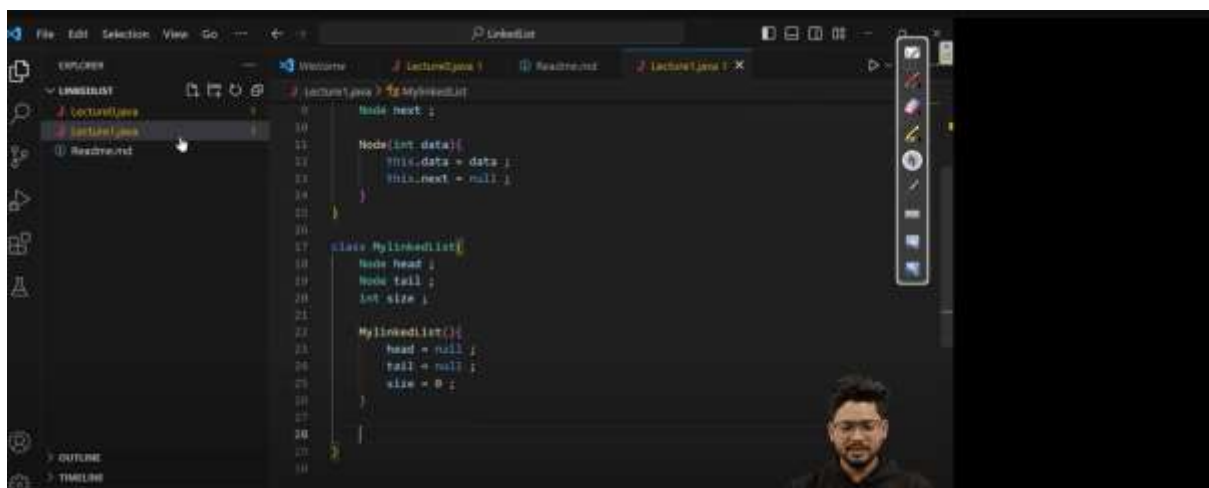
# Linked List(gfg)



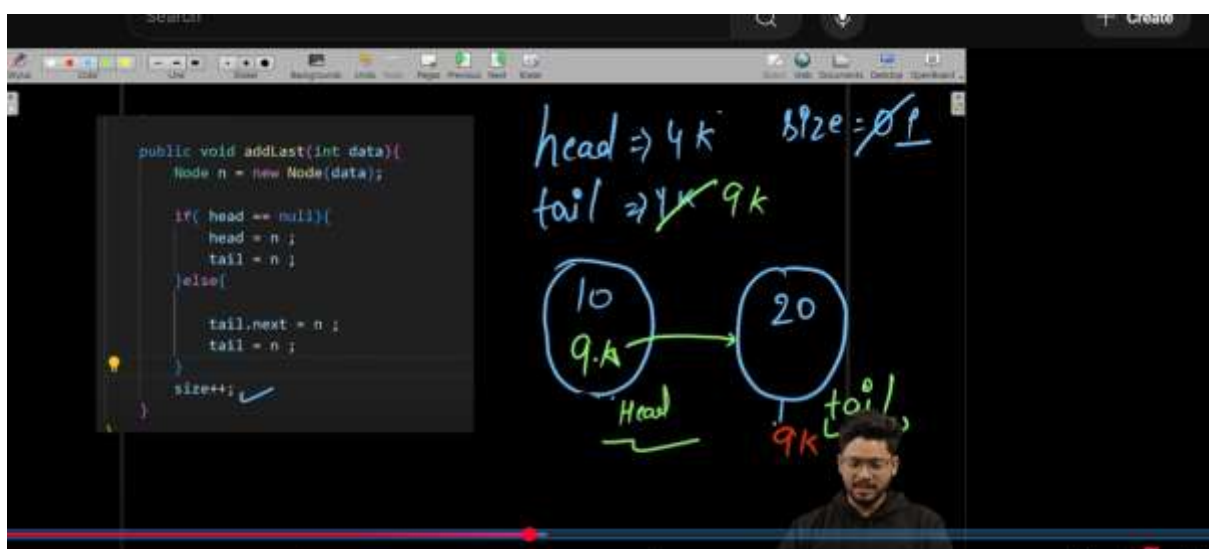
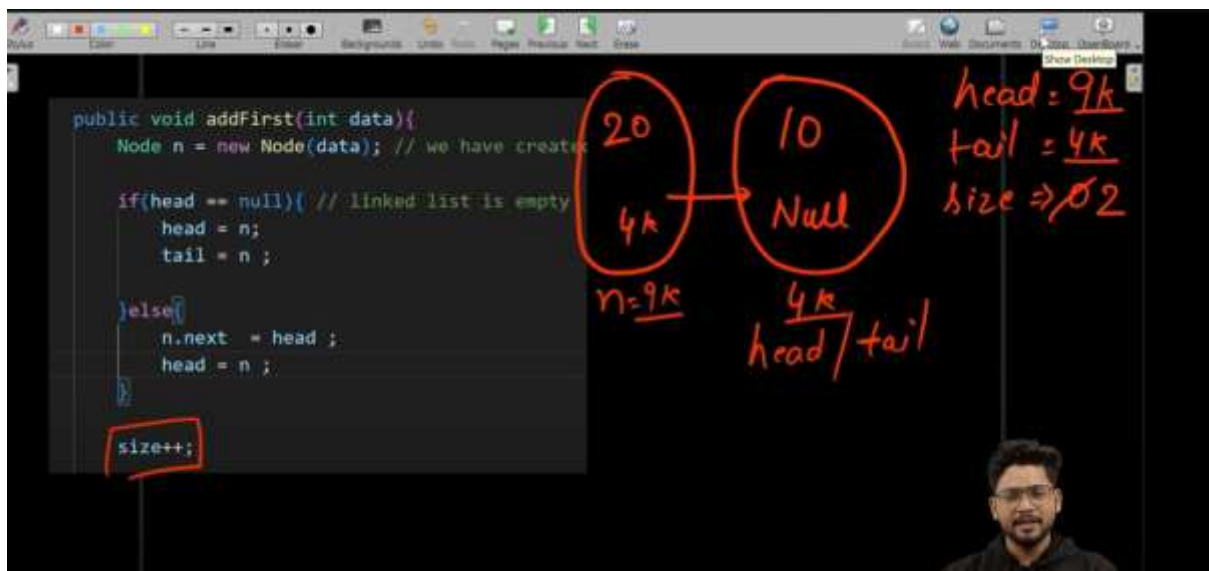
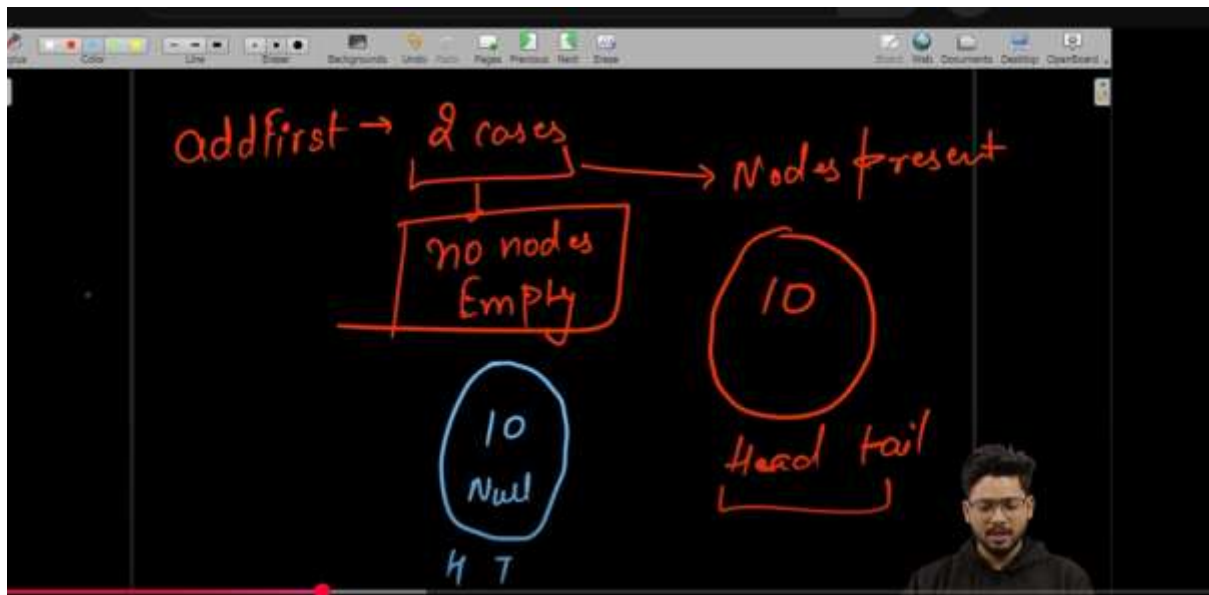
```
1  Welcome  Lecture0.java 1  Readme.md x  Lecture1.java
2  ① Readme.md > # Lecture 1 (1st Part)
3  8 - 3) This - The this keyword refers to the current object in a method or
4  constructor.
5  9
6  10 # Lecture 1 (1st Part)
7  11 - understanding the need of LinkedList
8  12 - Real life applications
9  13 - Implementing the Node Class
10 14 - Implementing the LinkedList Class
11 15 - Implementing Different Methods of Linked List
12 16 - 1) Add First
13 17 - 2) Add Last
14 18 - 3) Add At specific Index
15 19 - 4) Get Head
16 20 - 5) Get Tail
17 21 - 6) Get at Specific Index
18 22 - 7) Size/length of linked list
19 23 - 8) Print
20 24
21 25 # Lecture 1 (2nd Part)
22 26 - 1) Remove First
23 27 - 2) Remove Last
24 28 - 3) Remove at specific index
25 29
```



```
1  Welcome  Lecture0.java 1  Readme.md  Lecture1.java x
2  Lecture1.java > Lecture1 > main(String[])
3  1 public class Lecture1 {
4    2 public static void main(String[] args) {
5      3 Node n = new Node(data:4);
6      4 }
7      5 }
8      6
9      7 class Node{
10     8 int data ;
11     9 Node next ;
12
13     10 Node(int data){
14       11 this.data = data ;
15       12 this.next = null ;
16       13 }
17     14 }
18     15 }
19     16
```



```
1  File Edit Selection View Go  LinkedList
2  EXPLORER  Welcome  Lecture0.java 1  Readme.md  Lecture1.java x
3  1 Lecture1.java > MyLinkedList
4  2 Node next ;
5  3
6  4 Node(int data){
7  5 this.data = data ;
8  6 this.next = null ;
9  7 }
10 8
11 9
12 10
13 11
14 12
15 13
16 14
17 15
18 16
19 17
20 18
21 19
22 20
23 21
24 22
25 23
26 24
27 25
28 26
29 27
30 28
31 29
32 30
33 31
34 32
35 33
36 34
37 35
38 36
39 37
40 38
41 39
42 40
43 41
44 42
45 43
46 44
47 45
48 46
49 47
50 48
51 49
52 50
53 51
54 52
55 53
56 54
57 55
58 56
59 57
60 58
61 59
62 60
63 61
64 62
65 63
66 64
67 65
68 66
69 67
70 68
71 69
72 70
73 71
74 72
75 73
76 74
77 75
78 76
79 77
80 78
81 79
82 80
83 81
84 82
85 83
86 84
87 85
88 86
89 87
90 88
91 89
92 90
93 91
94 92
95 93
96 94
97 95
98 96
99 97
100 98
101 99
102 100
103 101
104 102
105 103
106 104
107 105
108 106
109 107
110 108
111 109
112 110
113 111
114 112
115 113
116 114
117 115
118 116
119 117
120 118
121 119
122 120
123 121
124 122
125 123
126 124
127 125
128 126
129 127
130 128
131 129
132 130
133 131
134 132
135 133
136 134
137 135
138 136
139 137
140 138
141 139
142 140
143 141
144 142
145 143
146 144
147 145
148 146
149 147
150 148
151 149
152 150
153 151
154 152
155 153
156 154
157 155
158 156
159 157
160 158
161 159
162 160
163 161
164 162
165 163
166 164
167 165
168 166
169 167
170 168
171 169
172 170
173 171
174 172
175 173
176 174
177 175
178 176
179 177
180 178
181 179
182 180
183 181
184 182
185 183
186 184
187 185
188 186
189 187
190 188
191 189
192 190
193 191
194 192
195 193
196 194
197 195
198 196
199 197
200 198
201 199
202 200
203 201
204 202
205 203
206 204
207 205
208 206
209 207
210 208
211 209
212 210
213 211
214 212
215 213
216 214
217 215
218 216
219 217
220 218
221 219
222 220
223 221
224 222
225 223
226 224
227 225
228 226
229 227
230 228
231 229
232 230
233 231
234 232
235 233
236 234
237 235
238 236
239 237
240 238
241 239
242 240
243 241
244 242
245 243
246 244
247 245
248 246
249 247
250 248
251 249
252 250
253 251
254 252
255 253
256 254
257 255
258 256
259 257
260 258
261 259
262 260
263 261
264 262
265 263
266 264
267 265
268 266
269 267
270 268
271 269
272 270
273 271
274 272
275 273
276 274
277 275
278 276
279 277
280 278
281 279
282 280
283 281
284 282
285 283
286 284
287 285
288 286
289 287
290 288
291 289
292 290
293 291
294 292
295 293
296 294
297 295
298 296
299 297
300 298
301 299
302 300
303 301
304 302
305 303
306 304
307 305
308 306
309 307
310 308
311 309
312 310
313 311
314 312
315 313
316 314
317 315
318 316
319 317
320 318
321 319
322 320
323 321
324 322
325 323
326 324
327 325
328 326
329 327
330 328
331 329
332 330
333 331
334 332
335 333
336 334
337 335
338 336
339 337
340 338
341 339
342 340
343 341
344 342
345 343
346 344
347 345
348 346
349 347
350 348
351 349
352 350
353 351
354 352
355 353
356 354
357 355
358 356
359 357
360 358
361 359
362 360
363 361
364 362
365 363
366 364
367 365
368 366
369 367
370 368
371 369
372 370
373 371
374 372
375 373
376 374
377 375
378 376
379 377
380 378
381 379
382 380
383 381
384 382
385 383
386 384
387 385
388 386
389 387
390 388
391 389
392 390
393 391
394 392
395 393
396 394
397 395
398 396
399 397
400 398
401 399
402 400
403 401
404 402
405 403
406 404
407 405
408 406
409 407
410 408
411 409
412 410
413 411
414 412
415 413
416 414
417 415
418 416
419 417
420 418
421 419
422 420
423 421
424 422
425 423
426 424
427 425
428 426
429 427
430 428
431 429
432 430
433 431
434 432
435 433
436 434
437 435
438 436
439 437
440 438
441 439
442 440
443 441
444 442
445 443
446 444
447 445
448 446
449 447
450 448
451 449
452 450
453 451
454 452
455 453
456 454
457 455
458 456
459 457
460 458
461 459
462 460
463 461
464 462
465 463
466 464
467 465
468 466
469 467
470 468
471 469
472 470
473 471
474 472
475 473
476 474
477 475
478 476
479 477
480 478
481 479
482 480
483 481
484 482
485 483
486 484
487 485
488 486
489 487
490 488
491 489
492 490
493 491
494 492
495 493
496 494
497 495
498 496
499 497
500 498
501 499
502 500
503 501
504 502
505 503
506 504
507 505
508 506
509 507
510 508
511 509
512 510
513 511
514 512
515 513
516 514
517 515
518 516
519 517
520 518
521 519
522 520
523 521
524 522
525 523
526 524
527 525
528 526
529 527
530 528
531 529
532 530
533 531
534 532
535 533
536 534
537 535
538 536
539 537
540 538
541 539
542 540
543 541
544 542
545 543
546 544
547 545
548 546
549 547
550 548
551 549
552 550
553 551
554 552
555 553
556 554
557 555
558 556
559 557
560 558
561 559
562 560
563 561
564 562
565 563
566 564
567 565
568 566
569 567
570 568
571 569
572 570
573 571
574 572
575 573
576 574
577 575
578 576
579 577
580 578
581 579
582 580
583 581
584 582
585 583
586 584
587 585
588 586
589 587
590 588
591 589
592 590
593 591
594 592
595 593
596 594
597 595
598 596
599 597
600 598
601 599
602 600
603 601
604 602
605 603
606 604
607 605
608 606
609 607
610 608
611 609
612 610
613 611
614 612
615 613
616 614
617 615
618 616
619 617
620 618
621 619
622 620
623 621
624 622
625 623
626 624
627 625
628 626
629 627
630 628
631 629
632 630
633 631
634 632
635 633
636 634
637 635
638 636
639 637
640 638
641 639
642 640
643 641
644 642
645 643
646 644
647 645
648 646
649 647
650 648
651 649
652 650
653 651
654 652
655 653
656 654
657 655
658 656
659 657
660 658
661 659
662 660
663 661
664 662
665 663
666 664
667 665
668 666
669 667
670 668
671 669
672 670
673 671
674 672
675 673
676 674
677 675
678 676
679 677
680 678
681 679
682 680
683 681
684 682
685 683
686 684
687 685
688 686
689 687
690 688
691 689
692 690
693 691
694 692
695 693
696 694
697 695
698 696
699 697
700 698
701 699
702 700
703 701
704 702
705 703
706 704
707 705
708 706
709 707
710 708
711 709
712 710
713 711
714 712
715 713
716 714
717 715
718 716
719 717
720 718
721 719
722 720
723 721
724 722
725 723
726 724
727 725
728 726
729 727
730 728
731 729
732 730
733 731
734 732
735 733
736 734
737 735
738 736
739 737
740 738
741 739
742 740
743 741
744 742
745 743
746 744
747 745
748 746
749 747
750 748
751 749
752 750
753 751
754 752
755 753
756 754
757 755
758 756
759 757
760 758
761 759
762 760
763 761
764 762
765 763
766 764
767 765
768 766
769 767
770 768
771 769
772 770
773 771
774 772
775 773
776 774
777 775
778 776
779 777
780 778
781 779
782 780
783 781
784 782
785 783
786 784
787 785
788 786
789 787
790 788
791 789
792 790
793 791
794 792
795 793
796 794
797 795
798 796
799 797
800 798
801 799
802 800
803 801
804 802
805 803
806 804
807 805
808 806
809 807
810 808
811 809
812 810
813 811
814 812
815 813
816 814
817 815
818 816
819 817
820 818
821 819
822 820
823 821
824 822
825 823
826 824
827 825
828 826
829 827
830 828
831 829
832 830
833 831
834 832
835 833
836 834
837 835
838 836
839 837
840 838
841 839
842 840
843 841
844 842
845 843
846 844
847 845
848 846
849 847
850 848
851 849
852 850
853 851
854 852
855 853
856 854
857 855
858 856
859 857
860 858
861 859
862 860
863 861
864 862
865 863
866 864
867 865
868 866
869 867
870 868
871 869
872 870
873 871
874 872
875 873
876 874
877 875
878 876
879 877
880 878
881 879
882 880
883 881
884 882
885 883
886 884
887 885
888 886
889 887
890 888
891 889
892 890
893 891
894 892
895 893
896 894
897 895
898 896
899 897
900 898
901 899
902 900
903 901
904 902
905 903
906 904
907 905
908 906
909 907
910 908
911 909
912 910
913 911
914 912
915 913
916 914
917 915
918 916
919 917
920 918
921 919
922 920
923 921
924 922
925 923
926 924
927 925
928 926
929 927
930 928
931 929
932 930
933 931
934 932
935 933
936 934
937 935
938 936
939 937
940 938
941 939
942 940
943 941
944 942
945 943
946 944
947 945
948 946
949 947
950 948
951 949
952 950
953 951
954 952
955 953
956 954
957 955
958 956
959 957
960 958
961 959
962 960
963 961
964 962
965 963
966 964
967 965
968 966
969 967
970 968
971 969
972 970
973 971
974 972
975 973
976 974
977 975
978 976
979 977
980 978
981 979
982 980
983 981
984 982
985 983
986 984
987 985
988 986
989 987
990 988
991 989
992 990
993 991
994 992
995 993
996 994
997 995
998 996
999 997
1000 998
1001 999
1002 1000
1003 1001
1004 1002
1005 1003
1006 1004
1007 1005
1008 1006
1009 1007
1010 1008
1011 1009
1012 1010
1013 1011
1014 1012
1015 1013
1016 1014
1017 1015
1018 1016
1019 1017
1020 1018
1021 1019
1022 1020
1023 1021
1024 1022
1025 1023
1026 1024
1027 1025
1028 1026
1029 1027
1030 1028
1031 1029
1032 1030
1033 1031
1034 1032
1035 1033
1036 1034
1037 1035
1038 1036
1039 1037
1040 1038
1041 1039
1042 1040
1043 1041
1044 1042
1045 1043
1046 1044
1047 1045
1048 1046
1049 1047
1050 1048
1051 1049
1052 1050
1053 1051
1054 1052
1055 1053
1056 1054
1057 1055
1058 1056
1059 1057
1060 1058
1061 1059
1062 1060
1063 1061
1064 1062
1065 1063
1066 1064
1067 1065
1068 1066
1069 1067
1070 1068
1071 1069
1072 1070
1073 1071
1074 1072
1075 1073
1076 1074
1077 1075
1078 1076
1079 1077
1080 1078
1081 1079
1082 1080
1083 1081
1084 1082
1085 1083
1086 1084
1087 1085
1088 1086
1089 1087
1090 1088
1091 1089
1092 1090
1093 1091
1094 1092
1095 1093
1096 1094
1097 1095
1098 1096
1099 1097
1100 1098
1101 1099
1102 1100
1103 1101
1104 1102
1105 1103
1106 1104
1107 1105
1108 1106
1109 1107
1110 1108
1111 1109
1112 1110
1113 1111
1114 1112
1115 1113
1116 1114
1117 1115
1118 1116
1119 1117
1120 1118
1121 1119
1122 1120
1123 1121
1124 1122
1125 1123
1126 1124
1127 1125
1128 1126
1129 1127
1130 1128
1131 1129
1132 1130
1133 1131
1134 1132
1135 1133
1136 1134
1137 1135
1138 1136
1139 1137
1140 1138
1141 1139
1142 1140
1143 1141
1144 1142
1145 1143
1146 1144
1147 1145
1148 1146
1149 1147
1150 1148
1151 1149
1152 1150
1153 1151
1154 1152
1155 1153
1156 1154
1157 1155
1158 1156
1159 1157
1160 1158
1161 1159
1162 1160
1163 1161
1164 1162
1165 1163
1166 1164
1167 1165
1168 1166
1169 1167
1170 1168
1171 1169
1172 1170
1173 1171
1174 1172
1175 1173
1176 1174
1177 1175
1178 1176
1179 1177
1180 1178
1181 1179
1182 1180
1183 1181
1184 1182
1185 1183
1186 1184
1187 1185
1188 1186
1189 1187
1190 1188
1191 1189
1192 1190
1193 1191
1194 1192
1195 1193
1196 1194
1197 1195
1198 1196
1199 1197
1200 1198
1201 1199
1202 1200
1203 1201
1204 1202
1205 1203
1206 1204
1207 1205
1208 1206
1209 1207
1210 1208
1211 1209
1212 1210
1213 1211
1214 1212
1215 1213
1216 1214
1217 1215
1218 1216
1219 1217
1220 1218
1221 1219
1222 1220
1223 1221
1224 1222
1225 1223
1226 1224
1227 1225
1228 1226
1229 1227
1230 1228
1231 1229
1232 1230
1233 1231
1234 1232
1235 1233
1236 1234
1237 1235
1238 1236
1239 1237
1240 1238
1241 1239
1242 1240
1243 1241
1244 1242
1245 1243
1246 1244
1247 1245
1248 1246
1249 1247
1250 1248
1251 1249
1252 1250
1253 1251
1254 1252
1255 1253
1256 1254
1257 1255
1258 1256
1259 1257
1260 1258
1261 1259
1262 1260
1263 1261
1264 1262
1265 1263
1266 1264
1267 1265
1268 1266
1269 1267
1270 1268
1271 1269
1272 1270
1273 1271
1274 1272
1275 1273
1276 1274
1277 1275
1278 1276
1279 1277
1280 1278
1281 1279
1282 1280
1283 1281
1284 1282
1285 1283
1286 1284
1287 1285
1288 1286
1289 1287
1290 1288
1291 1289
1292 1290
1293 1291
1294 1292
1295 1293
1296 1294
1297 1295
1298 1296
1299 1297
1300 1298
1301 1299
1302 1300
1303 1301
1304 1302
1305 1303
1306 1304
1307 1305
1308 1306
1309 1307
1310 1308
1311 1309
1312 1310
1313 1311
1314 1312
1315 1313
1316 1314
1317 1315
1318 1316
1319 1317
1320 1318
1321 1319
1322 1320
1323 1321
1324 1322
1325 1323
1326 1324
1327 1325
1328 1326
1329 1327
1330 1328
1331 1329
1332 1330
1333 1331
1334 1332
1335 1333
1336 1334
1337 1335
1338 1336
1339 1337
1340 1338
1341 1339
1342 1340
1343 1341
1344 1342
1345 1343
1346 1344
1347 1345
1348 1346
1349 1347
1350 1348
1351 1349
1352 1350
1353 1351
1354 1352
1355 1353
1356 1354
1357 1355
1358 1356
1359 1357
1360 1358
1361 1359
1362 1360
1363 1361
1364 1362
1365 1363
1366 1364
1367 1365
1368 1366
1369 1367
137
```



```

Code 1 Code 1 +
1 public class Lecture1 {
2     public static void main(String[] args) {
3         MyLinkedList list = new MyLinkedList();
4
5         list.addFirst(10);
6         list.addFirst(20);
7         list.addLast(100);
8         list.addLast(190);
9
10        list.addAtIndex(110, 2);
11
12        list.addAtIndex(99, 1);
13
14        System.out.println(" before deletion " + list);
15
16        System.out.println(list.removeAtIndex(2));
17        System.out.println(list.removeAtIndex(3));
18
19        System.out.println("After deletion " + list);
20    }
21 }
22
23 class Node {
24     int data;
25     Node next;

