Luka Cassar

Statement of Completion

All questions 1 throughout 12 were attempted and all work correctly as intended. Use of AI was stated under **How this was tested** for that corresponding question.

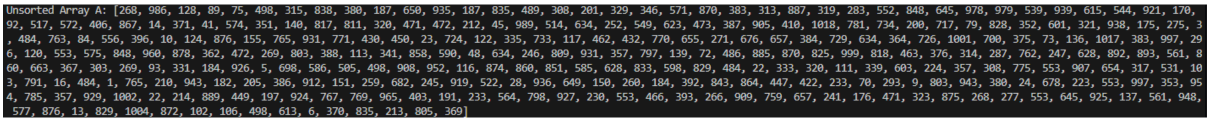
Question: 1

Source Code:

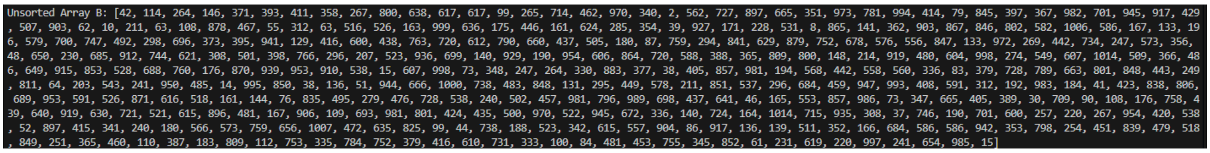
```
import random
arrA = []
sizeA = 300
arrB = []
sizeB = 400
for size in range(sizeA):
    arrA.append(random.randint(0,1024))
for size in range(sizeB):
    arrB.append(random.randint(0,1024))
temp = 0
print(f"Unsorted Array A: {arrA}")
print(f"Unsorted Array B: {arrB}")
# Shell Sort
gap = sizeA // 2
flag = True
while gap >= 1 and flag == True:
    flag = False
    for i in range(sizeA - gap):
        if arrA[i] > arrA[i + gap]:
            temp = arrA[i]
            arrA[i] = arrA[i + gap]
            arrA[i + gap] = temp
            flag = True
    if gap > 1:
        gap = gap // 2
# Quick Sort
def qSort(arr):
    quick(arr, 0, len(arr) - 1)
def quick(arr, first, last):
    if first < last:
        pivotPos = partition(arr, first, last)
        quick(arr, first, pivotPos - 1)
        quick(arr, pivotPos + 1, last)
def partition(arr, first, last):
    pivot = arr[first]
    u = first
    d = last
    while True:
        while u < last and arr[u] <= pivot:
            u += 1
        while arr[d] > pivot:
            d -= 1
        if u < d:
            temp = arr[u]
            arr[u] = arr[d]
            arr[d] = temp
        else:
            break
    temp = arr[first]
```

```
        arr[first] = arr[d]
        arr[d] = temp
        return d
qSort(arrB)
print(f"Sorted Array A: {arrA}")
print(f"Sorted Array B: {arrB}")
```

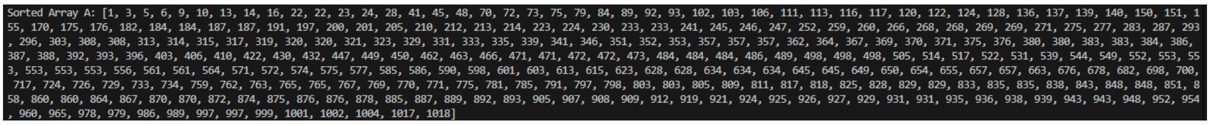
Sample Screen Dumps:



