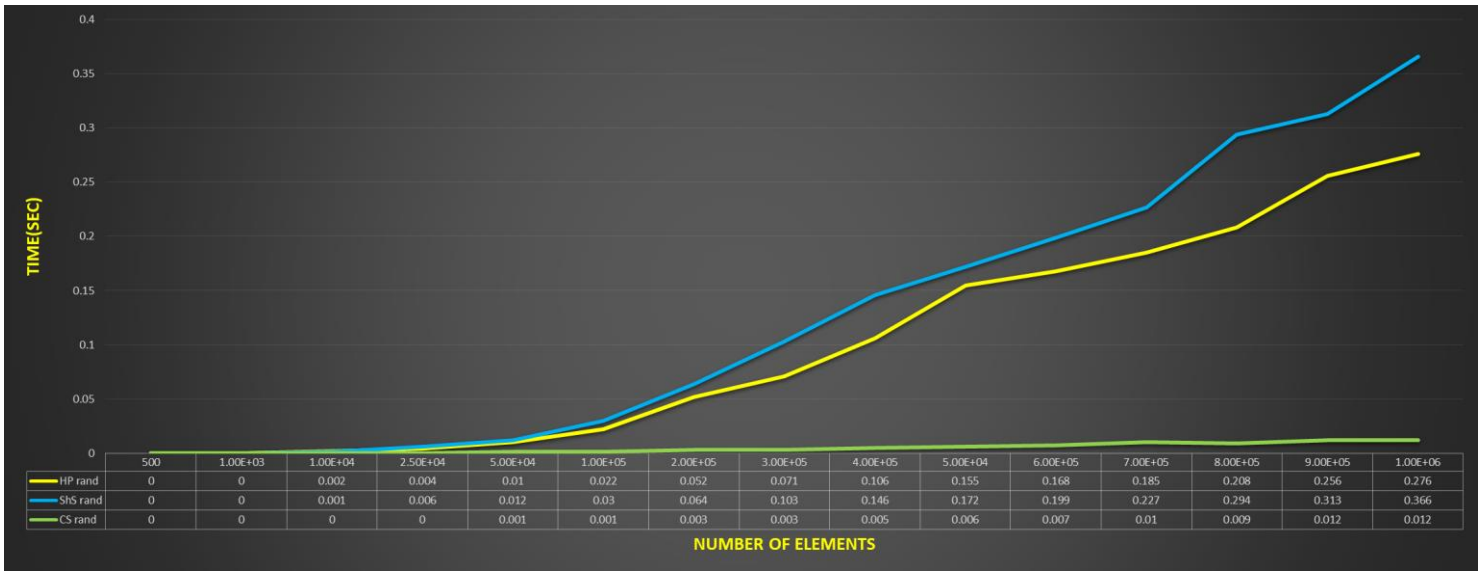
#Exercise 1 Sorting algorithms

Uladzimir Ivashka, 150281

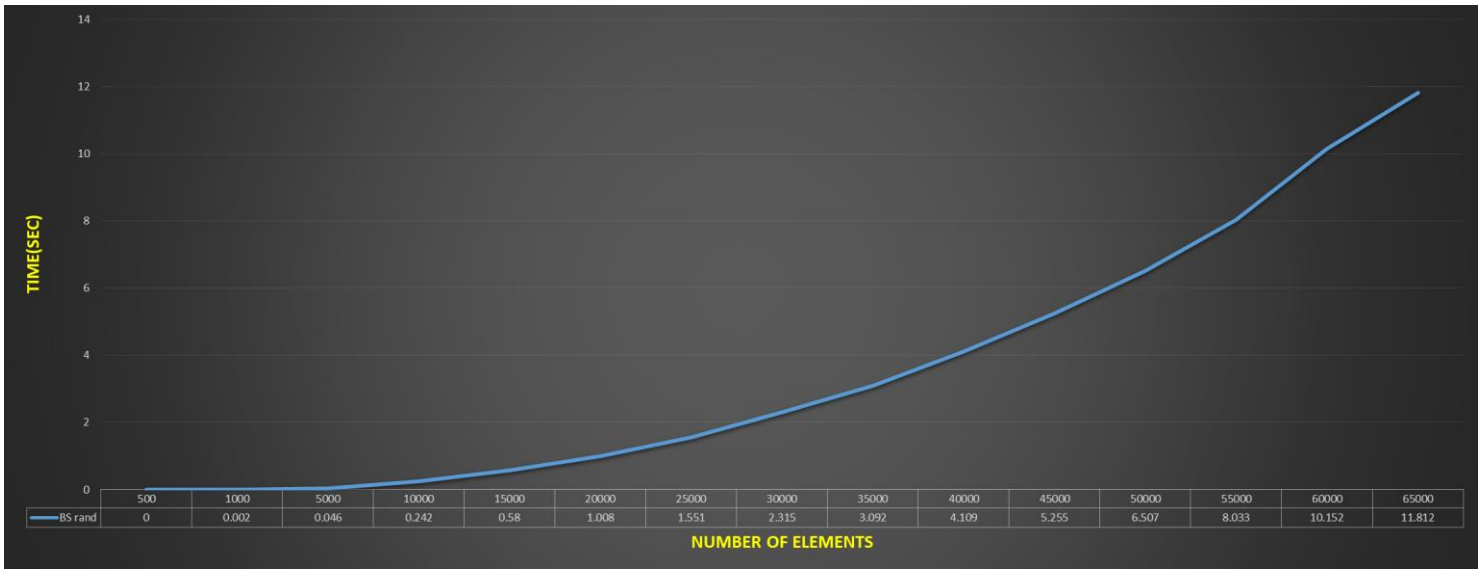
Sofya Aksenyuk, 150284

Exercise I.

1. Comparison of the speed of 4 sorting methods: BS, HS, CS, ShS for the array of integers randomly generated according to the uniform probability distribution:



	500	1.00E+03	1.00E+04	2.50E+04	5.00E+04	1.00E+05	2.00E+05	3.00E+05	4.00E+05	5.00E+04	6.00E+05	7.00E+05	8.00E+05	9.00E+05	1.00E+06
HP rand	0	0	0.002	0.004	0.01	0.022	0.052	0.071	0.106	0.155	0.168	0.185	0.208	0.256	0.276
ShS rand	0	0	0.001	0.006	0.012	0.03	0.064	0.103	0.146	0.172	0.199	0.227	0.294	0.313	0.366
CS rand	0	0	0	0	0.001	0.001	0.003	0.003	0.005	0.006	0.007	0.01	0.009	0.012	0.012



	500	1000	5000	10000	15000	20000	25000	30000	35000	40000	45000	50000	55000	60000	65000
BS rand	0	0.002	0.046	0.242	0.58	1.008	1.551	2.315	3.092	4.109	5.255	6.507	8.033	10.152	11.812