```

```

public void addAtIndex(int data, int idx) {
    if (idx < 0 || idx > size) {
        System.out.println(" Index is not valid ");
        return;
    } else if (idx == 0) {
        addFirst(data);
    } else if (idx == size) {
        addLast(data);
    } else {
        Node n = new Node(data);
        Node pre = head;
        while (idx > 0) {
            pre = pre.next;
            idx--;
        }
        Node nbr = pre.next;
        pre.next = n;
        n.next = nbr;
        size++;
    }
}

```

```

// public int addLast(int data) {
// }
public int getFirst() {
    if (head == null) {
        System.out.println("LinkedList is empty");
        return -1;
    } else {
        return head.data;
    }
}
public int getLast() {
    if (tail == null) {
        System.out.println("LinkedList is empty");
        return -1;
    } else {
        return tail.data;
    }
}
public int getAtIndex(int idx) {
    if (idx < 0 || idx > size) {

```

```

public int getAtIndex(int idx) {
    if (idx < 0 || idx > size) {
        System.out.println("Invalid index ");
        return -1;
    } else if (idx == 0) {
        return getFirst();
    } else if (idx == size - 1) {
        return getLast();
    } else {
        Node curr = head;
        while (idx > 0) {
            curr = curr.next;
            idx--;
        }
        return curr.data;
    }
}

```

```

142 public int removeFirst() {
143     if (head == null) {
144         System.out.println("LinkedList is empty");
145         return -1;
146     } else if (head.next == null) {
147         int data = head.data;
148         head = null;
149         tail = null;
150         size--;
151         return data;
152     } else {
153         int data = head.data;
154         head = head.next;
155         size--;
156         return data;
157     }
158 }
159
160 public int removeLast() {
161     if (head == null) { // empty LL
162         System.out.println("LinkedList is empty");
163         return -1;
164     } else if (head.next == null) { // one Node in LL
165         int data = head.data;
166         head = null;
167         tail = null;
168         size--;
169         return data;
170     } else {
171         Node curr = head;
172         int data = tail.data;
173         while (curr.next != tail) {
174             curr = curr.next;
175         }
176         curr.next = null;
177         size--;
178         tail = curr;
179         return data;
180     }
181 }
182
183 }

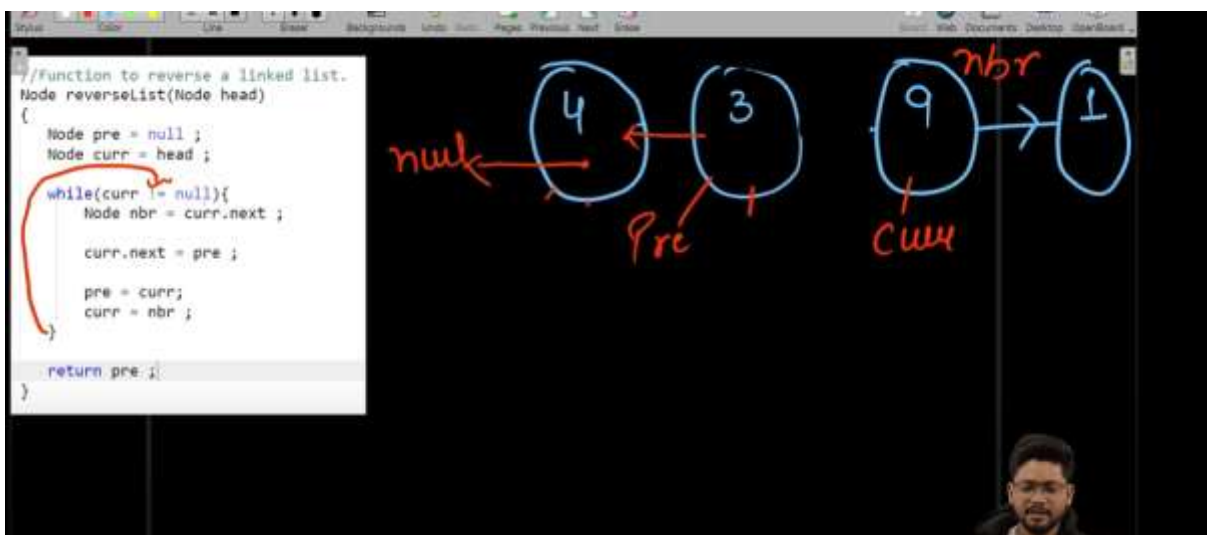
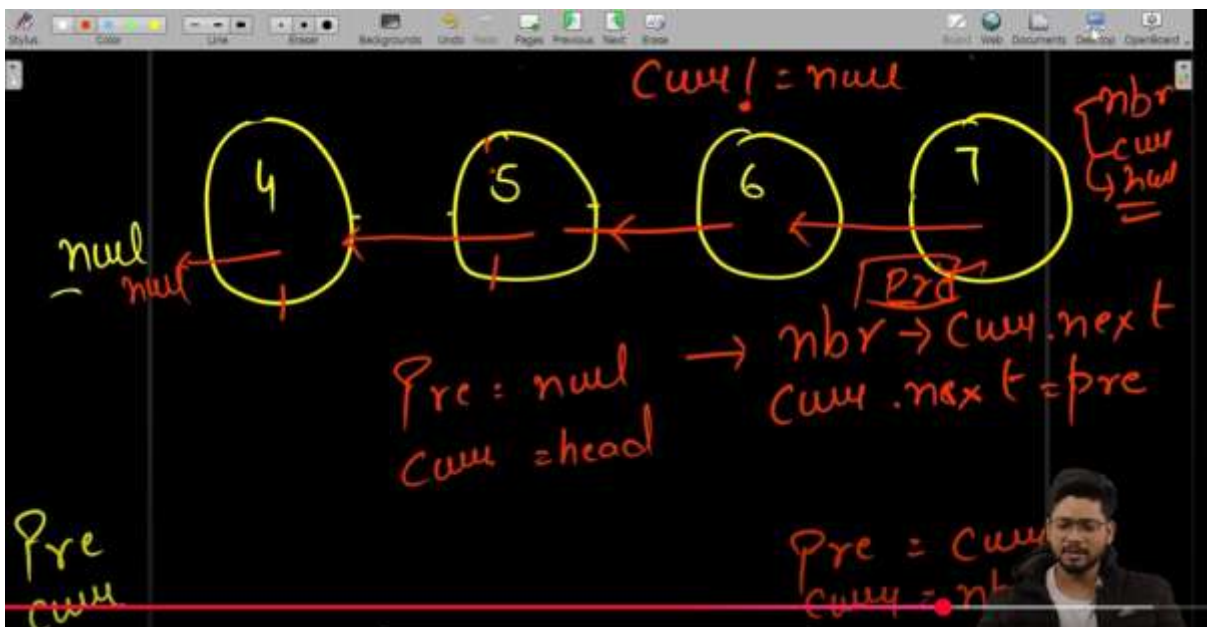
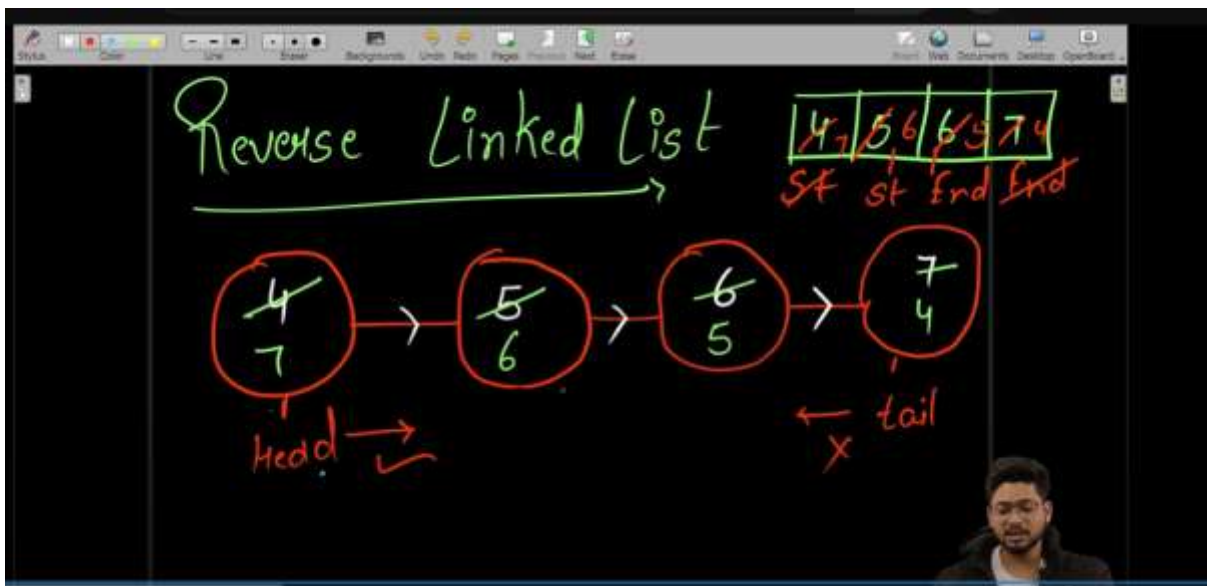
```

```

196
197 public int removeAtSpecificIndex(int idx){
198
199     if ( idx<0 || idx>= size ){
200         System.out.println("Invalid idx");
201         return -1 ;
202     }else if ( idx == 0 ){
203         return removeFirst();
204     }else if ( idx == size-1 ){
205         return removeLast();
206     }else {
207         Node curr = head ;
208
209         while( idx-1 > 0){
210             curr = curr.next ;
211             idx-- ;
212         }
213         int data = curr.next.data ;
214         curr.next = curr.next.next ;
215
216         size-- ;
217         return data ;
218     }
219 }
220
221
222
223 public String toString() {
224     String str = "";
225     Node curr = head;
226     while (curr != null) {
227         str = str + curr.data + " ";
228         curr = curr.next;
229     }
230     return str;
231 }
232
233
234
235
236
237
238
239 public int length() {
240     return size;

```

## 1. Reverse LinkedList



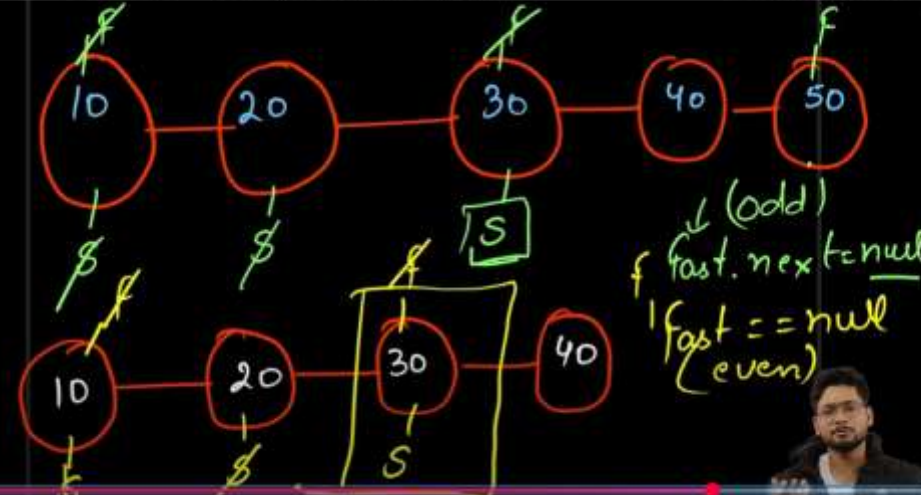

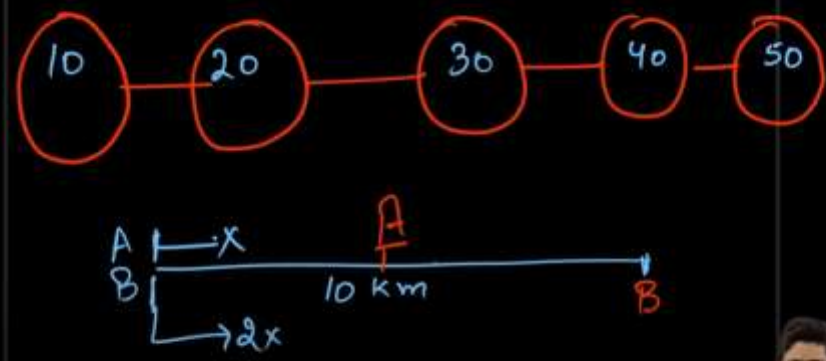



## 2.Middle Element in LinkedList


### Mid of Linked List

<Type Text Here>

- 1) Empty LinkedList
- 2) One Node LinkedList
- 3) Odd Number of nodes in LinkedList
- 4) Even Number of Nodes in LinkedList



