

376. Wiggle Subsequence

July 30, 2016 | 39.5K views

Average Rating: 4.66 (77 votes)

A sequence of numbers is called a **wiggle sequence** if the differences between successive numbers strictly alternate between positive and negative. The first difference (if one exists) may be either positive or negative. A sequence with fewer than two elements is trivially a wiggle sequence.

For example, `[1,7,4,9,2,5]` is a wiggle sequence because the differences `(6,-3,5,-7,3)` are alternately positive and negative. In contrast, `[1,4,7,2,5]` and `[1,7,4,5,5]` are not wiggle sequences, the first because its first two differences are positive and the second because its last difference is zero.

Given a sequence of integers, return the length of the longest subsequence that is a wiggle sequence. A subsequence is obtained by deleting some number of elements (eventually, also zero) from the original sequence, leaving the remaining elements in their original order.

Example 1:

Input: `[1,7,4,9,2,5]`
Output: 6
Explanation: The entire sequence is a wiggle sequence.

Example 2:

Input: `[1,17,5,10,13,15,10,5,16,8]`
Output: 7
Explanation: There are several subsequences that achieve this length. One is `[1,17,10,13,15,10,5]`.

Example 3:

Input: `[1,2,3,4,5,6,7,8,9]`
Output: 2

Follow up:

Can you do it in $O(n)$ time?

Summary

We need to find the length of the longest wiggle subsequence. A wiggle subsequence consists of a subsequence with numbers which appears in alternating ascending / descending order.

Solution

Approach #1 Brute Force [Time Limit Exceeded]

Here, we can find the length of every possible wiggle subsequence and find the maximum length out of them. To implement this, we use a recursive function, `calculate(nums, index, isUp)` which takes the array `nums`, the index from which we need to find the length of the longest wiggle subsequence, boolean variable `isUp` to tell whether we need to find an increasing wiggle or decreasing wiggle respectively. If the function `calculate` is called after an increasing wiggle, we need to find the next decreasing wiggle with the same function. If the function `calculate` is called after a decreasing wiggle, we need to find the next increasing wiggle with the same function.

```

1 public class Solution {
2     private int calculate(int[] nums, int index, boolean isUp) {
3         int maxcount = 0;
4         for (int i = index + 1; i < nums.length; i++) {
5             if ((isUp && nums[i] > nums[index]) || (!isUp && nums[i] < nums[index]))
6                 maxcount = Math.max(maxcount, 1 + calculate(nums, i, !isUp));
7         }
8         return maxcount;
9     }
10
11     public int wiggleMaxLength(int[] nums) {
12         if (nums.length < 2)
13             return nums.length;
14         return 1 + Math.max(calculate(nums, 0, true), calculate(nums, 0, false));
15     }
16 }

```

Complexity Analysis

- Time complexity: $O(n!)$. `calculate()` will be called maximum $n!$ times.
- Space complexity: $O(n)$. Recursion of depth n is used.

Approach #2 Dynamic Programming [Accepted]

Algorithm

To understand this approach, take two arrays for `dp` named `up` and `down`.

Whenever we pick up any element of the array to be a part of the wiggle subsequence, that element could be a part of a rising wiggle or a falling wiggle depending upon which element we have taken prior to it.

`up[i]` refers to the length of the longest wiggle subsequence obtained so far considering i^{th} element as the last element of the wiggle subsequence and ending with a rising wiggle.

Similarly, `down[i]` refers to the length of the longest wiggle subsequence obtained so far considering i^{th} element as the last element of the wiggle subsequence and ending with a falling wiggle.