2. Conclusions:

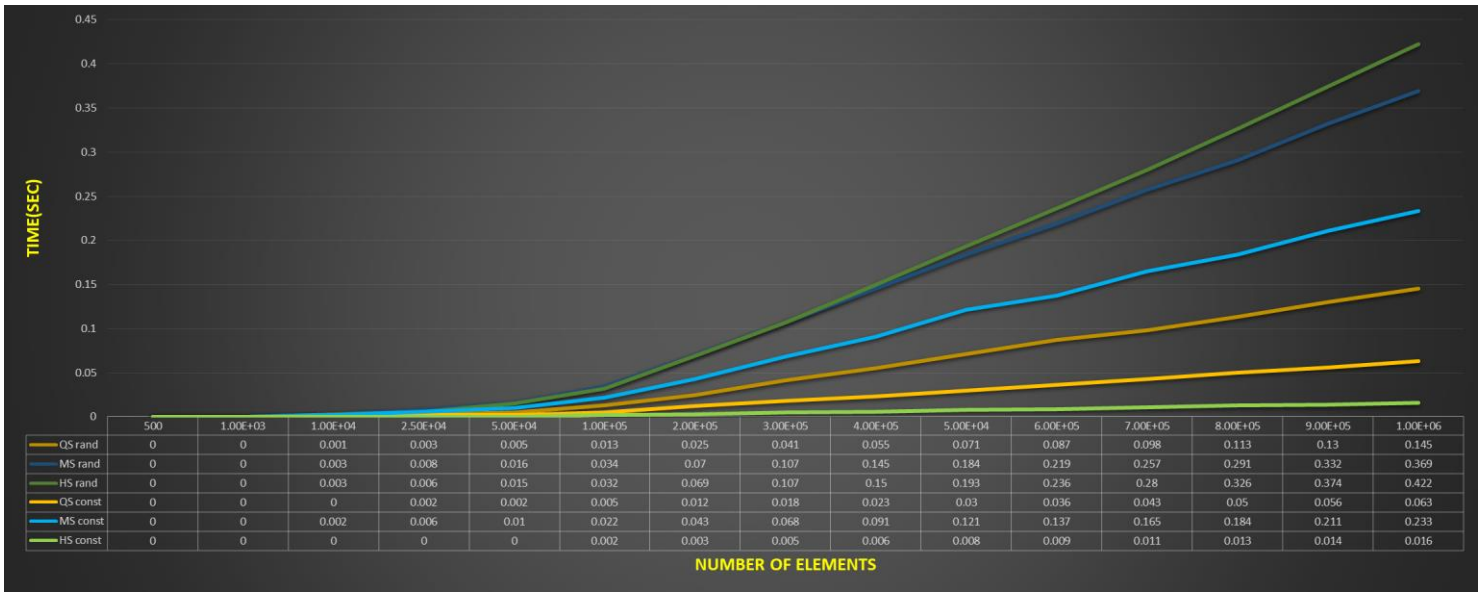
According to the chart above, it can be seen that Bubble Sort is the least efficient algorithm among all the other presented ones. BS has average and worst cases equal to O(n^2) and best case of O(n) complexity but best one happens only in case of evaluation of already sorted initial array. That is the reason why supplied computations for that method are that restricted and presented on the separate graph. It is mostly used for educational purposes as a way to teach sorting as a basic and the most understandable approach to do that.

The middle time execution values belong to Heap and Shell sorting methods. Nevertheless, HeapSort is more stable due to the same time complexity for each case (O(nlogn)), whereas ShellSort has average and worst cases equal to O(nlogn^2).

The best results belong to CountingSort that are slightly greater than zero for all presented measuring points (for this particular computations) which can be explained by the complexity of O(n+k). However, this sorting method is much more memory consuming comparing to other ones (O(k) to O(1) respectively).

Exercise II.