(odd)  
fast.next = null  
fast == null  
(even)

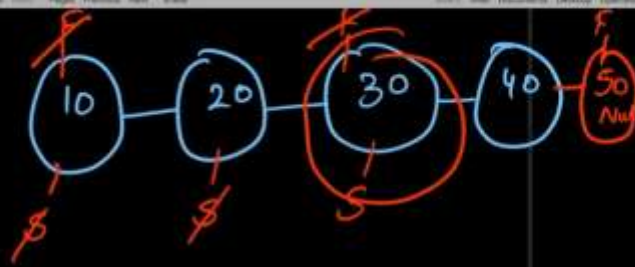


```
int getMiddle(Node head)
{
    if( head == null){
        return -1 ;
    }else if ( head.next == null){
        return head.data ;
    }

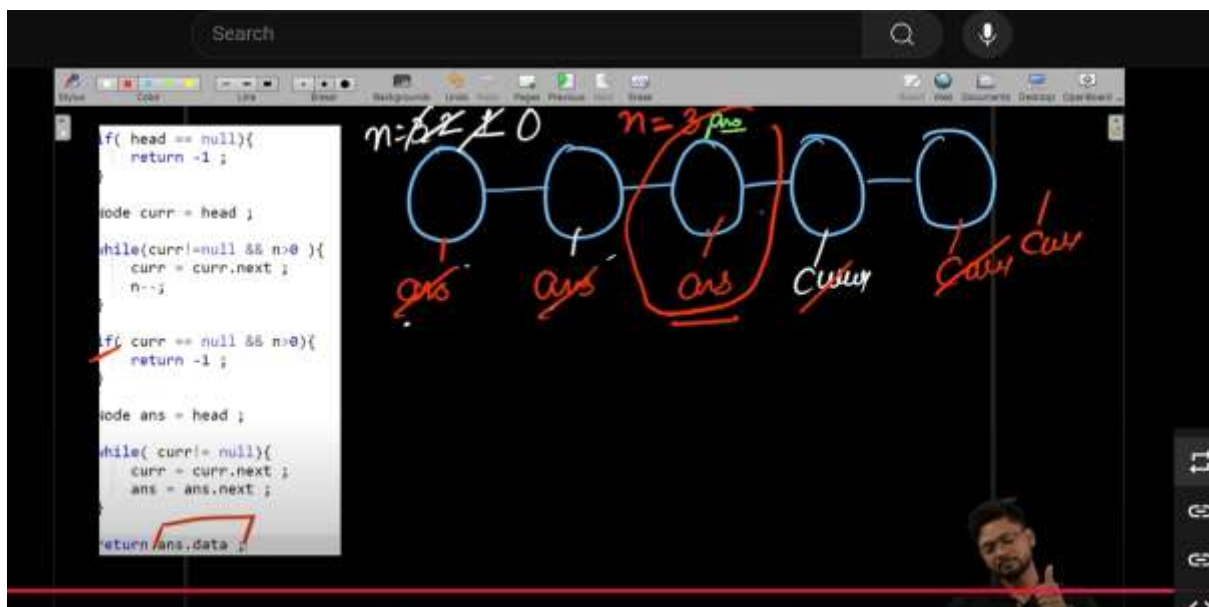
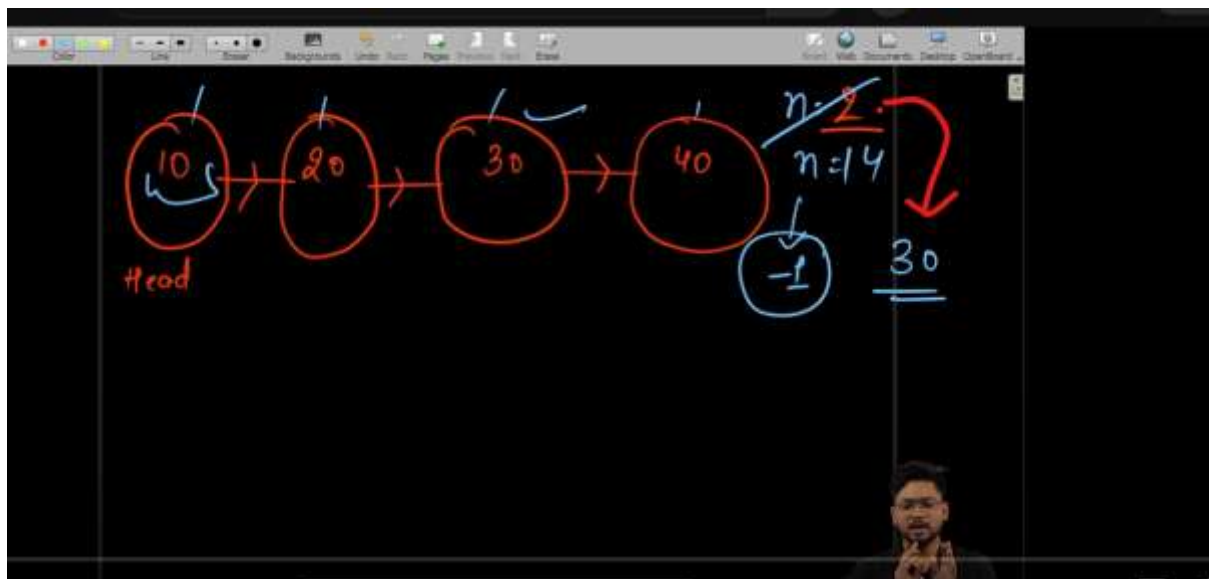
    Node fast = head ;
    Node slow = head ;

    while(fast!=null && fast.next!=null){
        fast = fast.next.next ;
        slow = slow.next ;
    }

    return slow.data ;
}
```

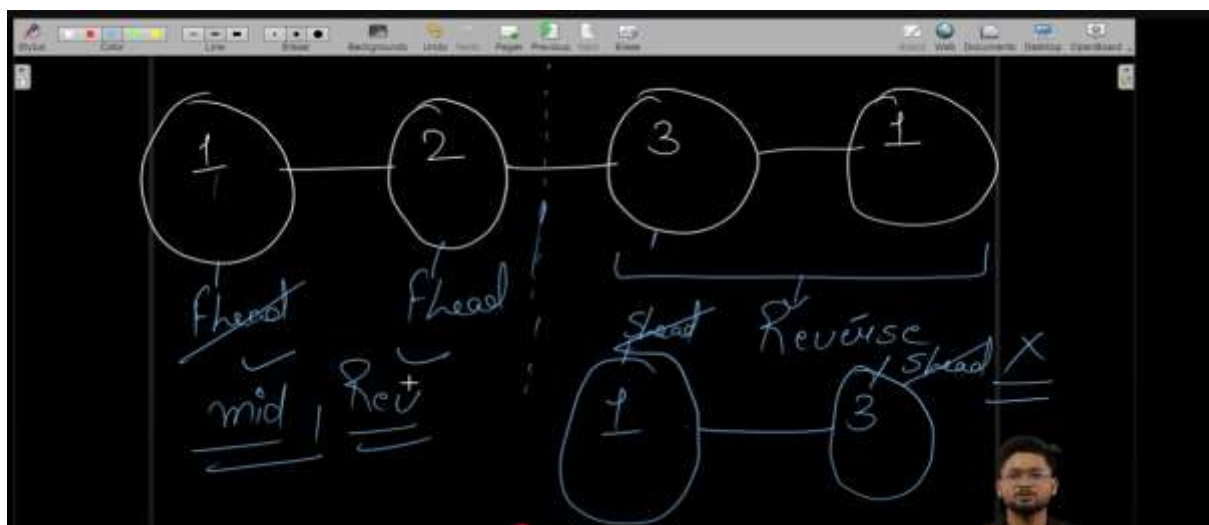
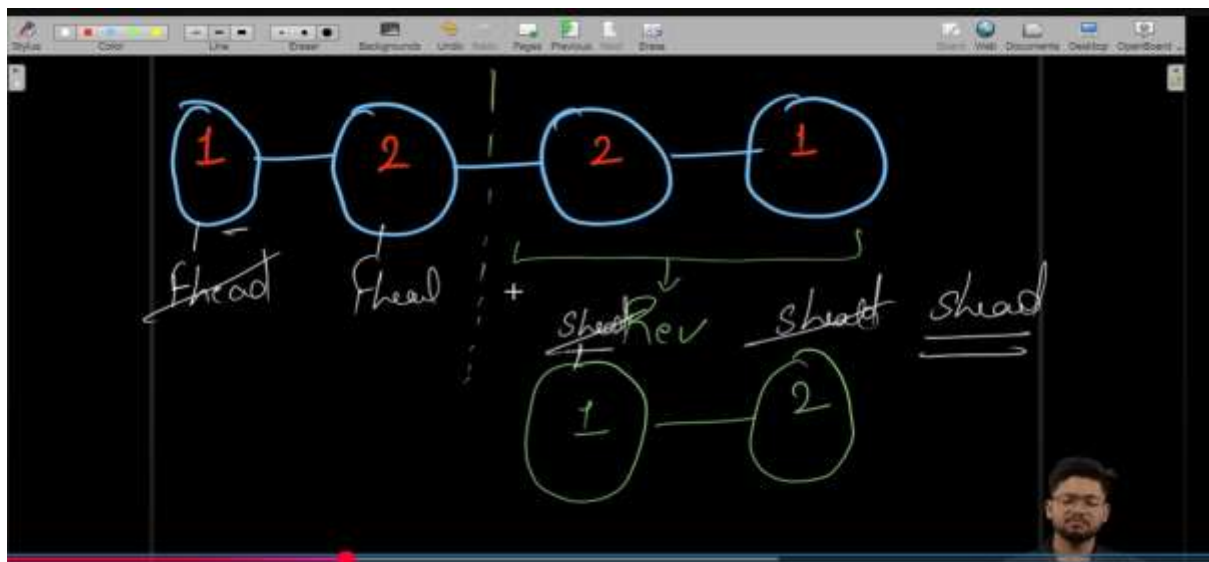
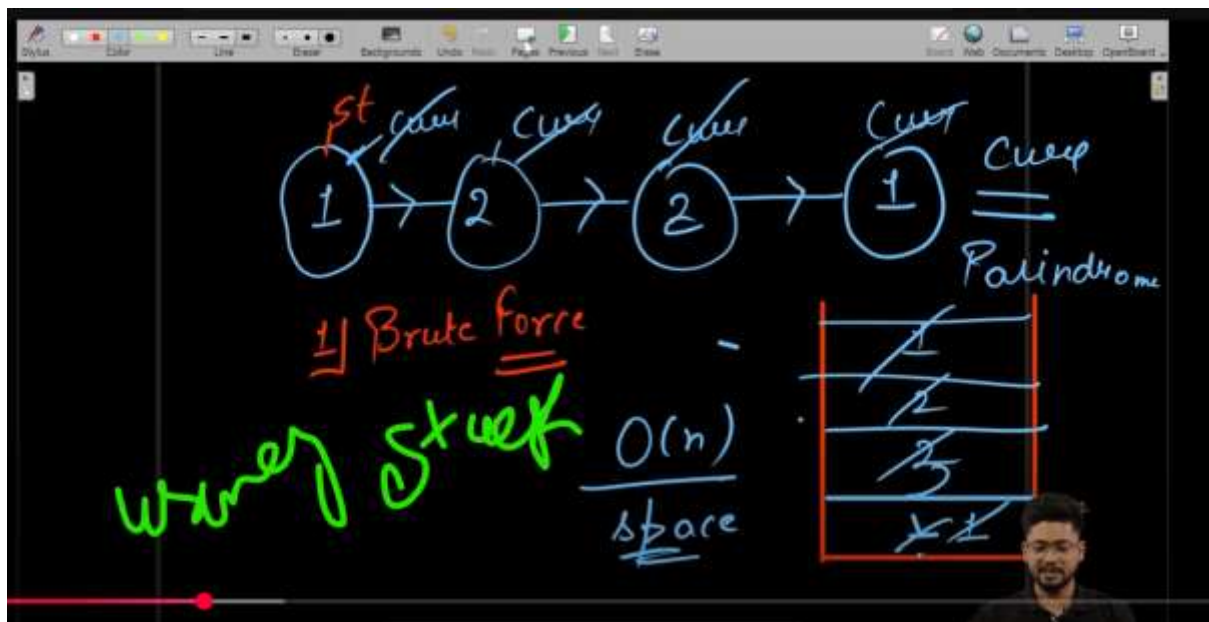


### 3.Nth Node from End of LinkedList





#### 4. Check if linked list is palindrome



```

boolean isPalindrome(Node head)
{
    if( head== null || head.next == null ){
        return true ;
    }

    Node mid = middle(head);

    Node shhead = reverse(mid); //it is the head
    Node fhead = head ; // it is the first part

    while( shhead!=null){
        if(shhead.data != fhead.data){
            return false ;
        }

        fhead = fhead.next ;
        shhead = shhead.next ;
    }

    return true ;
}

```

if the given linked list is palindrome or not.

**Example 1:**

Input:  
N = 3  
value[] = {1,2,1}

Output: 1

Explanation: The given linked list is 1 2 1, which is a palindrome and Hence, the output is 1.

**Example 2:**

Input:  
N = 4  
value[] = {1,2,3,4}

```

86 class Solution
87 {
88     Node middle(Node head ){
89         if( head == null){
90             return head ;
91         }
92
93         Node fast = head ;
94         Node slow = head ;
95
96         while( fast!=null && fast.next!= null){
97             fast = fast.next.next ;
98             slow = slow.next;
99         }
100
101         return slow ;
102     }
103
104     boolean isPalindrome(Node head)
105     {
106
107

```

if the given linked list is palindrome or not.

**Example 1:**

Input:  
N = 3  
value[] = {1,2,1}

Output: 1

Explanation: The given linked list is 1 2 1, which is a palindrome and Hence, the output is 1.

**Example 2:**

Input:  
N = 4  
value[] = {1,2,3,4}

Output: 0

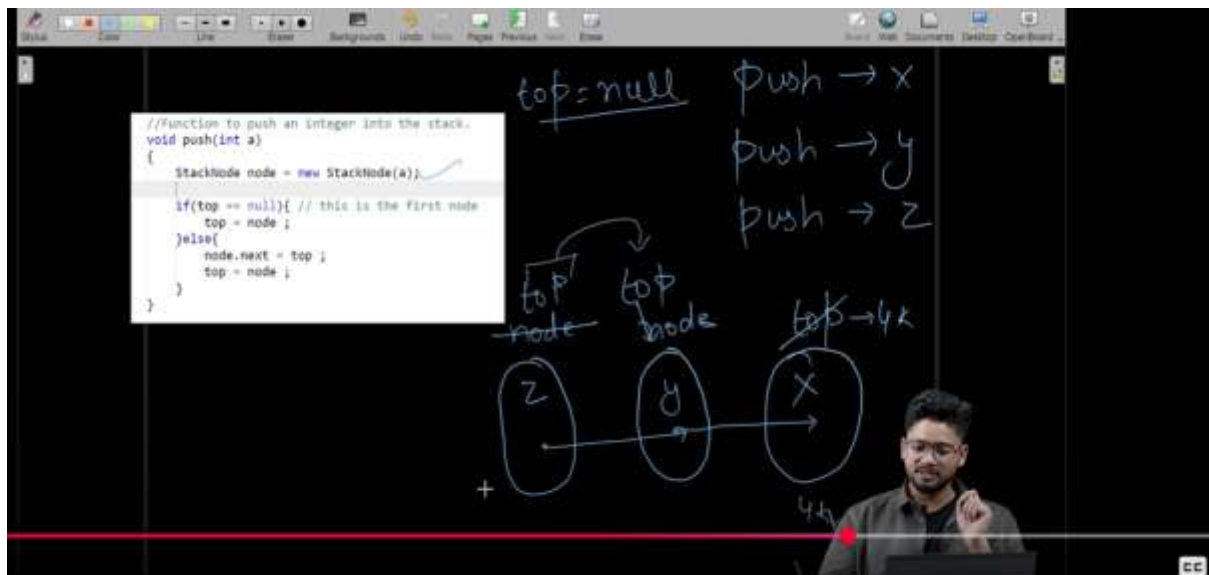
Explanation: The given linked list

```

184     Node reverse( Node head ){
185         if( head == null || head.next == null){
186             return head ;
187         }
188
189         Node curr = head ;
190         Node pre = null;
191
192         while( curr != null){
193             Node nbr = curr.next ;
194
195             curr.next = pre ;
196
197             pre = curr ;
198             curr = nbr ;
199         }
200
201         return pre ;
202     }
203
204     boolean isPalindrome(Node head)
205     {
206
207

```

## 5. Implement stack using Linked list



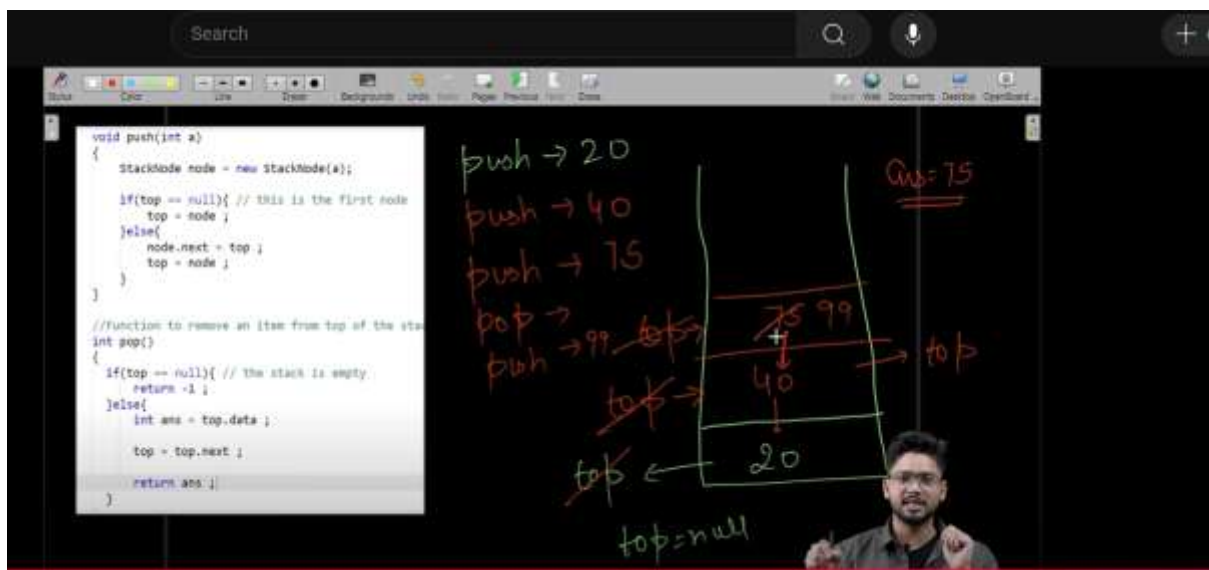
```
//Function to push an integer into the stack.
void push(int a)
{
    StackNode node = new StackNode(a);
    if(top == null){ // this is the first node
        top = node;
    }else{
        node.next = top;
        top = node;
    }
}
```

Handwritten notes:

- $top = null$
- push  $\rightarrow x$
- push  $\rightarrow y$
- push  $\rightarrow z$

Diagram of the linked list structure:

- Nodes:  $z \rightarrow y \rightarrow x$
- $top$  points to the first node (z).



```
void push(int a)
{
    StackNode node = new StackNode(a);
    if(top == null){ // this is the first node
        top = node;
    }else{
        node.next = top;
        top = node;
    }
}

//Function to remove an item from top of the stack
int pop()
{
    if(top == null){ // the stack is empty
        return -1;
    }else{
        int ans = top.data;
        top = top.next;
        return ans;
    }
}
```

Handwritten notes:

- push  $\rightarrow 20$
- push  $\rightarrow 40$
- push  $\rightarrow 75$
- pop  $\rightarrow 75$
- pop  $\rightarrow 40$
- pop  $\rightarrow 20$
- $top = null$

Diagram of the linked list structure:

- Nodes:  $20 \rightarrow 40 \rightarrow 75$
- $top$  points to the first node (20).