`up[i]` will be updated every time we find a rising wiggle ending with the i^{th} element. Now, to find `up[i]`, we need to consider the maximum out of all the previous wiggle subsequences ending with a falling wiggle i.e. `down[j]`, for every $j < i$ and `nums[i] > nums[j]`. Similarly, `down[i]` will be updated.

```

1 public class Solution {
2     public int wiggleMaxLength(int[] nums) {
3         if (nums.length < 2)
4             return nums.length;
5         int[] up = new int[nums.length];
6         int[] down = new int[nums.length];
7         for (int i = 1; i < nums.length; i++) {
8             for (int j = 0; j < i; j++) {
9                 if (nums[i] > nums[j]) {
10                     up[i] = Math.max(up[i], down[j] + 1);
11                 } else if (nums[i] < nums[j]) {
12                     down[i] = Math.max(down[i], up[j] + 1);
13                 }
14             }
15         }
16         return 1 + Math.max(down[nums.length - 1], up[nums.length - 1]);
17     }
18 }

```

Complexity Analysis

- Time complexity: $O(n^2)$. Loop inside a loop.
- Space complexity: $O(n)$. Two arrays of the same length are used for `dp`.

Approach #3 Linear Dynamic Programming [Accepted]

Algorithm

Any element in the array could correspond to only one of the three possible states:

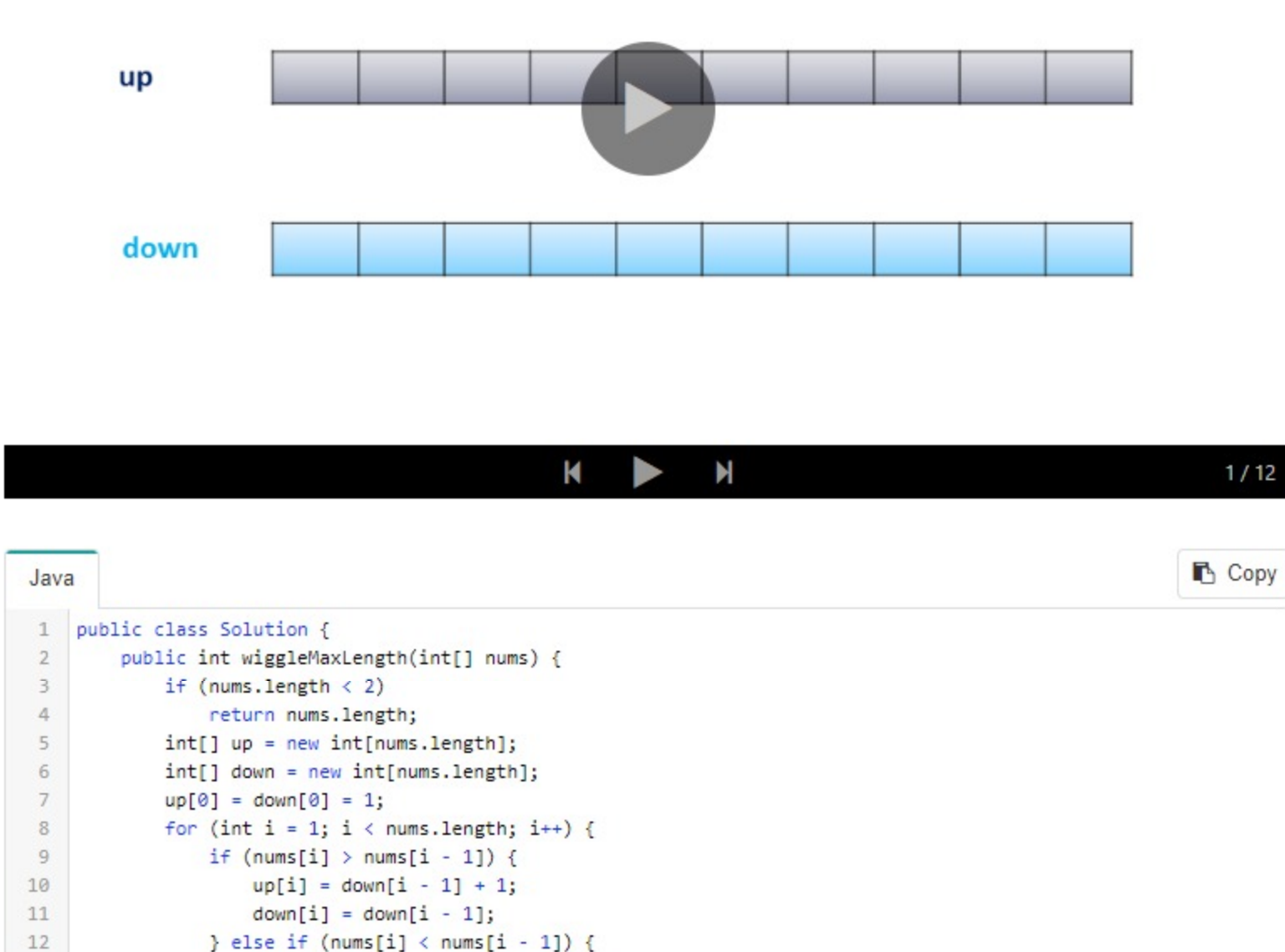
- up position, it means `nums[i] > nums[i - 1]`
- down position, it means `nums[i] < nums[i - 1]`
- equals to position, `nums[i] == nums[i - 1]`