1. Chart for random (uniform distribution) and constant value (e.g., equal to 0):

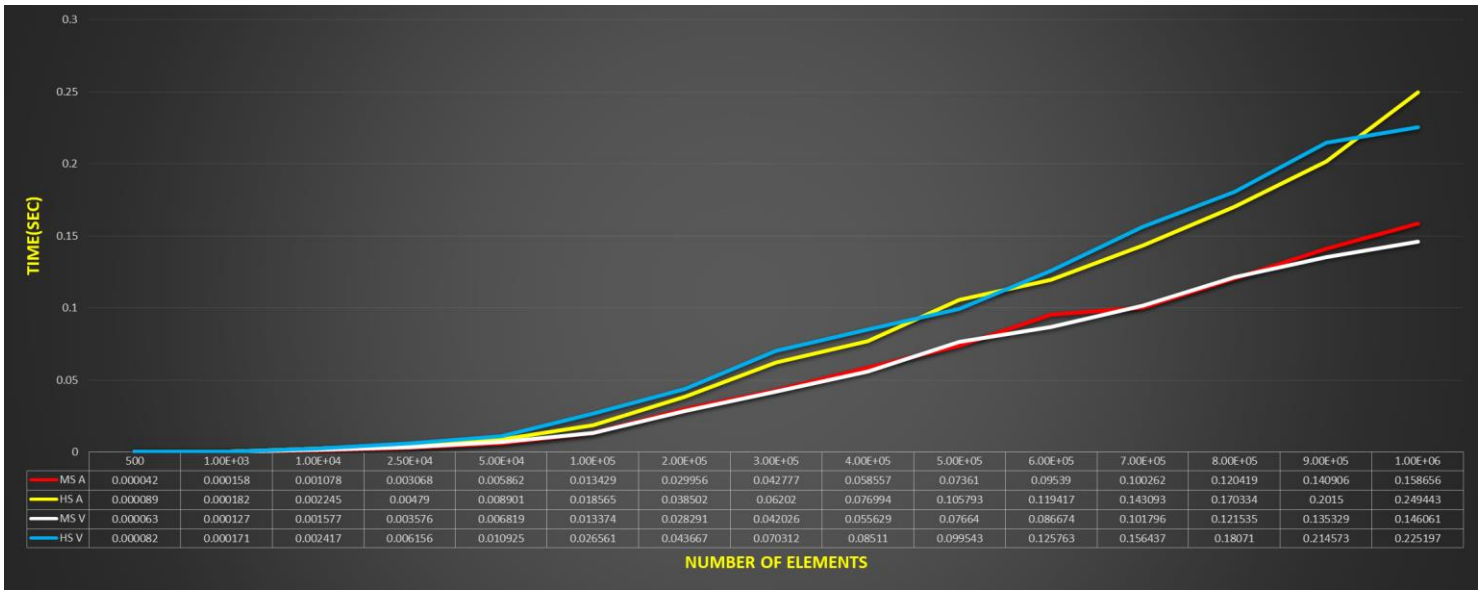


	500	1.00E+03	1.00E+04	2.50E+04	5.00E+04	1.00E+05	2.00E+05	3.00E+05	4.00E+05	5.00E+04	6.00E+05	7.00E+05	8.00E+05	9.00E+05	1.00E+06
QS rand	0	0	0.001	0.003	0.005	0.013	0.025	0.041	0.055	0.071	0.087	0.098	0.113	0.13	0.145
MS rand	0	0	0.003	0.008	0.016	0.034	0.07	0.107	0.145	0.184	0.219	0.257	0.291	0.332	0.369