## 6. Implement Queue using LinkedList

Handwritten notes on the left side of the board:

- push  $\rightarrow$  10
- push  $\rightarrow$  20
- push  $\rightarrow$  30
- Front = null
- Rear = null

Handwritten notes at the top center:

pop  $\rightarrow$  pop  $\rightarrow$

Diagram showing a table with three columns: 10, 20, 30. The first column (10) is crossed out with a red line.

Diagram showing a linked list with three nodes: 10, 20, 30. The first node (10) is crossed out with a red line. The second node (20) is labeled 'Front' and the third node (30) is labeled 'Rear'. The next pointer of the second node is labeled 'N'.

Handwritten notes on the right side:

Rear.next = n  
n = Rear

Handwritten notes on the left side of the board:

```
void push(int a)
{
    QueueNode n = new QueueNode(a);
    if(front == null){ // queue is empty
        front = n;
        rear = n;
    }else{
        rear.next = n;
        rear = n;
    }
}

//Function to pop front element from the queue.
int pop()
{
    if(front == null){ //Queue is empty
        return -1;
    }else{
        int data = front.data;
        front = front.next;
        return data;
    }
}
```

Handwritten notes at the top center:

push  $\rightarrow$  10

push  $\rightarrow$  20

push  $\rightarrow$  30

pop  $\rightarrow$

pop  $\rightarrow$

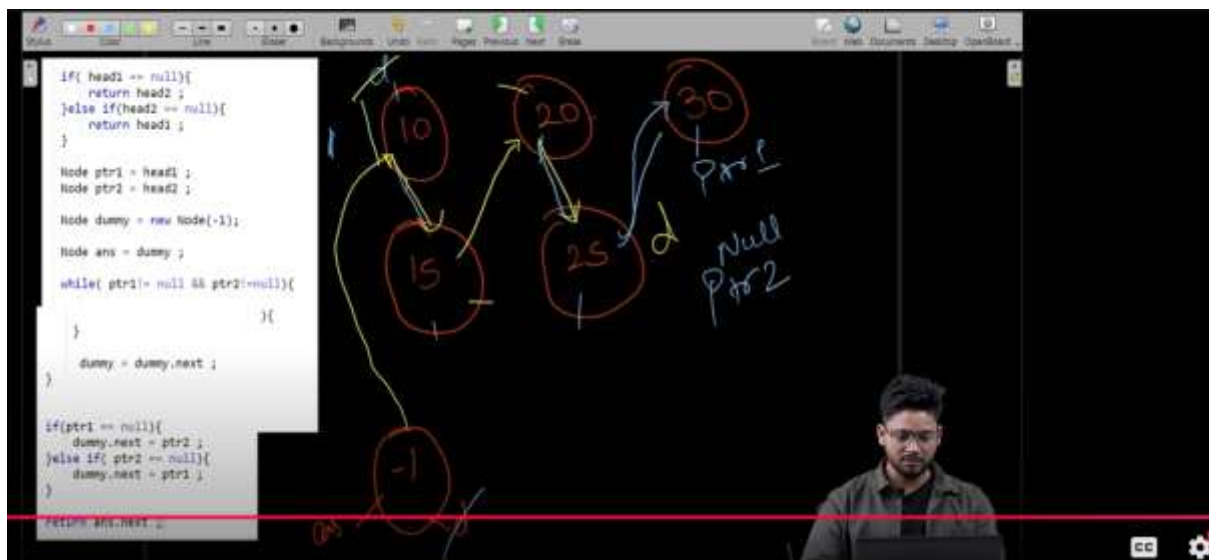
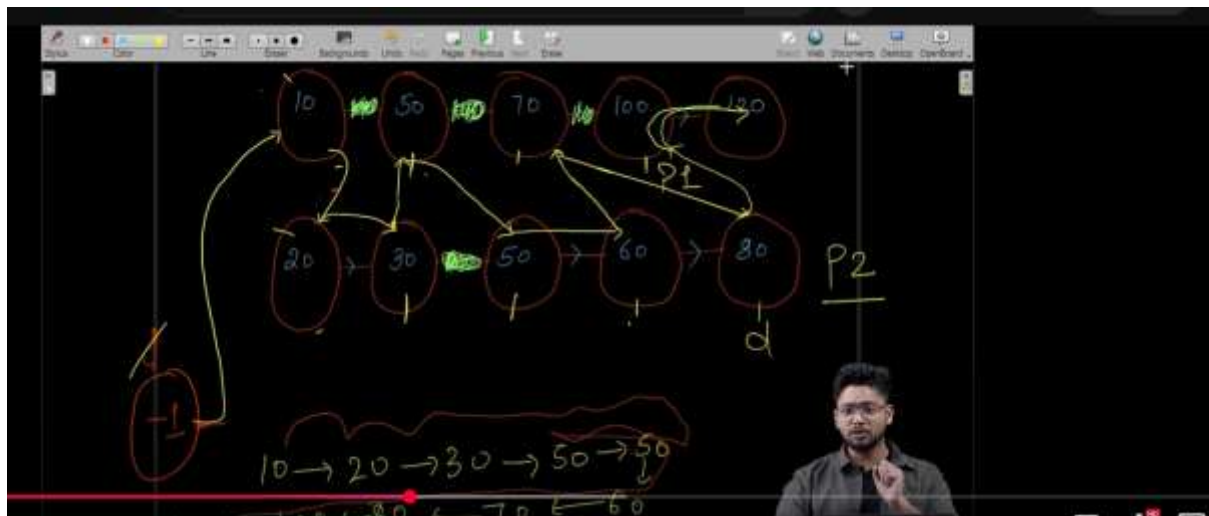
Diagram showing a linked list with three nodes: 10, 20, 30. The first node (10) is crossed out with a red line. The second node (20) is labeled 'Front' and the third node (30) is labeled 'Rear'. The next pointer of the second node is labeled 'n'. The first node (10) is labeled 'data'.

Handwritten notes on the right side:

Front = null

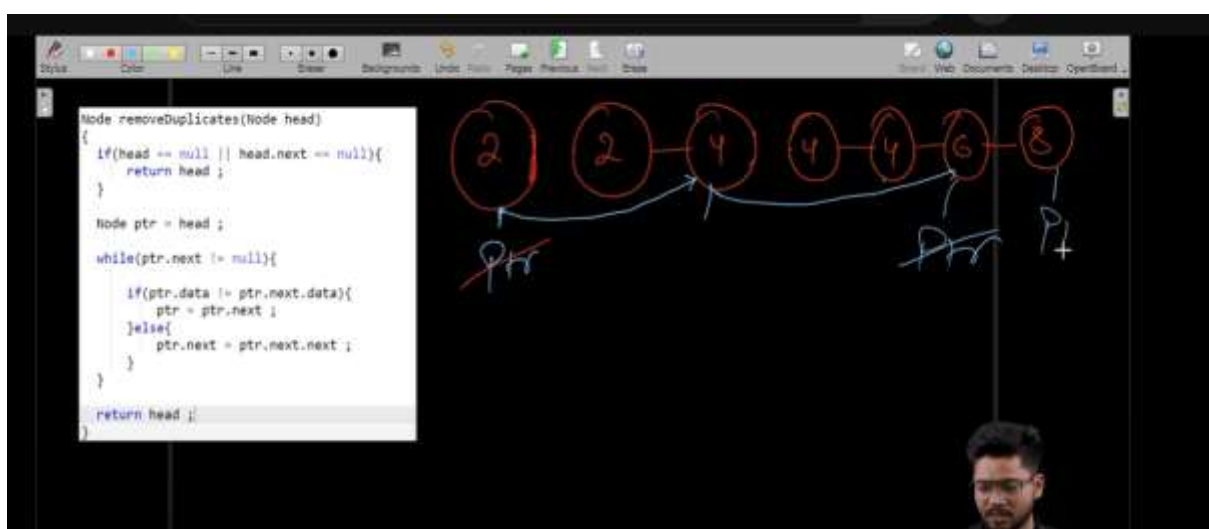
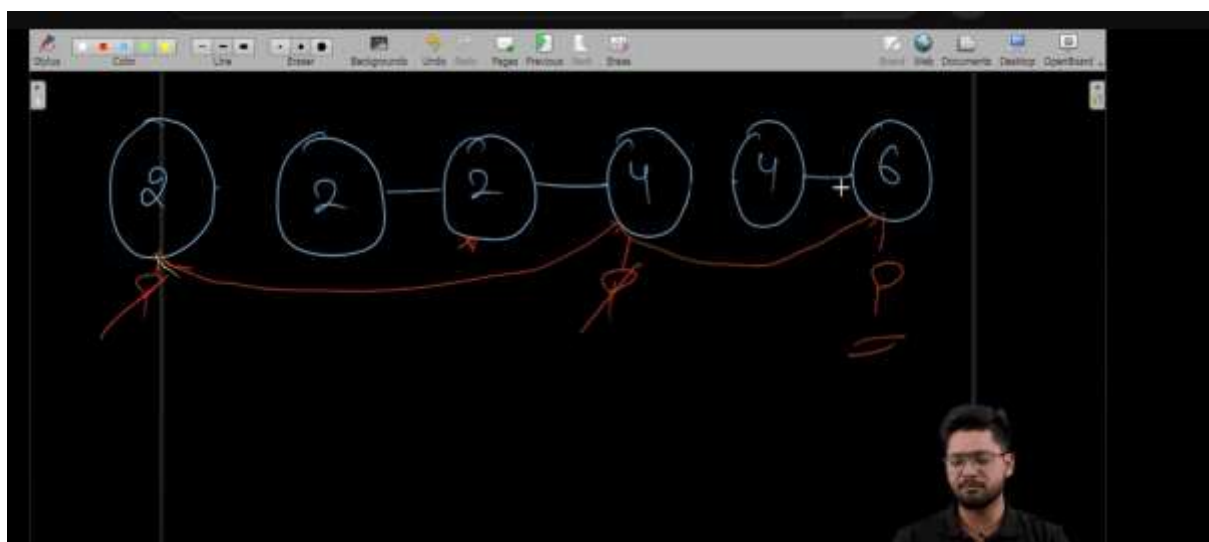
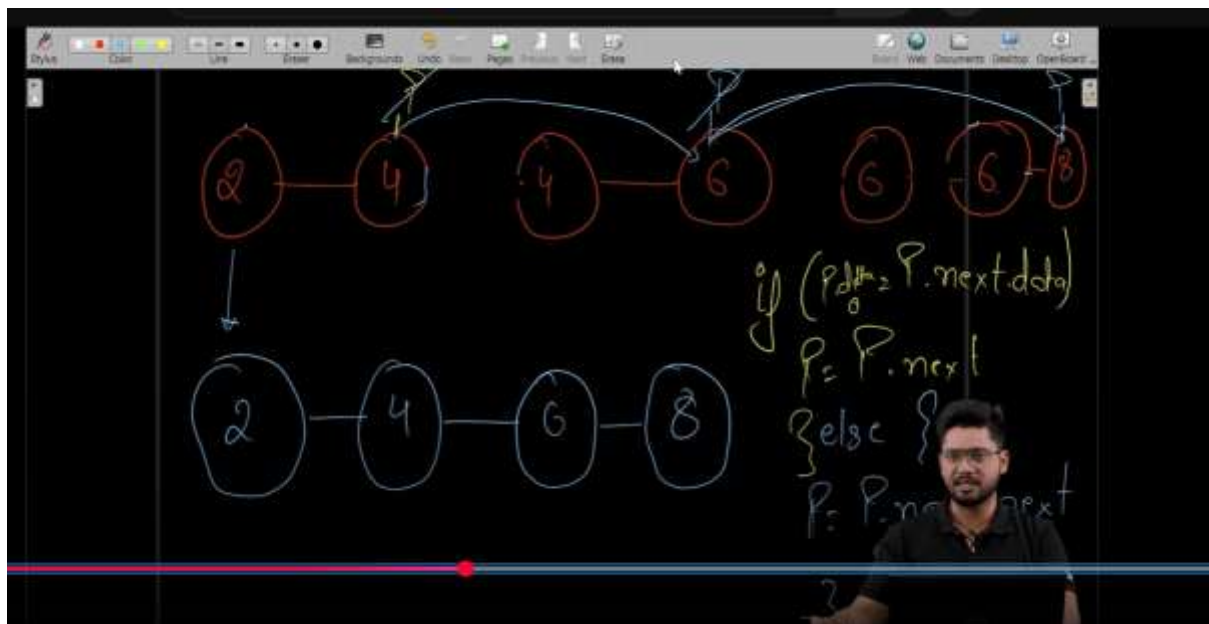
Rear = null

## 7.merge Two Sorted LinkedList





## 8. Remove duplicate element from sorted linked list



## 9. Add Two Number represented by linked list

Sum = val1 + val2 + carry

Sum = 11

Carry = 1

L1: 4 → 6 → 3 → 2

L2: 6 → 5 → 1 → 7

Sum: 1 → 1 → 4 → 9

```

if(first == null){
    return second;
} else if( second == null){
    return first;
}

Node ptr1 = reverseList(first);
Node ptr2 = reverseList(second);

int carry = 0;
Node ans = null;

while(ptr1 != null || ptr2 != null){
    int val1 = ptr1 != null ? ptr1.data : 0;
    int val2 = ptr2 != null ? ptr2.data : 0;

    int sum = val1 + val2 + carry;
    carry = sum / 10;
    int d = sum % 10;

    Node n = new Node(d);
    if(ans == null){
        ans = n;
    } else {
        n.next = ans;
        ans = n;
    }

    ptr1 = ptr1 != null ? ptr1.next : null;
    ptr2 = ptr2 != null ? ptr2.next : null;
}

if(carry > 0){
    Node n = new Node(carry);
    n.next = ans;
    ans = n;
}
    
```