The updates are done as:

If `nums[i] > nums[i - 1]`, that means it wiggles up. The element before it must be a down position. So `up[i] = down[i - 1] + 1`, `down[i]` remains the same as `down[i - 1]`. If `nums[i] < nums[i - 1]`, that means it wiggles down. The element before it must be a up position. So `down[i] = up[i - 1] + 1`, `up[i]` remains the same as `up[i - 1]`. If `nums[i] == nums[i - 1]`, that means it will not change anything because it didn't wiggle at all. So both `down[i]` and `up[i]` remain the same as `down[i - 1]` and `up[i - 1]`.

At the end, we can find the larger out of `up[length - 1]` and `down[length - 1]` to find the max. wiggle subsequence length, where `length` refers to the number of elements in the given array.

The process can be illustrated with the following example:



```

1 public class Solution {
2     public int wiggleMaxLength(int[] nums) {
3         if (nums.length < 2)
4             return nums.length;
5         int[] up = new int[nums.length];
6         int[] down = new int[nums.length];
7         up[0] = down[0] = 1;
8         for (int i = 1; i < nums.length; i++) {
9             if (nums[i] > nums[i - 1]) {
10                 up[i] = down[i - 1] + 1;
11                 down[i] = down[i - 1];
12             } else if (nums[i] < nums[i - 1]) {
13                 down[i] = up[i - 1] + 1;
14                 up[i] = up[i - 1];
15             } else {
16                 down[i] = down[i - 1];
17                 up[i] = up[i - 1];
18             }
19         }
20         return Math.max(down[nums.length - 1], up[nums.length - 1]);
21     }
22 }

```

Complexity Analysis

- Time complexity: $O(n)$. Only one pass over the array length.
- Space complexity: $O(n)$. Two arrays of the same length are used for `dp`.

Approach #4 Space-Optimized Dynamic Programming [Accepted]

Algorithm

This approach relies on the same concept as [Approach #3](#). But we can observe that in the DP approach, for updating elements `up[i]` and `down[i]`, we need only the elements `up[i - 1]` and `down[i - 1]`. Thus, we can save space by not using the whole array, but only the last elements.

```

1 public class Solution {
2     public int wiggleMaxLength(int[] nums) {
3         if (nums.length < 2)
4             return nums.length;
5         int down = 1, up = 1;
6         for (int i = 1; i < nums.length; i++) {
7             if (nums[i] > nums[i - 1])
8                 up = down + 1;
9             else if (nums[i] < nums[i - 1])
10                 down = up + 1;
11         }
12         return Math.max(down, up);
13     }
14 }