HS rand	0	0	0.003	0.006	0.015	0.032	0.069	0.107	0.15	0.193	0.236	0.28	0.326	0.374	0.422
QS const	0	0	0	0.002	0.002	0.005	0.012	0.018	0.023	0.03	0.036	0.043	0.05	0.056	0.063
MS const	0	0	0.002	0.006	0.01	0.022	0.043	0.068	0.091	0.121	0.137	0.165	0.184	0.211	0.233
HS const	0	0	0	0	0	0.002	0.003	0.005	0.006	0.008	0.009	0.011	0.013	0.014	0.016

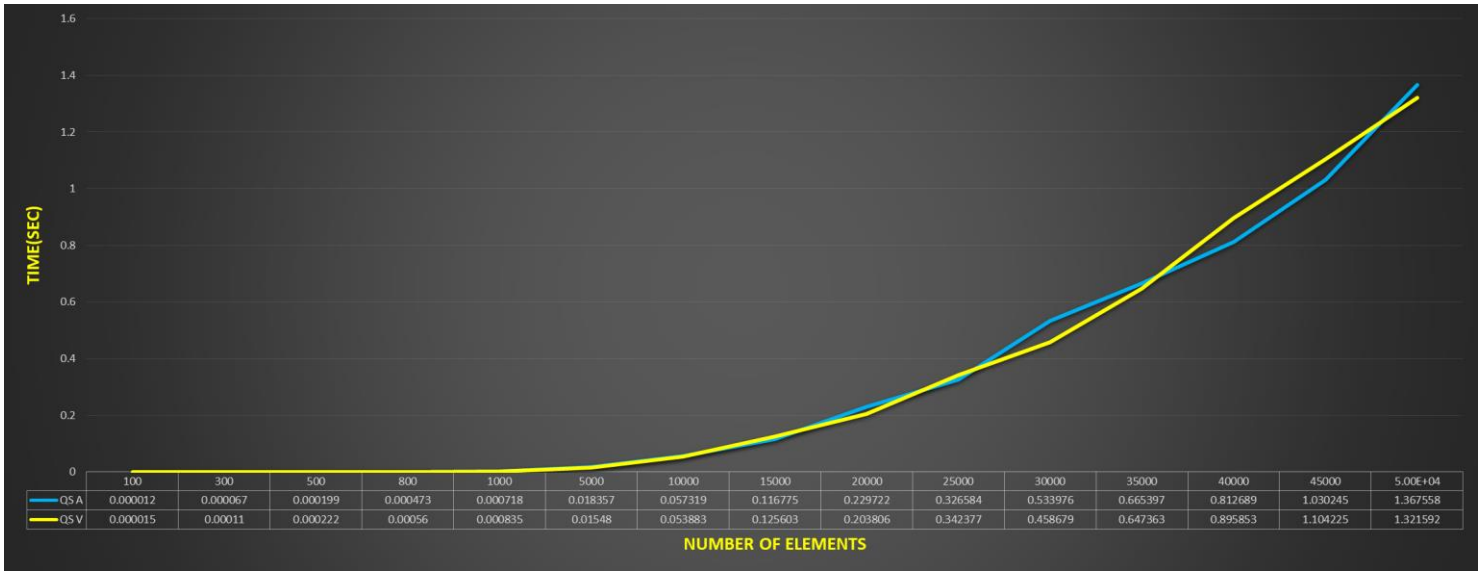
Chart for increasing order (step equal to 1) and descending order (step equal to 1):