Handwritten diagrams show the reversal of the lists and the addition process.

```

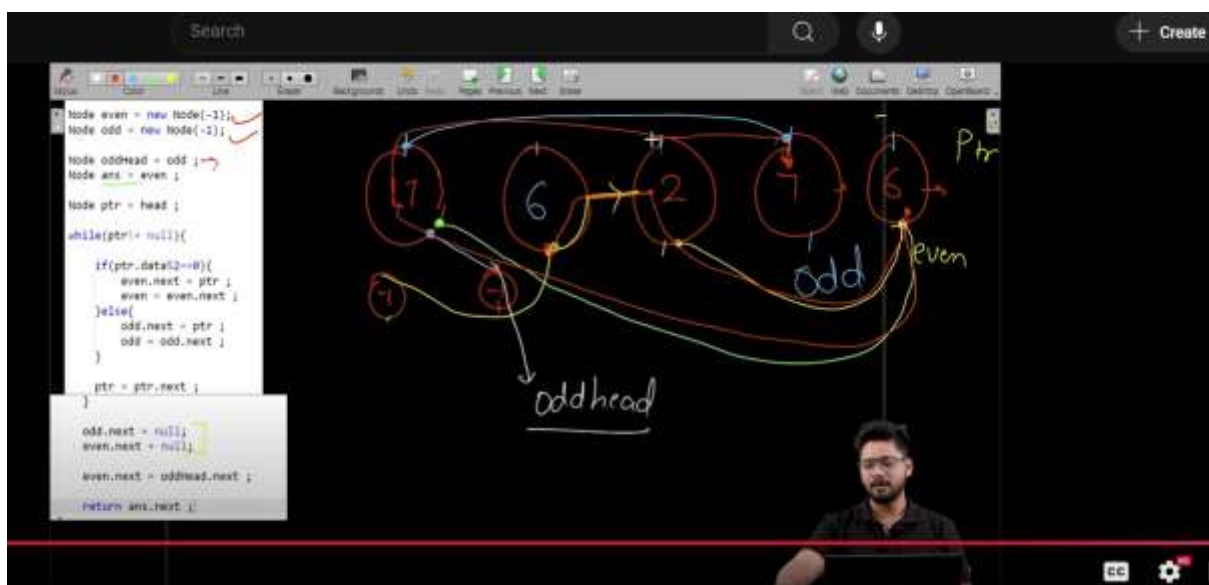
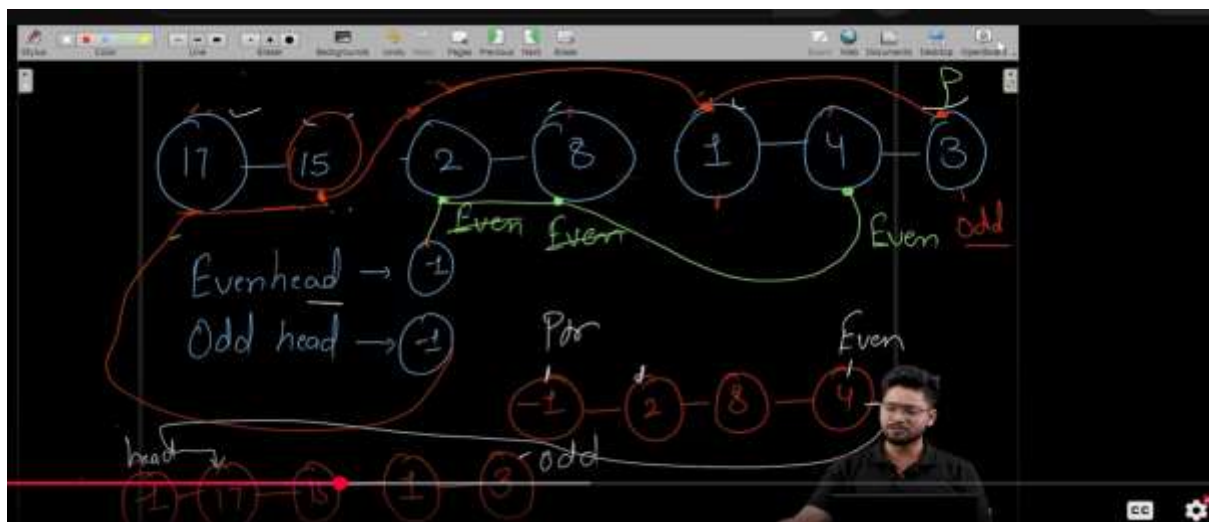
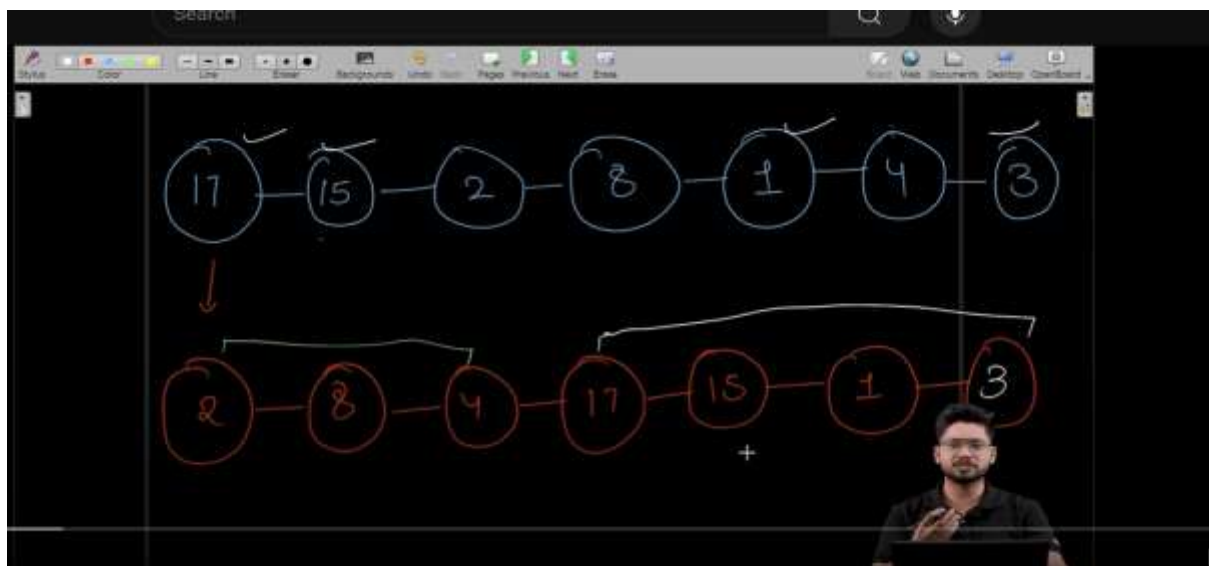
//Function to reverse a linked list.
Node reverseList(Node head)
{
    Node pre = null;
    Node curr = head;

    while(curr != null){
        Node nbr = curr.next;
        curr.next = pre;
        pre = curr;
        curr = nbr;
    }

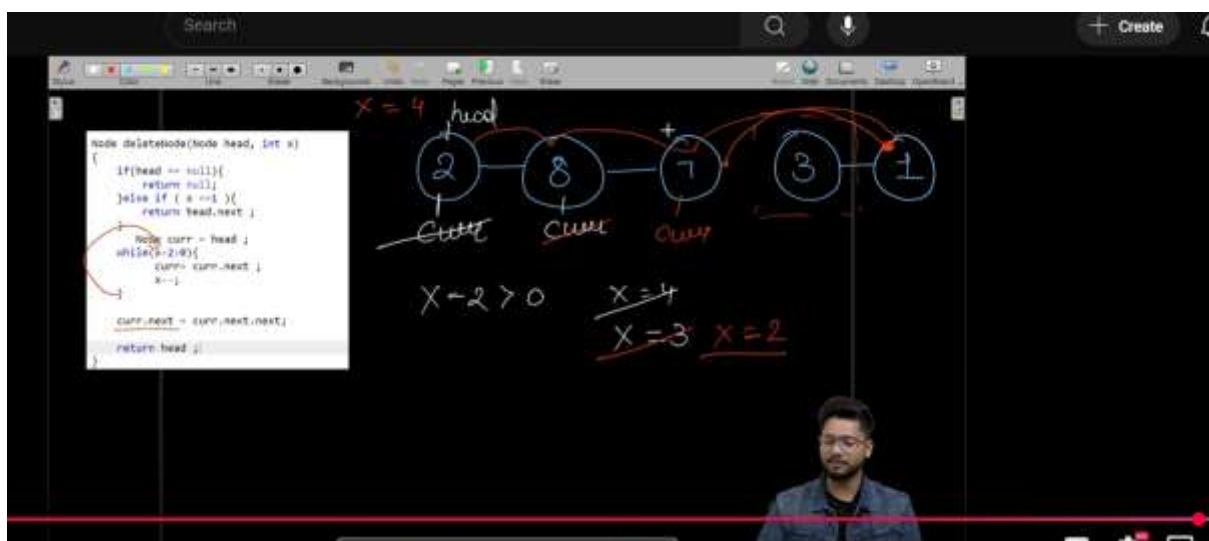
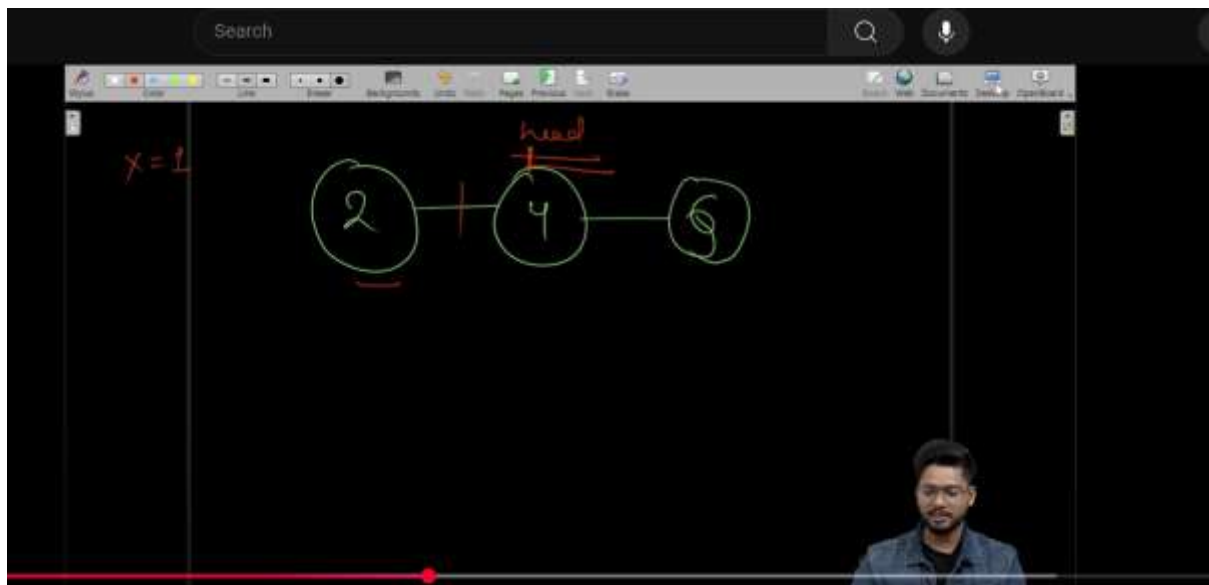
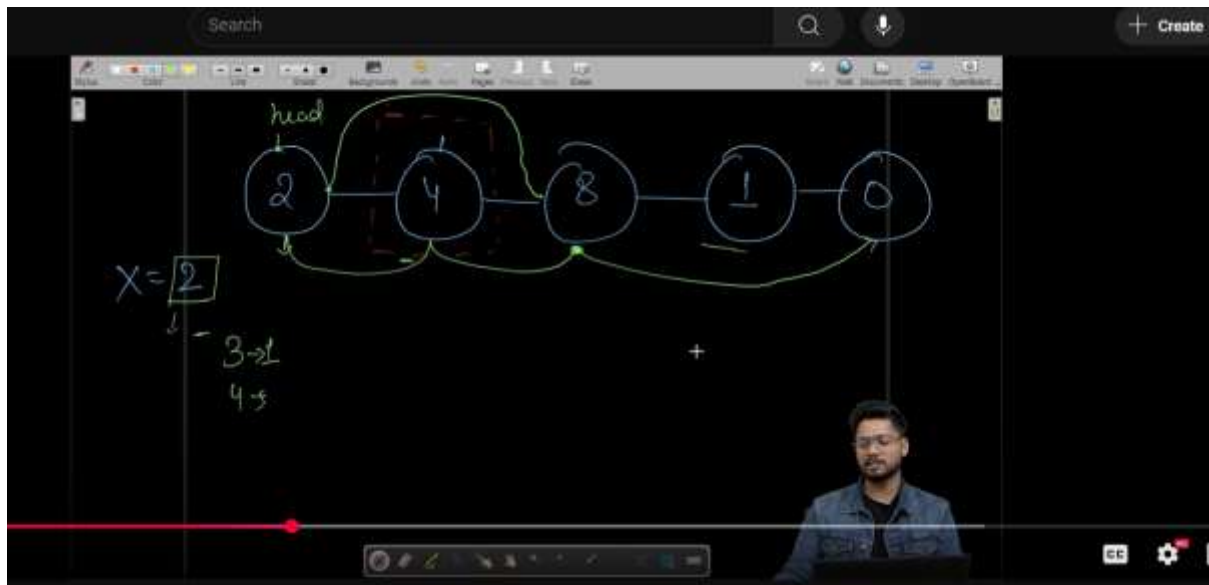
    return pre;
}
    
```

Handwritten diagrams show the reversal process.

## 10. Segregate even and odd in linked list

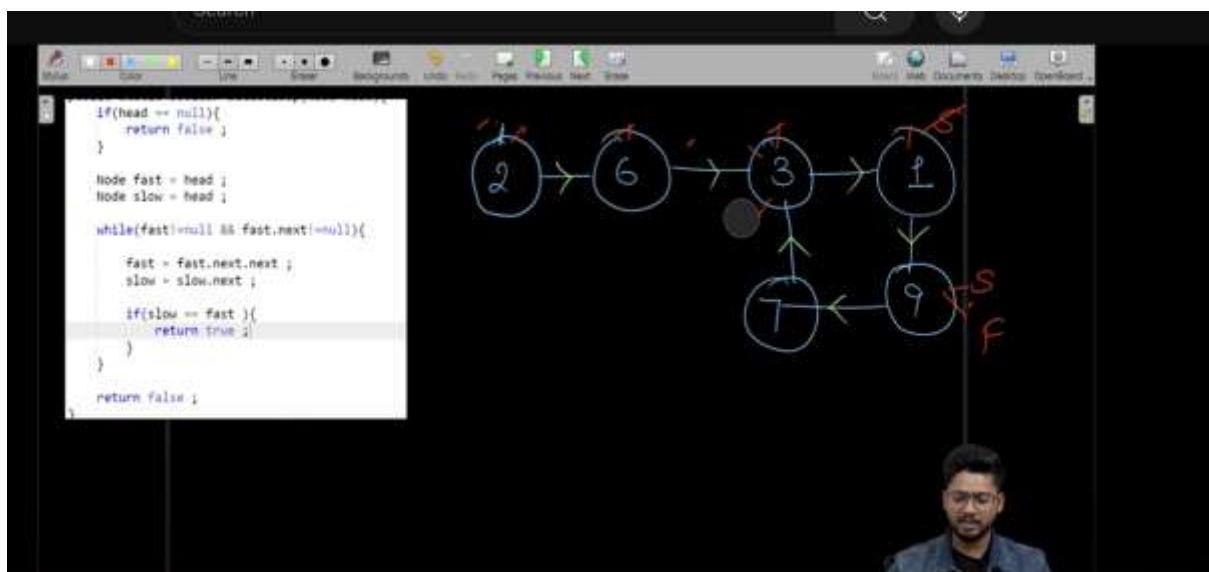
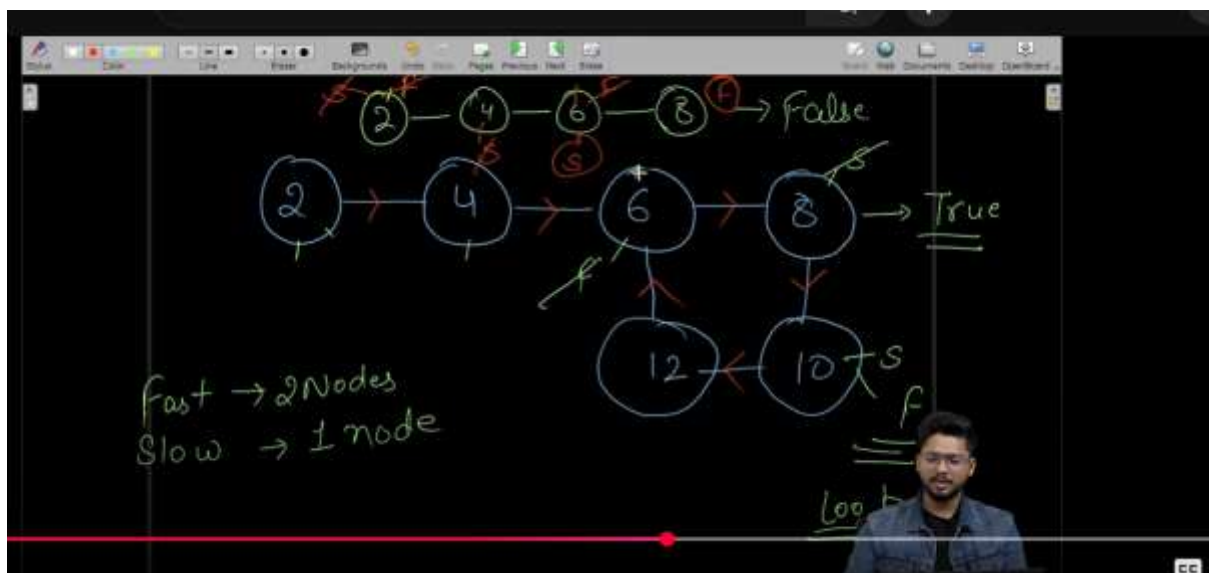
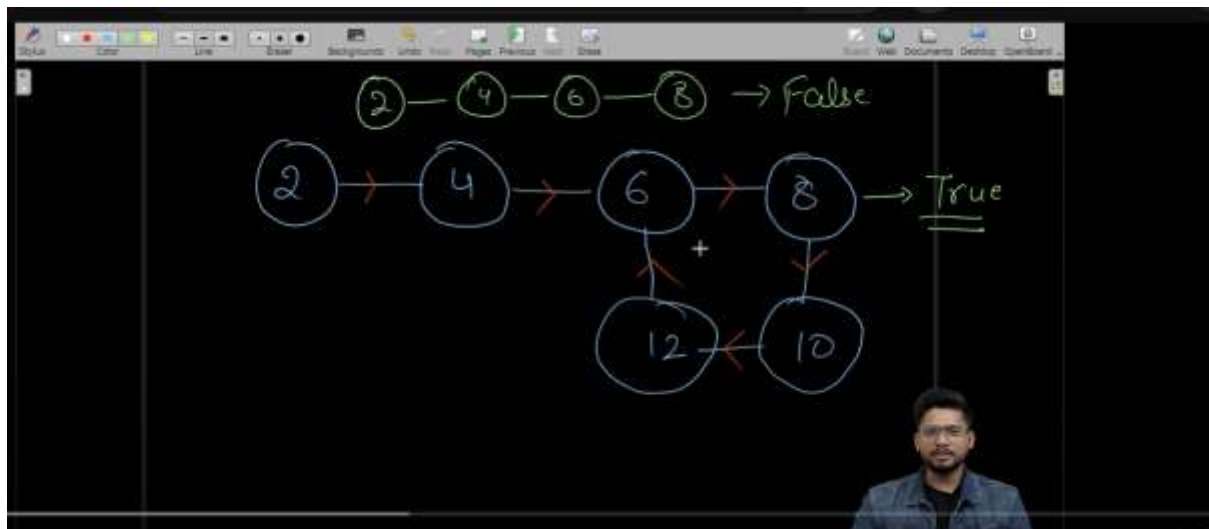