```

Complexity Analysis

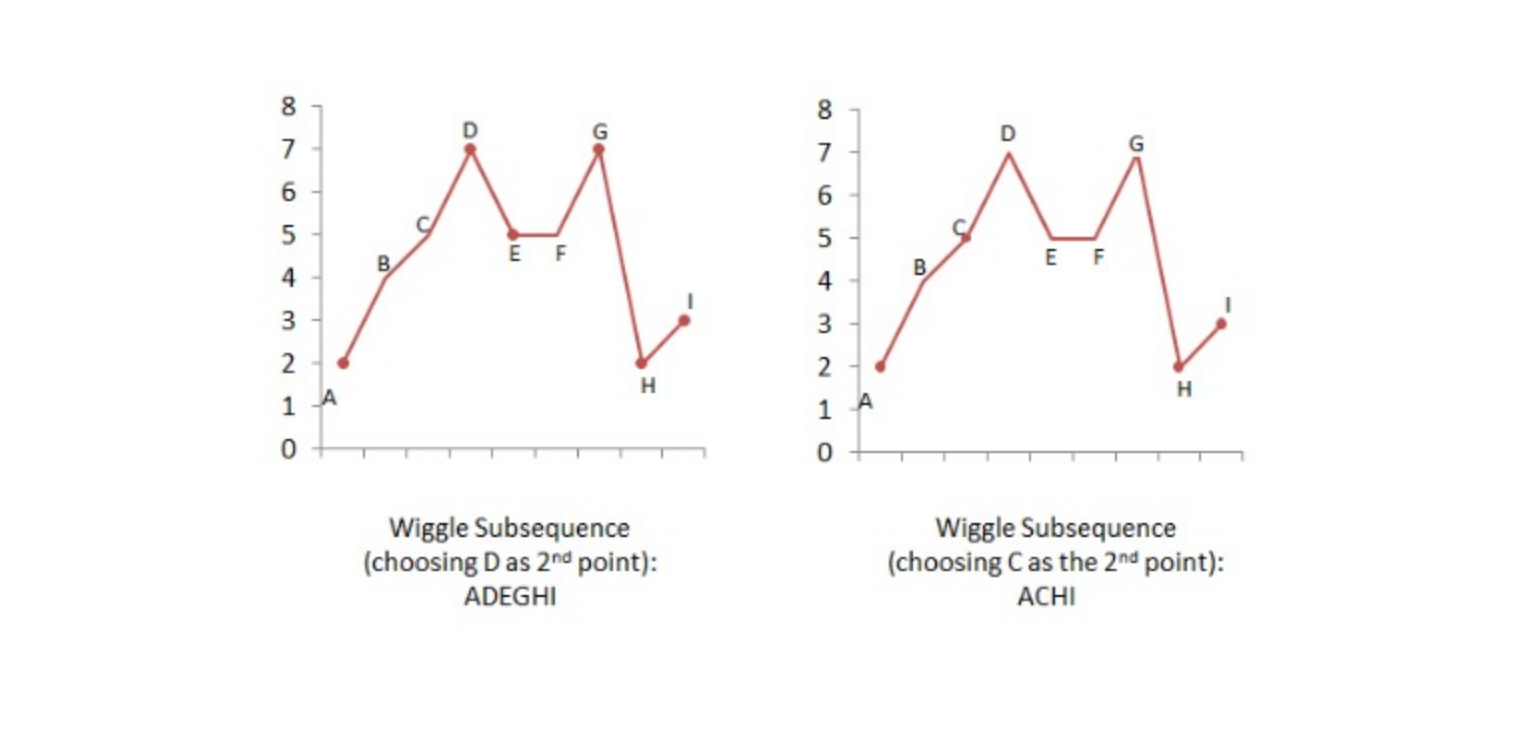
- Time complexity: $O(n)$. Only one pass over the array length.
- Space complexity: $O(1)$. Constant space is used.

Approach #5 Greedy Approach [Accepted]

Algorithm

We need not necessarily need `dp` to solve this problem. This problem is equivalent to finding the number of alternating max. and min. peaks in the array. Since, if we choose any other intermediate number to be a part of the current wiggle subsequence, the maximum length of that wiggle subsequence will always be less than or equal to the one obtained by choosing only the consecutive max. and min. elements.

This can be clarified by looking at the following figure:



From the above figure, we can see that if we choose **D** as the 2nd point in the wiggle subsequence, we can't include the point **E**. Thus, we won't obtain the maximum length wiggle subsequence.