	500	1.00E+03	1.00E+04	2.50E+04	5.00E+04	1.00E+05	2.00E+05	3.00E+05	4.00E+05	5.00E+04	6.00E+05	7.00E+05	8.00E+05	9.00E+05	1.00E+06
QS inc	0	0	0.001	0.002	0.003	0.006	0.011	0.018	0.025	0.032	0.038	0.046	0.053	0.059	0.067
MS inc	0	0	0.002	0.006	0.01	0.02	0.044	0.068	0.091	0.114	0.14	0.162	0.187	0.209	0.239
HS inc	0	0	0.001	0.005	0.011	0.023	0.049	0.075	0.103	0.13	0.161	0.189	0.218	0.25	0.28
QS dec	0	0	0	0.001	0.003	0.005	0.012	0.017	0.024	0.03	0.037	0.045	0.056	0.06	0.066
MS dec	0	0	0.002	0.005	0.011	0.022	0.045	0.07	0.094	0.12	0.141	0.174	0.191	0.218	0.248
HS dec	0	0	0.002	0.006	0.011	0.022	0.047	0.072	0.1	0.129	0.156	0.184	0.216	0.244	0.274

Chart for ascending-descending order (A shape – increase odd numbers - decrease even) and descending-ascending order (V -shape – decrease odd numbers - increase even):



	500	1.00E+03	1.00E+04	2.50E+04	5.00E+04	1.00E+05	2.00E+05	3.00E+05	4.00E+05	5.00E+05	6.00E+05	7.00E+05	8.00E+05	9.00E+05	1.00E+06
MS A	0.000042	0.000158	0.001078	0.003068	0.005862	0.013429	0.029956	0.042777	0.058557	0.07361	0.09539	0.100262	0.120419	0.140906	0.158656
HS A	0.000089	0.000182	0.002245	0.00479	0.008901	0.018565	0.038502	0.06202	0.076994	0.105793	0.119417	0.143093	0.170334	0.2015	0.249443
MS V	0.000063	0.000127	0.001577	0.003576	0.006819	0.013374	0.028291	0.042026	0.055629	0.07664	0.086674	0.101796	0.121535	0.135329	0.146061
HS V	0.000082	0.000171	0.002417	0.006156	0.010925	0.026561	0.043667	0.070312	0.08511	0.099543	0.125763	0.156437	0.18071	0.214573	0.225197



	100	300	500	800	1000	5000	10000	15000	20000	25000	30000	35000	40000	45000	5.00E+04
QS A	0.000012	0.000067	0.000199	0.000473	0.000718	0.018357	0.057319	0.116775	0.229722	0.326584	0.533976	0.665397	0.812689	1.030245	1.367558
QS V	0.000015	0.00011	0.000222	0.00056	0.000835	0.01548	0.053883	0.125603	0.203806	0.342377	0.458679	0.647363	0.895853	1.104225	1.321592