## 11.Delete a node in single linked list



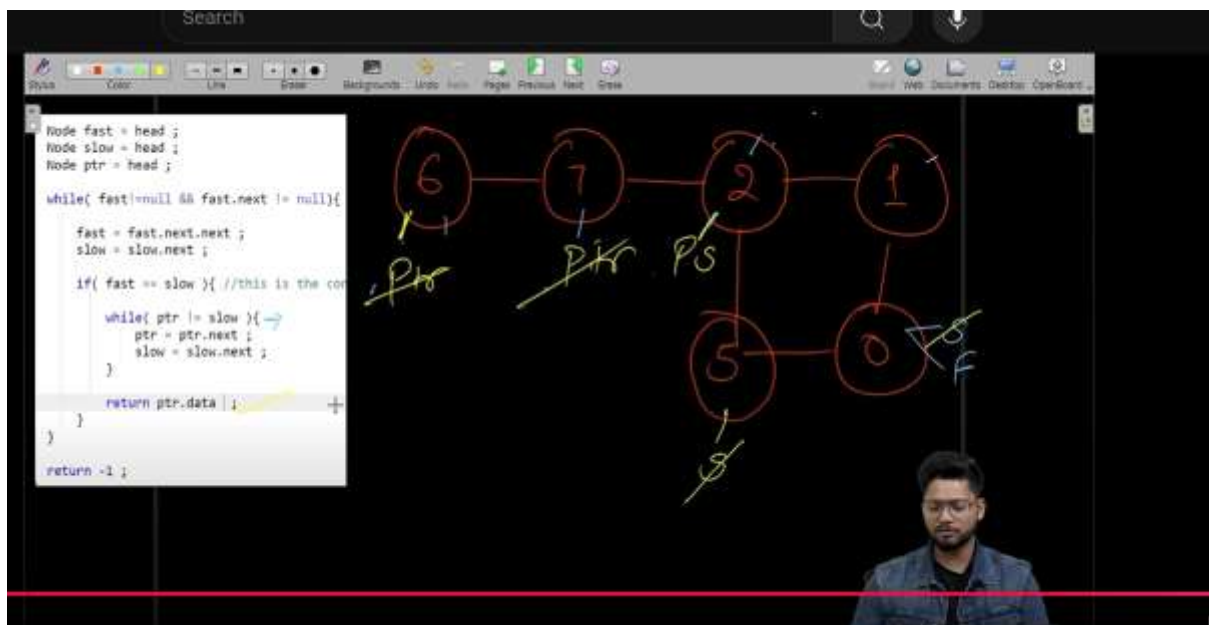
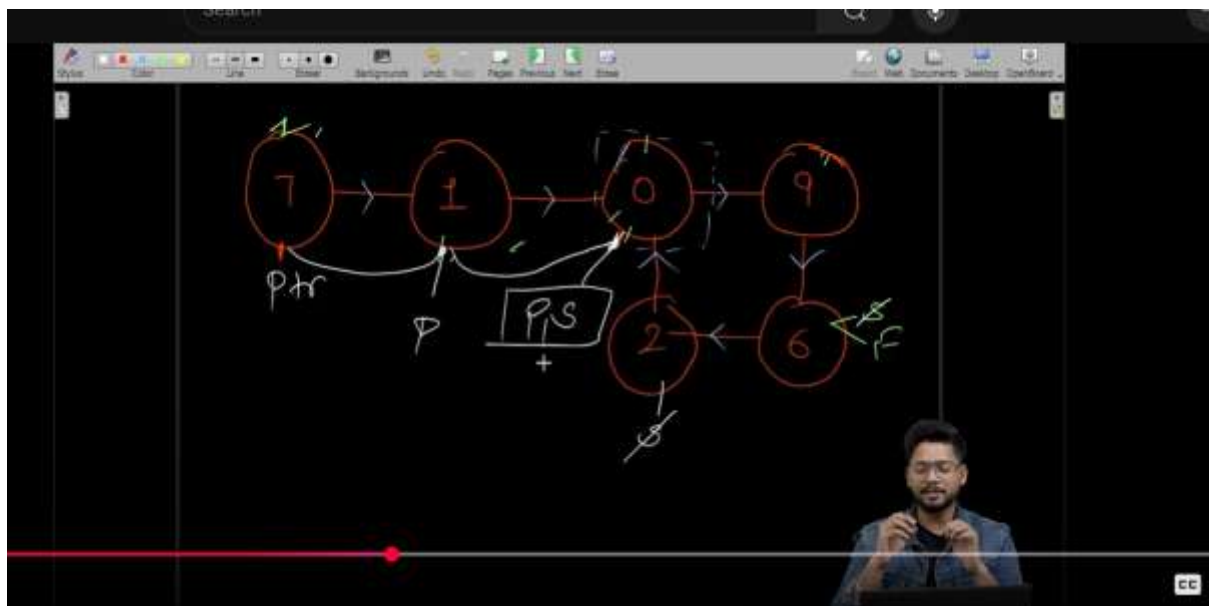
```
node deletenode(node head, int x)
{
    if(head == null){
        return null;
    }else if (x == 1){
        return head.next;
    }
    Node curr = head;
    while(x > 2){
        curr = curr.next;
        x--;
    }
    curr.next = curr.next.next;
    return head;
}
```

## 12. Detect loop in linked list





### 13.find the first node of loop in linked list



## 14 .Find the length of the loop in linked list

Hand-drawn diagram of a linked list with a loop. The list contains nodes 4, 6, 7, 3, and 2. Node 2 points to node 1, which points back to node 3, creating a loop. A box around node 1 is labeled 'S' and 'F' with a '+' sign. To the right, 'Count' is written with a list of values: 1, 2, 3.

Hand-drawn diagram of a linked list with a loop. The list contains nodes 10, 20, 30, 50, and 40. Node 30 points to node 50, which points back to node 20, creating a loop. To the right, 'Count' is written with a list of values: 1, 2, 3, 4.

```
Node fast = head ;
Node slow = head ;

while(fast!=null && fast.next!=null){
    fast = fast.next.next ;
    slow = slow.next ;

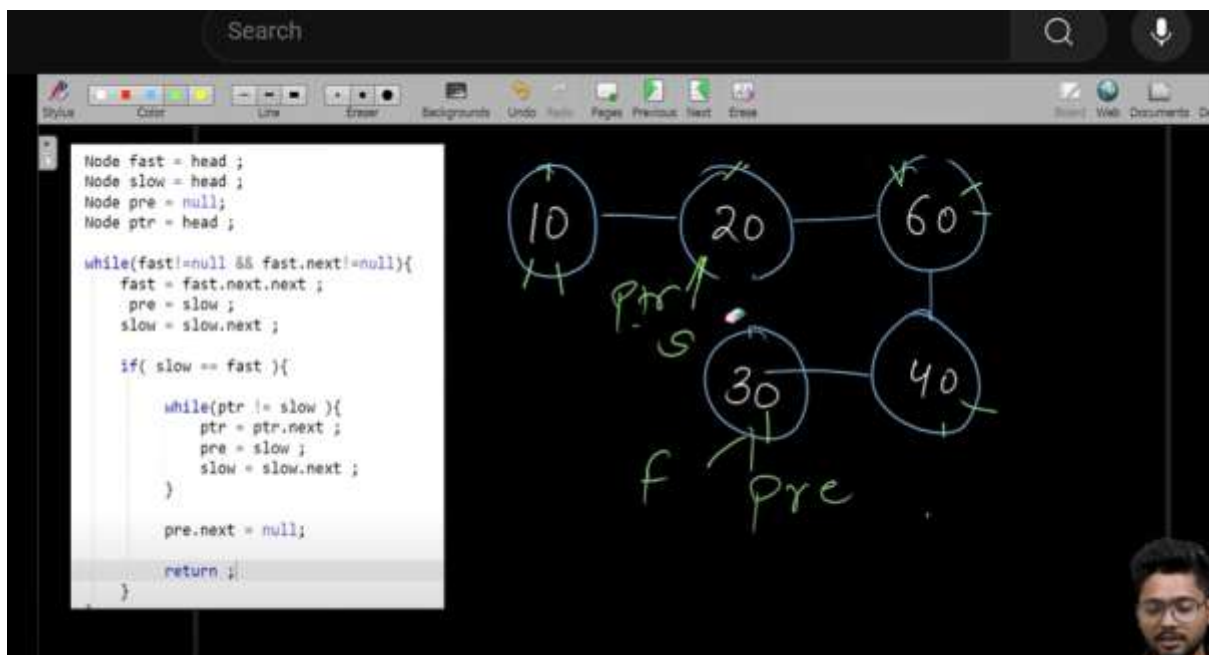
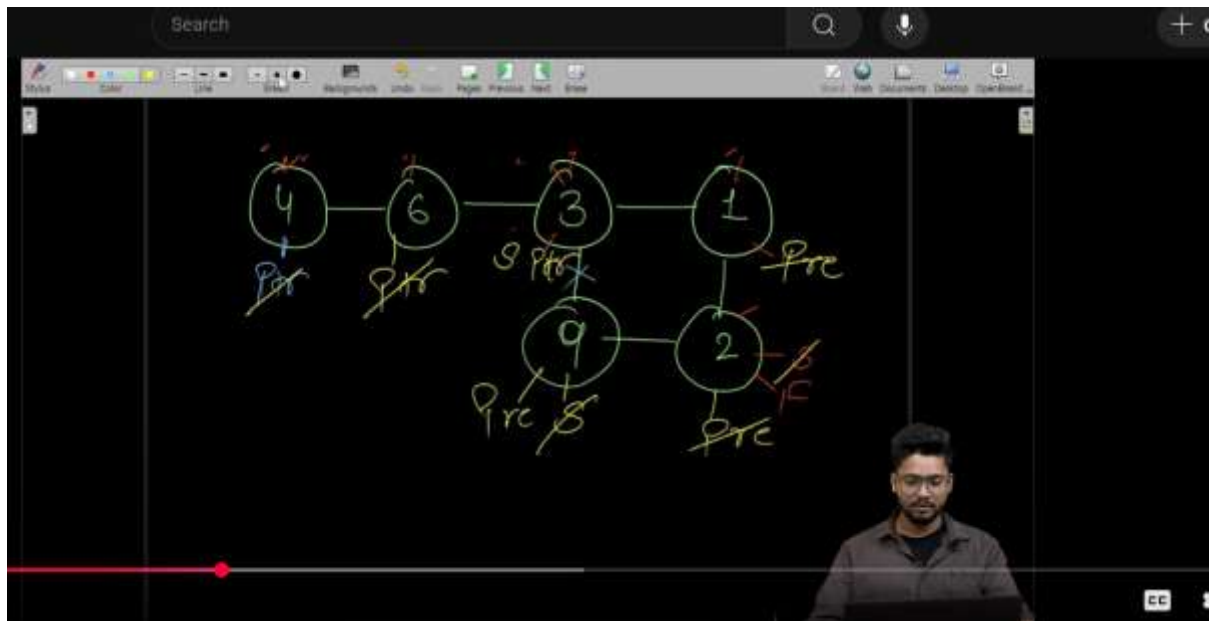
    if(fast == slow ){ // there is a loop
        int count = 1 ;
        slow = slow.next ;

        while(fast!=slow ){
            slow = slow.next ;
            count++;
        }

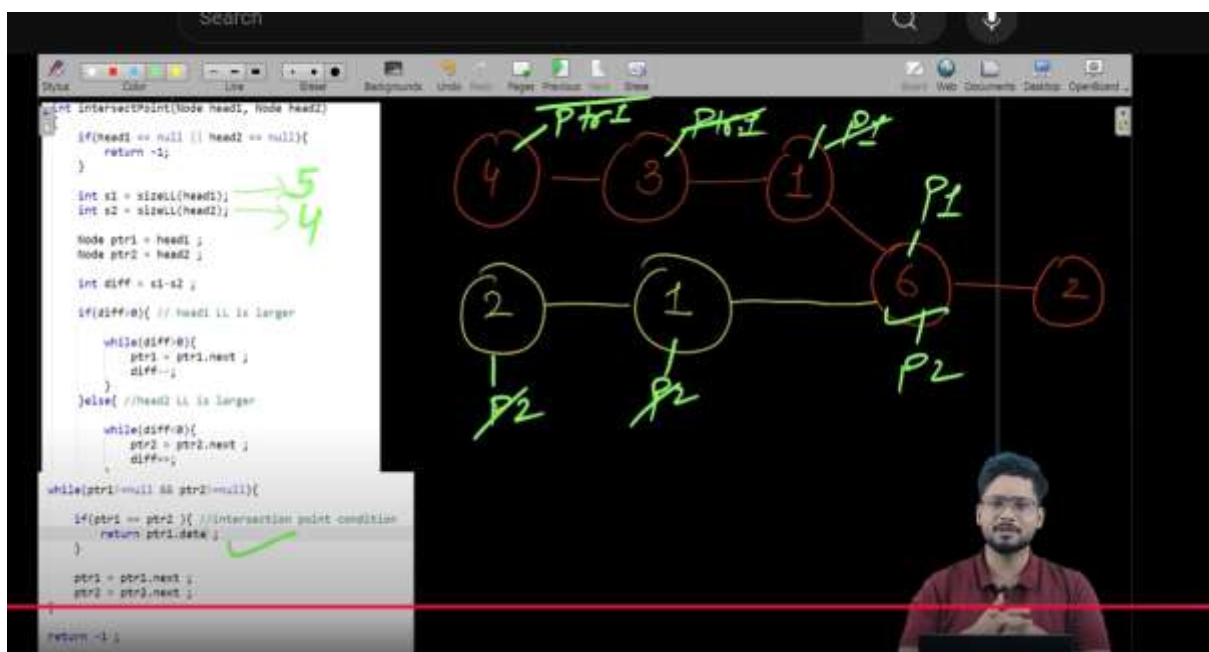
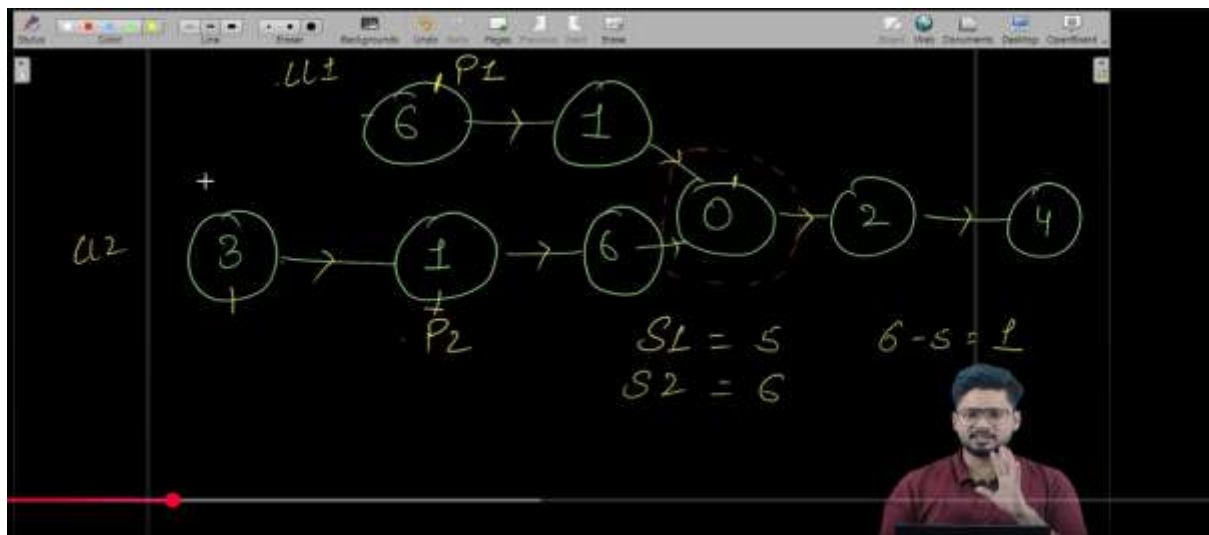
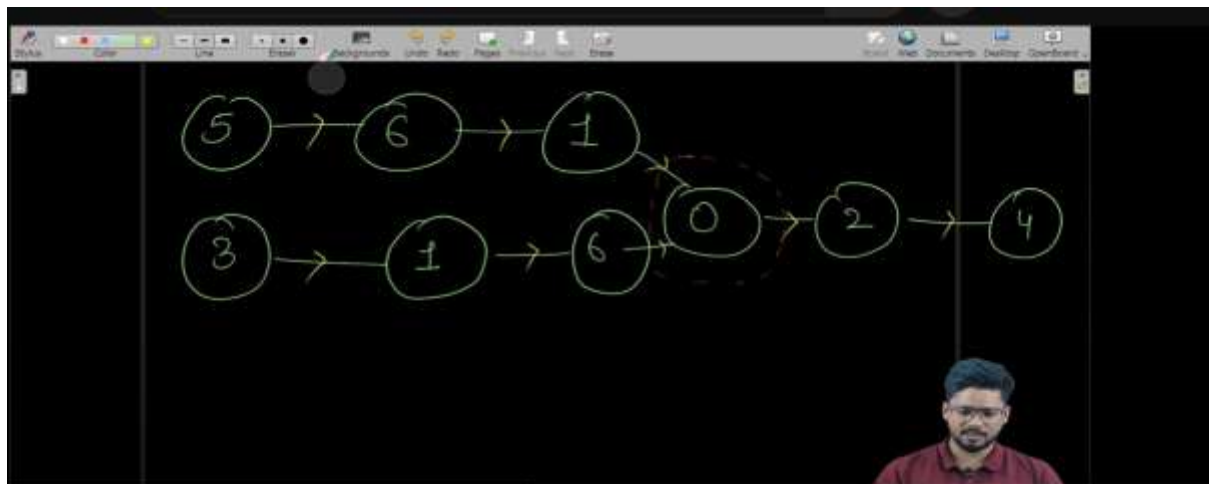
        return count ;
    }
}

return 0 ;
```