Thus, to solve this problem, we maintain a variable `prevalldiff`, where `prevalldiff` is used to indicate whether the current subsequence of numbers lies in an increasing or decreasing wiggle. If `prevalldiff > 0`, it indicates that we have found the increasing wiggle and are looking for a decreasing wiggle now. Thus, we update the length of the found subsequence when `diff (nums[i] - nums[i - 1])` becomes negative. Similarly, if `prevalldiff < 0`, we will update the count when `diff (nums[i] - nums[i - 1])` becomes positive.

When the complete array has been traversed, we get the required count, which represents the length of the longest wiggle subsequence.

```

1 public class Solution {
2     public int wiggleMaxLength(int[] nums) {
3         if (nums.length < 2)
4             return nums.length;
5         int prevalldiff = nums[1] - nums[0];
6         int count = prevalldiff != 0 ? 2 : 1;
7         for (int i = 2; i < nums.length; i++) {
8             int diff = nums[i] - nums[i - 1];
9             if ((diff > 0 && prevalldiff <= 0) || (diff < 0 && prevalldiff >= 0)) {
10                 count++;
11                 prevalldiff = diff;
12             }
13         }
14         return count;
15     }
16 }

```

Complexity Analysis

- Time complexity: $O(n)$. We traverse the given array once.
- Space complexity: $O(1)$. No extra space is used.

Rate this article: ★★★★★

Previous Next

Comments: 30 Sort By

Type comment here... (Markdown is supported)

Preview Post

markov_r ★219 December 3, 2018 5:32 PM

For the above figure, in brief:

When you have increasing or decreasing sub-sequence longer than 2 just ignore all middle elements and use the first and the last only (you don't gain anything from the middle ones). Also ignore duplicates.

So at the end you'll have a "clean" wiggles sequence with the required length.

19 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 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040 1041 1042 1043 1044 1045 1046 1047 1048 1049 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062 1063 1064 1065 1066 1067 1068 1069 1070 1071 1072 1073 1074 1075 1076 1077 1078 1079 1080 1081 1082 1083 1084 1085 1086 1087 1088 1089 1090 1091 1092 1093 1094 1095 1096 1097 1098 1099 1100 1101 1102 1103 1104 1105 1106 1107 1108 1109 1110 1111 1112 1113 1114 1115 1116 1117 1118 1119 1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142 1143 1144 1145 1146 1147 1148 1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160 1161 1162 1163 1164 1165 1166 1167 1168 1169 1170 1171 1172 1173 1174 1175 1176 1177 1178 1179 1180 1181 1182 1183 1184 1185 1186 1187 1188 1189 1190 1191 1192 1193 1194 1195 1196 1197 1198 1199 1200 1201 1202 1203 1204 1205 1206 1207 1208 1209 1210 1211 1212 1213 1214 1215 1216 1217 1218 1219 1220 1221 1222 1223 1224 1225 1226 1227 1228 1229 1230 1231 1232 1233 1234 1235 1236 1237 1238 1239 1240 1241 1242 1243 1244 1245 1246 1247 1248 1249 1250 1251 1252 1253 1254 1255 1256 1257 1258 1259 1260 1261 1262 1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1280 1281 1282 1283 1284 1285 1286 1287 1288 1289 1290 1291 1292 1293 1294 1295 1296 1297 1298 1299 1300 1301 1302 1303 1304 1305 1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320 1321 1322 1323 1324 1325 1326 1327 1328 1329 1330 1331 1332 1333 1334 1335 1336 1337 1338 1339 1340 1341 1342 1343 1344 1345 1346 1347 1348 1349 1350 1351 1352 1353 1354 1355 1356 1357 1358 1359 1360 1361 1362 1363 1364 1365 1366 1367 1368 1369 1370 1371 1372 1373 1374 1375 1376 1377 1378 1379 1380 1381 1382 1383 1384 1385 1386 1387 1388 1389 1390 1391 1392 1393 1394 1395 1396 1397 1398 1399 1400 1401 1402 1403 1404 1405 1406 1407 1408 1409 1410 1411 1412 1413 1414 1415 1416 1417 1418 1419 1420 1421 1422 1423 1