2. Conclusions on the computational complexity and computational efficiency of QS execution:

When it comes to random type of data, QS shows the best results comparing with MS and HS, despite the fact that in average cases their time complexities are equal (O(nlogn)). Nevertheless, QS evaluates constant value sequence faster than MS but slower than HS. Constant value sequence would be worst case for QS (O(n^2)) if pivot would be either first or last element of an array, but presented results were executed by implementing QS with middle element selected as pivot which is why it's quite fast work as expected. As for increasing-descending ordered data, QS has the best time complexity overall (both for increasing and decreasing types of initial sequences), which proves that it is best case of QS (i.e., O(nlogn)).

Remark: the most memory-consuming sorting algorithms are MS(O(n)) and CS(O(k)), the middle space complexity belongs to QS(O(logn)) but BS and ShS have least consuming complexity of O(1).

Worst scenarios of QS:

The median effect on QS sorting time is that it guarantees that an array will be splited into two approximately equal parts. We cannot be sure about implementing other kinds of pivot selection (first/last element), because it may come up with a pivot element near to the highest or the lowest number that would make almost empty subarray out of one of the splited arrays which would lead to time complexity of O(n^2).

Nevertheless, middle element as pivot leads to worst cases executing A- and V-shaped arrays which happens due to respectively largest/smallest value of an array immediately selected as pivot. Therefore, according to the concept of QS work – bugs appear at points of comparison (i.e., there are no greater values to the left in case of A-shape and no less values for V-shape).

The table for QS algorithm for specified requirements:

(pivot – middle element)

QS	49'999	50'000	50'001
A-shape	1.4175	1.627823	1.550437
V-shape	1.634428	1.500488	1.790733
Random	0.004778	0.005649	0.00628

As it could be observed, A-shape and V-shape types of data structures form some kind of shapes of A and V, respectively. Such time complexity could be explained by the way such arrays are formed and middle element selected as pivot (as it is mentioned above) - the algorithm bugs and time complexity gets worse when it faces the middle point.

Results of algorithms work for n = 100:

BS: Input: 41 467 334 500 169 724 478 358 962 464 705 145 281 827 961 491 995 942 827 436 391 604 902 153 292 382 421 716 718 895 447 726 771 538 869 912 667 299 35 894 703 811 322 333 673 664 141 711 253 868 547 644 662 757 37 859 723 741 529 778 316 35 190 842 288 106 40 942 264 648 446 805 890 729 370 350 6 101 393 548 629 623 84 954 756 840 966 376 931 308 944 439 626 323 537 538 118 82 929 541

Output: 6 35 35 37 40 41 82 84 101 106 118 141 145 153 169 190 253 264 281 288 292 299 308 316 322 323 333 334 350 358 370 376 382 391 393 421 436 439 446 447 464 467 478 491 500 529 537 538 538 541 547 548 604 623 626 629 644 648 662 664 667 673 703 705 711 716 718 723 724 726 729 741 756 757 771 778 805 811 827 827 840 842 859 868 869 890 894 895 902 912 929 931 942 942 944 954 961 962 966 995

HS: Input: 154 773 286 810 382 755 938 283 714 371 309 173 172 682 10 757 566 122 415 300 417 634 932 774 672 179 126 759 400 342 612 759 638 764 234 328 26 462 748 5 330 439 157 685 434 654 49 496 563 456 248 817 865 757 693 509 210 994 645 363 772 455 275 164 405 786 307 981 337 412 85 843 296 458 989 350 782 895 465 73 591 48 539 496 378 336 824 763 437 460 223 755 338 146 494 587 304 254 450 753

Output: 5 10 26 48 49 73 85 122 126 146 154 157 164 172 173 179 210 223 234 248 254 275 283 286 296 300 304 307 309 328 330 336 337 338 342 350 363 371 378 382 400 405 412 415 417 434 437 439 450 455 456 458 460 462 465 494 496 496 509 539 563 566 587 591 612 634 638 645 654 672 682 685 693 714 748 753 755 755 757 757 759 759 763 764 772 773 774 782 786 810 817 824 843 865 895 932 938 981 989 994