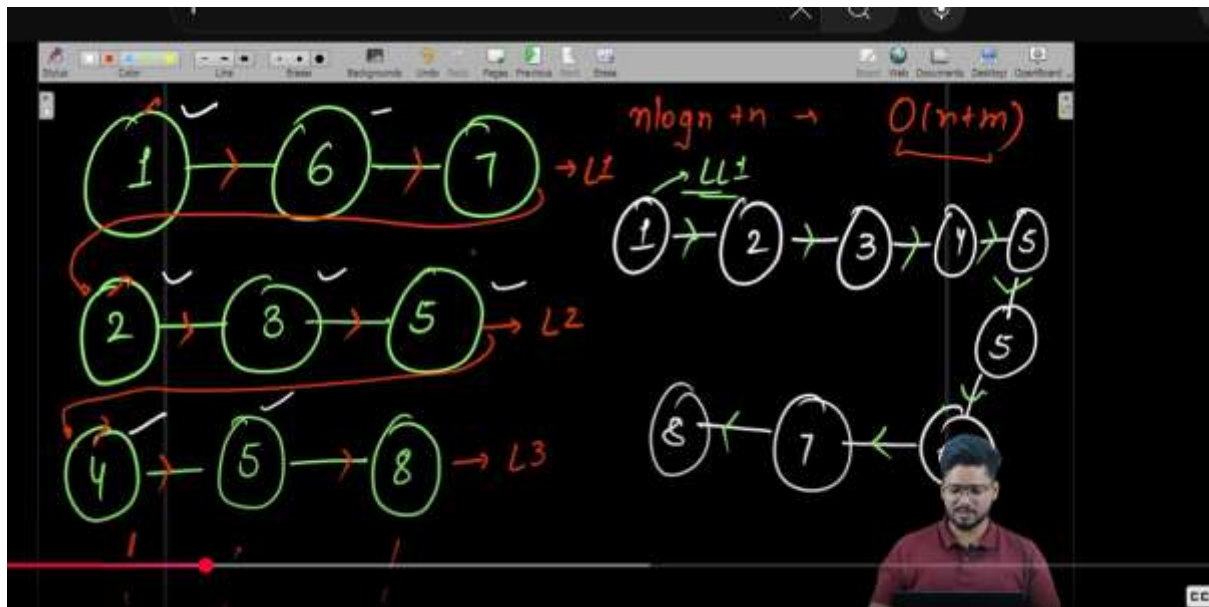
## 15. Remove loop in linked list



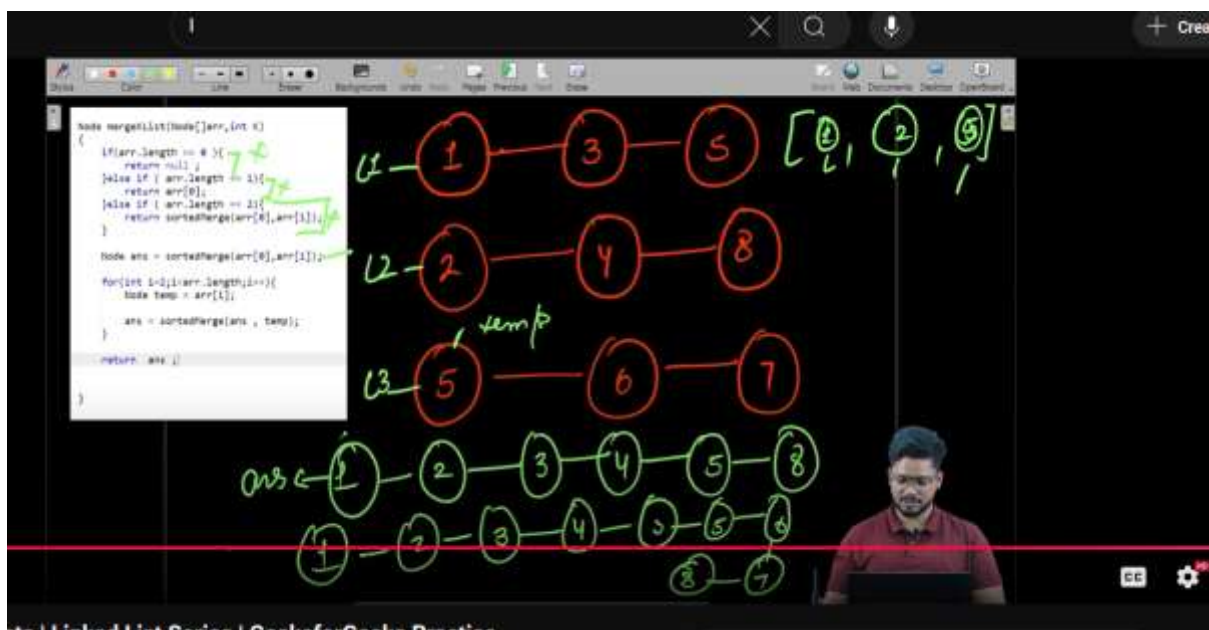
## 16. Intersection point in Y shaped linked list



## 17.Merge k Sorted Linked List



**Approach :** In this, first we merge the first two arr into linked list and then for upcoming arr we use for loop from  $i=2$ , and most important is to use the exact code for (merge two sorted linked list) ans we call that function(sortedMerge) when converted arr to linked node.





## 18.Rotate a Linked List

Input:  
 N = 5  
 value[] = {2, 4, 7, 8, 9}  
 k = 3  
 Output: 8 9 2 4 7  
 Explanation:  
 Rotate 1: 4 → 7 → 8 → 9 → 2  
 Rotate 2: 7 → 8 → 9 → 2 → 4  
 Rotate 3: 8 → 9 → 2 → 4 → 7

Input:  
 N = 5  
 value[] = {2, 4, 7, 8, 9}  
 k = 3  
 Output: 8 9 2 4 7  
 Explanation:  
 Rotate 1: 4 → 7 → 8 → 9 → 2  
 Rotate 2: 7 → 8 → 9 → 2 → 4  
 Rotate 3: 8 → 9 → 2 → 4 → 7

Handwritten calculations and arrays:  
 $k = 12 \rightarrow \frac{12 \% 5}{\text{size}} \Rightarrow 2$   
 $R1 \rightarrow [4, 7, 8, 9, 2]$   
 $R2 \rightarrow [7, 8, 9, 2, 4]$   
 $R3 \rightarrow [8, 9, 2, 4, 7]$   
 $R4 \rightarrow [9, 2, 4, 7, 8]$   
 $R5 \rightarrow [2, 4, 7, 8, 9]$   
 $R10 \rightarrow [2, 4, 7, 8, 9]$   
 $R11 \rightarrow [4, 7, 8, 9, 2]$   
 $R12 \rightarrow [7, 8, 9, 2, 4]$

Handwritten diagram illustrating the rotation of a linked list:

The diagram shows the initial list: 1 → 2 → 3 → 4. The 'nhead' pointer is at node 3, and the 'tail' pointer is at node 4. The rotation is performed by moving the tail to the head, resulting in the new list: 3 → 4 → 1 → 2.

Rotate a Linked List | Practice | GeeksforGeeks

Rotate a Linked List [PUBLISHED]

Medium Accuracy: 39.95% Submissions: 200K+ Points: 4

Given a singly linked list of size  $N$ . The task is to left-shift the linked list by  $k$  nodes, where  $k$  is a given positive integer smaller than or equal to length of the linked list.

Example 1:

Input:  
 $N = 5$   
 $arr[] = \{1, 2, 3, 4, 5\}$   
 $k = 2$

```
class Solution {
public:
    int sizeLL(Node head) {
        if(head == null) {
            return 0;
        }
        Node curr = head;
        int count = 0;
        while(curr != null) {
            curr = curr.next;
            count++;
        }
        return count;
    }
    Node rotate(Node head, int k) {
        if(head == null) {
            return null;
        }
        else if (k == 0) {
            return head;
        }
        int size = sizeLL(head);
        int ar = [

```

Rotate a Linked List | Practice | GeeksforGeeks

Rotate a Linked List [PUBLISHED]

Medium Accuracy: 39.95% Submissions: 200K+ Points: 4

Given a singly linked list of size  $N$ . The task is to left-shift the linked list by  $k$  nodes, where  $k$  is a given positive integer smaller than or equal to length of the linked list.

Example 1:

Input:  
 $N = 5$   
 $arr[] = \{1, 2, 3, 4, 5\}$   
 $k = 2$

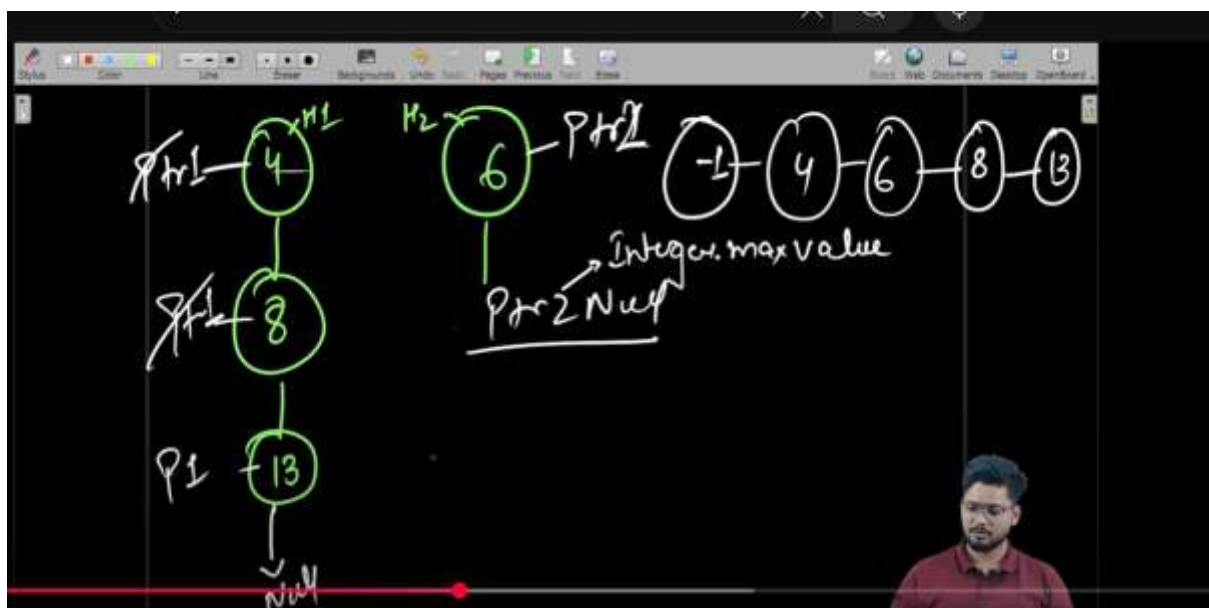
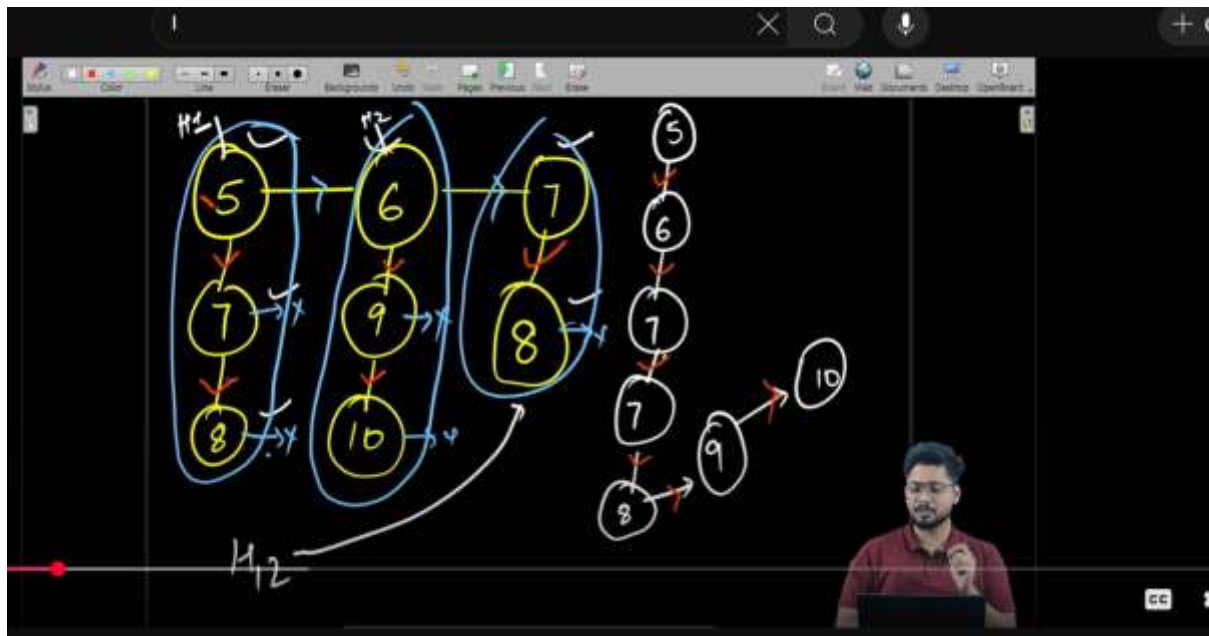
Diagram illustrating the rotation of a linked list by  $k$  nodes:

Initial linked list:  $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow \text{Null}$

After rotation by  $k=2$  nodes, the linked list becomes:  $3 \rightarrow 4 \rightarrow 1 \rightarrow 2$

The diagram shows the process of finding the  $k$ th node (node 3) and the node at  $(N-k)$ th position (node 4), and then connecting the  $k$ th node to the node at  $(N-k)$ th position.

## 19.Flattening a Linked List



```

Node mergeBottomSorted(Node head1 , Node head2){
    if(head1 == null || head2 == null){
        return head1==null?head2:head1;
    }

    Node ptr1 = head1 ;
    Node ptr2 = head2 ;
    Node dummy = new Node(-1);
    Node ans = dummy ;

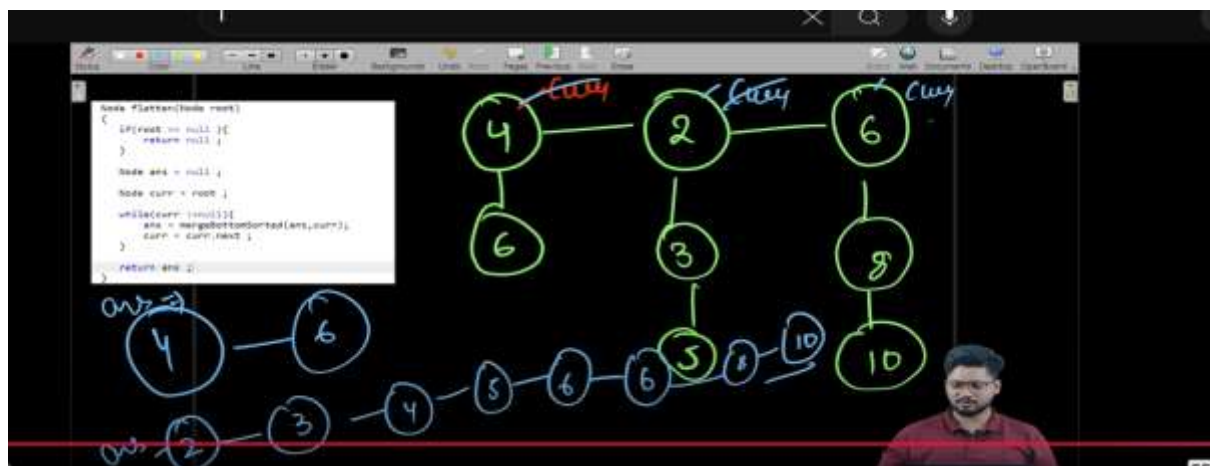
    while(ptr1!=null || ptr2!=null){
        int val1 = ptr1!=null ? ptr1.data : Integer.MAX_VALUE ;
        int val2 = ptr2!=null ? ptr2.data : Integer.MAX_VALUE ;

        if(val1<val2 ){
            dummy.bottom = ptr1 ;
            ptr1 = ptr1.bottom ;
        }else{
            dummy.bottom = ptr2 ;
            ptr2 = ptr2.bottom ;
        }

        dummy = dummy.bottom ;
    }

    return ans.bottom ;
}

```



## 20.Insertion sort for Single linked list

The image shows a video lecture interface. On the left, a code editor displays the following C++ code for insertion sort on a single linked list:

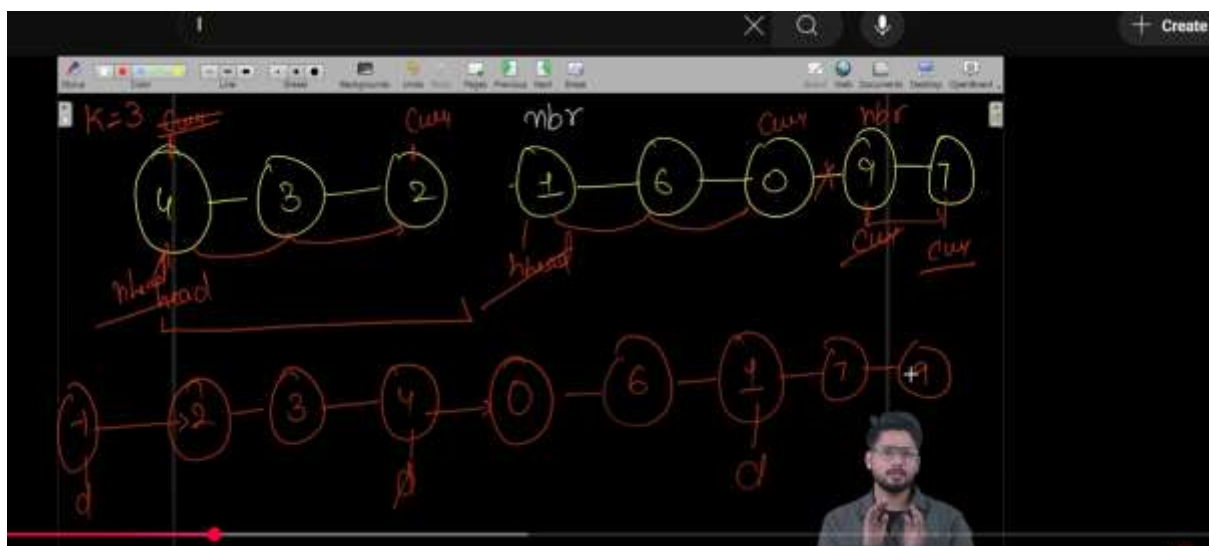
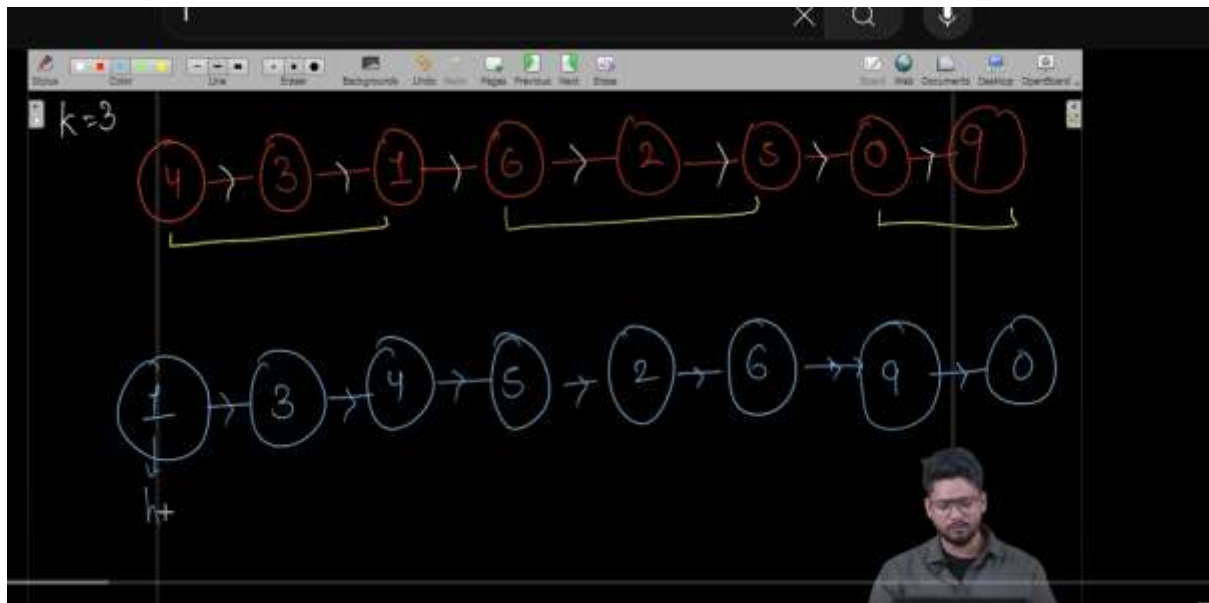
```
create a dummy node then node ans-dummy
Node curr = head;
while(curr != null){
    Node temp = ans;
    while(temp.next != null){
        if(temp.next.data < curr.data){
            temp = temp.next;
        } else{
            break;
        }
    }
    if(temp.next == null){
        Node nbr = curr.next;
        temp.next = curr;
        curr.next = null;
        curr = nbr;
    } else {
        Node nbr = curr.next;
        Node tempnbr = temp.next;
        temp.next = curr;
        curr.next = tempnbr;
        curr = nbr;
    }
}
return ans.next;
```

On the right, a hand-drawn diagram illustrates the linked list structure. It shows three nodes: a node with value 2, a node with value 4, and a node with value 3. The node with value 2 is labeled 'temp' and has an arrow pointing to the node with value 4, which is labeled 'curr'. The node with value 4 has an arrow pointing to the node with value 3, which is labeled 'nbr'. Below the node with value 2, there is a node with value -1, labeled 'ans', with an arrow pointing to the node with value 2. The diagram also shows a node with value 1, labeled 'temp', with an arrow pointing to the node with value -1.

A small inset in the bottom right corner shows a person, presumably the lecturer, speaking.



## 21.Reverse a linked list in group size





## Problem Set

### 22.Count Linked List Nodes

Given a singly linked list. The task is to find the length of the linked list, where length is defined as the number of nodes in the linked list.

Examples :

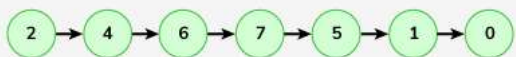
Input: LinkedList : 1->2->3->4->5



Output: 5

Explanation: Count of nodes in the linked list is 5, which is its length.

Input: LinkedList : 2->4->6->7->5->1->0



Output: 7

Explanation: Count of nodes in the linked list is 7. Hence, the output is 7.

```
5 class Solution {
6     // Function to count nodes of a linked list.
7     public int getCount(Node head) {
8         // code here
9         Node temp=head;
10        int count=0;
11
12        while(temp!=null){
13            count++;
14            temp=temp.next;
15        }
16        return count;
17    }
18 }
19 // } Driver Code Ends
```

## 23.Remove loop in LinkedList

Given the head of a linked list that may contain a loop. A loop means that the last node of the linked list is connected back to a node in the same list. The task is to remove the loop from the linked list (if it exists).

**Custom Input format:**

A **head** of a singly linked list and a **pos** (1-based index) which denotes the position of the node to which the last node points to. If **pos = 0**, it means the last node points to null, indicating there is no loop.

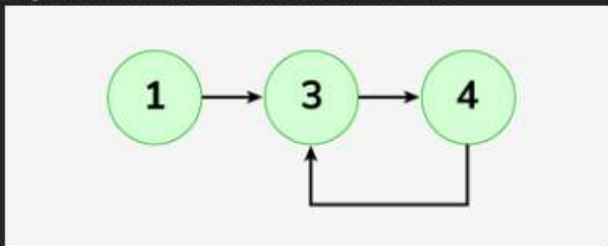
The generated output will be **true** if there is no loop in list and other nodes in the list remain unchanged, otherwise, **false**.

**Examples:**

**Input:** head = 1 -> 3 -> 4, pos = 2

**Output:** true

**Explanation:** The linked list looks like

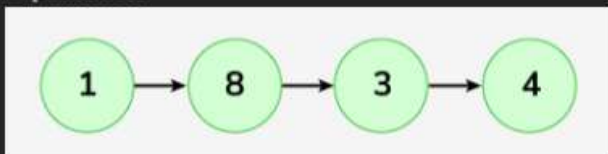


A loop is present in the list, and it is removed.

**Input:** head = 1 -> 8 -> 3 -> 4, pos = 0

**Output:** true

**Explanation:**

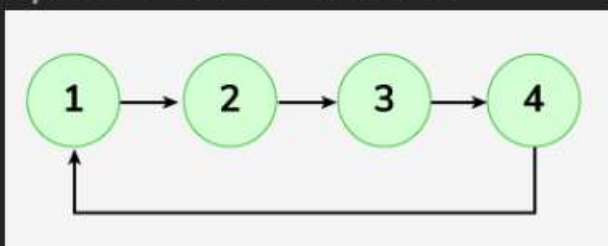


The Linked list does not contains any loop.

**Input:** head = 1 -> 2 -> 3 -> 4, pos = 1

**Output:** true

**Explanation:** The linked list looks like



A loop is present in the list, and it is removed.

```
class Solution {
    // Function to remove a loop in the linked list.
    public static void removeLoop(Node head) {
        // code here
        if(head==null){
            return;
        }
        Node slow=head;
        Node fast=head;
        Node prev=null;
        Node ptr=head;

        while(fast!=null && fast.next!=null){
            fast=fast.next.next;
            prev=slow;
            slow=slow.next;

            if(slow==fast){
                while(ptr!=slow){
                    ptr=ptr.next;
                    prev=slow;
                    slow=slow.next;
                }
                prev.next=null;
                return;
            }
        }
    }
}
```



## 24.check if the Circular Linked List

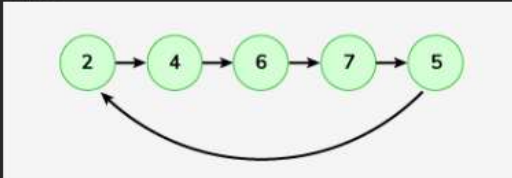
Given the **head**, the head of a singly linked list, Returns **true** if the linked list is circular & **false** if it is not circular.

A linked list is called circular if it is not NULL terminated and all nodes are connected in the form of a cycle.

**Note:** The linked list does not contain any inner loop.

**Examples:**

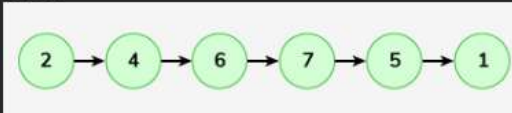
**Input:**



**Output:** true

**Explanation:** As shown in figure the first and last node is connected, i.e. 5 --> 2

**Input:**



**Output:** false

**Explanation:** As shown in figure this is not a circular linked list.

```

}
*/
class Solution {
    boolean isCircular(Node head) {
        // Your code here
        // Your code here
        Node temp=head;
        while(temp!=null)
        {
            if(temp.next==head)
            {
                return true;
            }
            temp = temp.next;
        }

        return false;
    }
}
```

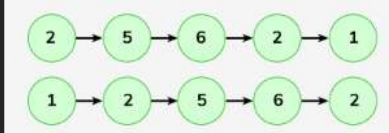
## 25. Move last Element to front of a linked list

You are given the head of a Linked List. You have to move the last element to the front of the Linked List and return the head of the modified linked list.

**Examples:**

**Input:** Linked List: 2->5->6->2->1

**Output:** 1->2->5->6->2



**Explanation:** In the given linked list, the last element is 1, after moving the last element to the front the linked list will be 1->2->5->6->2

**Input:** Linked List: 2

**Output:** 2

**Explanation:** Here 2 is the only element so, the linked list will remain the same.

**Expected Time Complexity:**  $O(n)$

```
class Solution {
    public static Node moveToFront(Node head) {
        // code here
        Node temp=head;
        Node last=null;
        while(temp.next!=null){
            last=temp;
            temp=temp.next;
        }

        last.next=null;
        temp.next=head;
        return temp;
    }
}
```

## 26.Kth from End of Linked List

Given the head of a linked list and the number  $k$ , Your task is to find the  $k^{\text{th}}$  node from the end. If  $k$  is more than the number of nodes, then the output should be -1.

### Examples

**Input:** LinkedList: 1->2->3->4->5->6->7->8->9,  $k = 2$

**Output:** 8

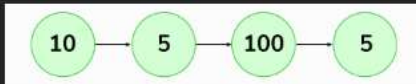
**Explanation:** The given linked list is 1->2->3->4->5->6->7->8->9. The 2nd node from end is 8.



**Input:** LinkedList: 10->5->100->5,  $k = 5$

**Output:** -1

**Explanation:** The given linked list is 10->5->100->5. Since 'k' is more than the number of nodes, the output is -1.



```
class Solution {  
    // Function to find the data of kth node from  
    // the end of a linked list.  
    int callLength(Node head){  
        Node curr = head;  
        int length = 0;  
        while(curr != null){  
            curr = curr.next;  
            length++;  
        }  
        return length;  
    }  
    int getKthFromLast(Node head, int k) {  
        if(head == null){  
            return -1;  
        }  
  
        int length = callLength(head);  
  
        if(k > length){  
            return -1;  
        }  
  
        Node curr = head;  
        for(int i = 0; i < length - k; i++){  
            curr = curr.next;  
        }  
        return curr.data;  
    }  
}
```

// } Driver Code Ends

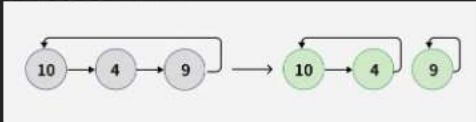
## 27.Split a Linked List into two halves

Given a Circular linked list. The task is split into two Circular Linked lists. If there are an odd number of nodes in the given circular linked list then out of the resulting two halved lists, the first list should have one node more than the second list.

Examples :

Input: LinkedList : 10->4->9

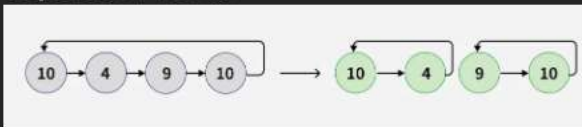
Output: 10->4 , 9



Explanation: After dividing linked list into 2 parts , the first part contains 10, 4 and second part contain only 9.

Input: LinkedList : 10->4->9->10

Output: 10->4 , 9->10



Explanation: After dividing linked list into 2 parts , the first part contains 10, 4 and second part contain 9, 10.

```
0  /
9
0  class Solution {
1  public Pair<Node, Node> splitList(Node head) {
2      // Code here
3      if(head==null||head.next==null){
4          return null;
5      }
6      Node slow=head;
7      Node fast=head.next;
8
9      while(fast!=head&&fast.next!=head){
10         slow=slow.next;
11         fast=fast.next.next;
12     }
13     Node temp=slow.next;
14     slow.next=head;
15     fast=temp;
16
17     while(fast.next!=head){
18         fast=fast.next;
19     }
20     fast.next=temp;
21
22     return new Pair<>(head, temp);
23
24 }
25 }
```

## 28.Remove duplicate from unsorted Linked list

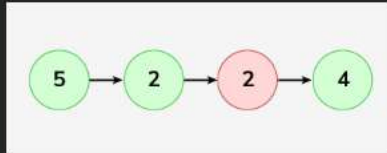
Given an unsorted linked list. The task is to remove duplicate elements from this unsorted Linked List. When a value appears in multiple nodes, the node which appeared first should be kept, all other duplicates are to be removed.

**Examples:**

**Input:** LinkedList: 5->2->2->4

**Output:** 5->2->4

**Explanation:** Given linked list elements are 5->2->2->4, in which 2 is repeated only. So, we will delete the extra repeated elements 2 from the linked list and the resultant linked list will contain 5->2->4



**Input:** LinkedList: 2->2->2->2->2

**Output:** 2

**Explanation:** Given linked list elements are 2->2->2->2->2, in which 2 is repeated. So, we will delete the extra repeated elements 2 from the linked list and the resultant linked list will contain only 2.

```
class Solution {
    // Function to remove duplicates from unsorted linked list.
    public Node removeDuplicates(Node head) {
        // Your code here
        if(head==null){
            return head;
        }
        HashSet<Integer> s=new HashSet<>();
        Node temp=head;
        Node cur=head.next;
        s.add(temp.data);
        while(cur!=null){
            if(s.add(cur.data)){
                temp.next=cur;
                temp=cur;
            }
            cur=cur.next;
        }
        temp.next=null;
        return head;
    }
}
```



## 29.Add 1 to a Linked List number

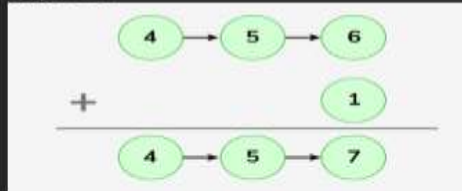
You are given a linked list where each element in the list is a node and have an integer data. You need to add 1 to the number formed by concatenating all the list node numbers together and return the head of the modified linked list.

**Note:** The head represents the first element of the given array.

**Examples :**

**Input:** LinkedList: 4->5->6

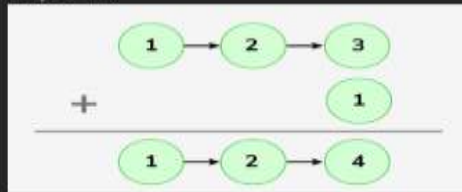
**Output:** 457



**Explanation:** 4->5->6 represents 456 and when 1 is added it becomes 457.

**Input:** LinkedList: 1->2->3

**Output:** 124



**Explanation:** 1->2->3 represents 123 and when 1 is added it becomes 124.

```
7  */
8
9  class Solution {
10     int rem=1;
11
12
13     public void add(Node head) {
14
15         if(head.next!=null){
16             add(head.next);
17         }
18         rem=head.data+rem;
19         head.data=rem%10;
20         rem/=10;
21         return ;
22     }
23     public Node addOne(Node head) {
24
25         add(head);
26         if(rem!=0){
27             Node l=new Node(rem);
28             l.next=head;
29             return l;
30         }
31         return head;
32     }
33 }
34
```

## 30.Intersection of sorted Linked list

Given that two linked lists are sorted in increasing order, create a new linked list representing the intersection of the two linked lists. The new linked list should be made without changing the original lists.

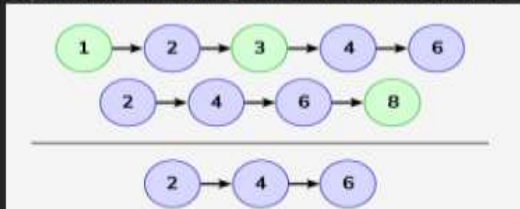
**Note:** The elements of the linked list are not necessarily distinct.

**Examples:**

**Input:** LinkedList1 = 1->2->3->4->6, LinkedList2 = 2->4->6->8

**Output:** 2->4->6

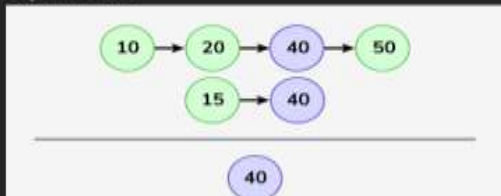
**Explanation:** For the given two linked list, 2, 4 and 6 are the elements in the intersection.



**Input:** LinkedList1 = 10->20->40->50, LinkedList2 = 15->40

**Output:** 40

**Explanation:**



```
class Solution {
    public static Node findIntersection(Node head1, Node head2) {
        // code here.
        Node temp=new Node(0);
        Node curr=temp;
        while(head1!=null && head2!=null){
            if(head1.data==head2.data){
                curr.next=head1;
                curr=curr.next;
                head1=head1.next;
                head2=head2.next;
            }
            else if(head1.data<head2.data){
                head1=head1.next;
            }
            else if(head1.data>head2.data){
                head2=head2.next;
            }
        }
        curr.next=null;
        return temp.next;
    }
}
```

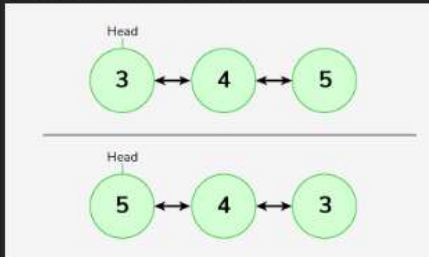
## 31.Reverse a Double Linked list

Given a **doubly linked list**. Your task is to **reverse** the doubly linked list and return its head.

Examples:

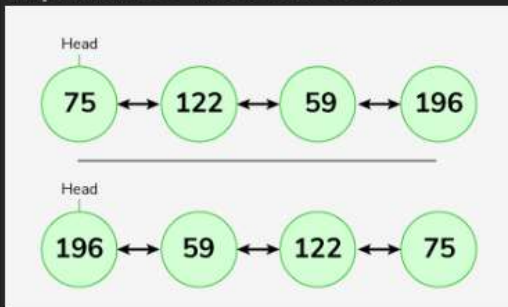
**Input:** LinkedList: 3 <-> 4 <-> 5

**Output:** 5 <-> 4 <-> 3



**Input:** LinkedList: 75 <-> 122 <-> 59 <-> 196

**Output:** 196 <-> 59 <-> 122 <-> 75



```
0  */
1  class Solution {
2  public DLLNode reverseDLL(DLLNode head) {
3      // Your code here
4      if(head == null || head.next == null){
5          return head;
6      }
7
8      DLLNode tail = head;
9      while(tail.next != null){
10         tail = tail.next;
11     }
12
13     DLLNode newhead = tail;
14     while(tail != null){
15         tail.next = tail.prev;
16         tail.prev = tail.next;
17         tail = tail.next;
18     }
19     return newhead;
20 }
21 }
```

## 32.Delete node having greater value on right

Given a singly linked list, remove all nodes that have a node with a greater value anywhere to their right in the list. Return the head of the modified linked list.

Examples:

Input:  
LinkedList = 12->15->10->11->5->6->2->3  
Output: 15->11->6->3



Explanation: Since, 12, 10, 5 and 2 are the elements which have greater elements on the following nodes. So, after deleting them, the linked list would like be 15, 11, 6, 3.

Input:  
LinkedList = 10->20->30->40->50->60  
Output: 60



Explanation: All the nodes except the last node has a greater value node on its right, so all the nodes except the last node must be removed.

```
6  */
7  class Solution {
8
9      Node reverse(Node head){
10         if(head == null || head.next == null){
11             return head;
12         }
13
14         Node res = reverse(head.next);
15         head.next.next = head;
16         head.next = null;
17         return res;
18     }
19     Node compute(Node head) {
20
21         if(head == null || head.next == null){
22             return head;
23         }
24
25         Node newhead = reverse(head);
26
27         Node prev = newhead;
28         Node curr = newhead.next;
29
30         while(curr != null){
31             if(prev.data > curr.data){
32                 prev.next = curr.next;
33                 curr = curr.next;
34             }else{
35                 prev = curr;
36                 curr = curr.next;
37             }
38         }
39         return reverse(newhead);
40     }
41 }
42 }
```