CS: Input: 261 790 980 467 315 266 577 987 308 93 428 626 859 655 456 858 824 331 292 951 913 985 133 880 818 671 412 401 488 689 339 144 610 527 502 390 329 190 16 217 730 161 147 448 706 578 420 727 457 603 726 275 154 466 950 672 538 357 207 905 87 562 991 807 523 58 34 415 569 854 613 685 911 986 733 813 116 770 769 779 836 915 69 409 826 787 649 170 941 275 74 820 438 851 911 251 97 663 158 140

Output: 16 34 58 69 74 87 93 97 116 133 140 144 147 154 158 161 170 190 207 217 251 261 266 275 275 292 308 315 329 331 339 357 390 401 409 412 415 420 428 438 448 456 457 466 467 488 502 523 527 538 562 569 577 578 603 610 613 626 649 655 663 671 672 685 689 706 726 727 730 733 769 770 779 787 790 807 813 818 820 824 826 836 851 854 858 859 880 905 911 911 913 915 941 950 951 980 985 986 987 991

ShS: Input: 349 85 560 816 665 983 77 254 369 935 203 816 785 422 820 466 13 524 186 757 219 986 318 666 882 346 687 260 386 30 829 12 532 46 462 399 298 135 577 571 106 592 538 834 216 718 982 518 610 171 999 525 447 970 778 575 498 94 422 294 660 739 136 12 520 189 5 28 107 272 583 746 588 824 348 596 432 58 939 685 345 491 545 856 247 847 499 98 535 556 461 147 436 833 930 454 459 201 940 30

Output: 5 12 12 13 28 30 30 46 58 77 85 94 98 106 107 135 136 147 171 186 189 201 203 216 219 247 254 260 272 294 298 318 345 346 348 349 369 386 399 422 422 432 436 447 454 459 461 462 466 491 498 499 518 520 524 525 532 535 538 545 556 560 571 575 577 583 588 592 596 610 660 665 666 685 687 718 739 746 757 778 785 816 816 820 824 829 833 834 847 856 882 930 935 939 940 970 982 983 986 999

MS: Input: 470 560 407 423 88 432 166 249 706 832 730 445 723 44 730 384 471 897 386 201 609 725 810 62 136 305 671 269 677 613 792 639 692 381 115 932 825 51 161 891 189 510 927 617 329 497 71 684 130 294 614 944 966 381 631 578 174 719 240 427 548 976 436 383 486 733 81 342 236 986 970 419 31 223 999 459 266 836 100 104 414 242 599 567 224 148 974 567 902 726 721 860 962 533 99 658 693 447 87 261

Output: 31 44 51 62 71 81 87 88 99 100 104 115 130 136 148 161 166 174 189 201 223 224 236 240 242 249 261 266 269 294 305 329 342 381 381 383 384 386 407 414 419 423 427 432 436 445 447 459 470 471 486 497 510 533 548 560 567 567 578 599 609 613 614 617 631 639 658 671 677 684 692 693 706 719 721 723 725 726 730 730 733 792 810 825 832 836 860 891 897 902 927 932 944 962 966 970 974 976 986 999

QS: Input: 29 453 477 983 103 416 267 754 244 84 837 100 727 635 644 508 47 946 758 78 371 167 751 626 802 434 136 19 842 398 702 867 408 984 133 237 639 351 420 731 692 92 610 121 754 660 233 681 313 179 41 241 877 732 945 538 43 958 906 866 391 609 244 877 772 418 22 887 308 163 271 773 841 758 666 668 218 75 682 250 770 728 802 860 582 529 408 896 558 288 155 908 173 846 451 966 60 835 400 589

Output: 19 22 29 41 43 47 60 75 78 84 92 100 103 121 133 136 155 163 167 173 179 218 233 237 241 244 244 250 267 271 288 308 313 351 371 391 398 400 408 408 416 418 420 434 451 453 477 508 529 538 558 582 589 609 610 626 635 639 644 660 666 668 681 682 692 702 727 728 731 732 751 754 754 758 758 770 772 773 802 802 835 837 841 842 846 860 866 867 877 877 887 896 906 908 945 946 958 966 983 984