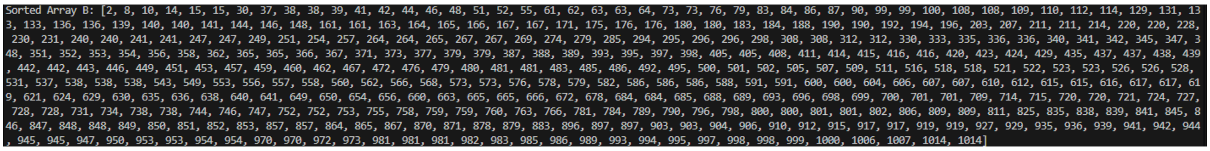
```
Unsorted Array A: [268, 986, 128, 89, 75, 498, 315, 838, 380, 187, 650, 935, 187, 835, 489, 308, 201, 329, 346, 571, 870, 383, 313, 887, 319, 283, 552, 848, 645, 978, 979, 539, 939, 615, 544, 921, 170,
92, 517, 572, 406, 867, 14, 371, 41, 574, 351, 140, 817, 811, 320, 471, 472, 212, 45, 989, 514, 634, 252, 549, 623, 473, 387, 905, 410, 1018, 781, 734, 200, 717, 79, 828, 352, 601, 321, 938, 175, 275, 3
, 484, 763, 84, 556, 396, 10, 124, 876, 155, 765, 931, 771, 430, 450, 23, 724, 122, 335, 733, 117, 462, 432, 770, 655, 271, 676, 657, 384, 729, 634, 364, 726, 1001, 700, 375, 73, 136, 1017, 383, 997, 29
6, 120, 553, 575, 848, 960, 878, 362, 472, 269, 803, 388, 113, 341, 858, 590, 48, 634, 246, 809, 931, 357, 797, 139, 72, 486, 885, 870, 825, 999, 818, 463, 376, 314, 287, 762, 247, 628, 892, 893, 561, 8
60, 663, 367, 303, 269, 93, 331, 184, 926, 5, 698, 586, 505, 498, 908, 952, 116, 874, 860, 851, 585, 628, 833, 598, 829, 484, 22, 333, 320, 111, 339, 603, 224, 357, 308, 775, 553, 907, 654, 317, 531, 10
3, 791, 16, 484, 1, 765, 210, 943, 182, 205, 386, 912, 151, 259, 682, 245, 919, 522, 28, 936, 649, 150, 260, 184, 392, 843, 864, 447, 422, 233, 70, 293, 9, 803, 943, 380, 24, 678, 223, 553, 997, 353, 95
4, 785, 357, 929, 1002, 22, 214, 889, 449, 197, 924, 767, 769, 965, 403, 191, 233, 564, 798, 927, 230, 553, 466, 393, 266, 909, 759, 657, 241, 176, 471, 323, 875, 268, 277, 553, 645, 925, 137, 561, 948,
 577, 876, 13, 829, 1004, 872, 102, 106, 498, 613, 6, 370, 835, 213, 805, 369]
```



```
Unsorted Array B: [42, 114, 264, 146, 371, 393, 411, 358, 267, 800, 638, 617, 617, 99, 265, 714, 462, 970, 340, 2, 562, 727, 897, 665, 351, 973, 781, 994, 414, 79, 845, 397, 367, 982, 701, 945, 917, 429
, 507, 903, 62, 10, 211, 63, 108, 878, 467, 55, 312, 63, 516, 526, 163, 999, 636, 175, 446, 161, 624, 285, 354, 39, 927, 171, 228, 531, 8, 865, 141, 362, 903, 867, 846, 802, 582, 1006, 586, 167, 133, 19
6, 579, 700, 747, 492, 298, 696, 373, 395, 941, 129, 416, 600, 438, 763, 720, 612, 790, 660, 437, 505, 180, 87, 759, 294, 841, 629, 879, 752, 678, 576, 556, 847, 133, 972, 269, 442, 734, 247, 573, 356,
48, 650, 230, 685, 912, 744, 621, 308, 501, 398, 766, 296, 207, 523, 936, 699, 140, 929, 190, 954, 606, 864, 720, 588, 388, 365, 809, 800, 148, 214, 919, 480, 604, 998, 274, 549, 607, 1014, 509, 366, 48
6, 649, 915, 853, 528, 688, 760, 176, 870, 939, 953, 910, 538, 15, 607, 998, 73, 348, 247, 264, 330, 883, 377, 38, 405, 857, 981, 194, 568, 442, 558, 560, 336, 83, 379, 728, 789, 663, 801, 848, 443, 249
, 811, 64, 203, 543, 241, 950, 485, 14, 995, 850, 38, 136, 51, 944, 666, 1000, 738, 483, 848, 131, 295, 449, 578, 211, 851, 537, 296, 684, 459, 947, 993, 408, 591, 312, 192, 983, 184, 41, 423, 838, 806,
 689, 953, 591, 526, 871, 616, 518, 161, 144, 76, 835, 495, 279, 476, 728, 538, 240, 502, 457, 981, 796, 989, 698, 437, 641, 46, 165, 553, 857, 986, 73, 347, 665, 405, 389, 30, 709, 90, 108, 176, 758, 4
39, 640, 919, 630, 721, 521, 615, 896, 481, 167, 906, 109, 693, 981, 801, 424, 435, 500, 970, 522, 945, 672, 336, 140, 724, 164, 1014, 715, 935, 308, 37, 746, 190, 701, 600, 257, 220, 267, 954, 420, 538
, 52, 897, 415, 341, 240, 180, 566, 573, 759, 656, 1007, 472, 635, 825, 99, 44, 738, 188, 523, 342, 615, 557, 904, 86, 917, 136, 139, 511, 352, 166, 684, 586, 586, 942, 353, 798, 254, 451, 839, 479, 518
, 849, 251, 365, 460, 110, 387, 183, 809, 112, 753, 335, 784, 752, 379, 416, 610, 731, 333, 100, 84, 481, 453, 755, 345, 852, 61, 231, 619, 220, 997, 241, 654, 985, 15]
```



```
Sorted Array A: [1, 3, 5, 6, 9, 10, 13, 14, 16, 22, 22, 23, 24, 28, 41, 45, 48, 70, 72, 73, 75, 79, 84, 89, 92, 93, 102, 103, 106, 111, 113, 116, 117, 120, 122, 124, 128, 136, 137, 139, 140, 150, 151, 1
55, 170, 175, 176, 182, 184, 184, 187, 187, 191, 197, 200, 201, 205, 210, 212, 213, 214, 223, 224, 230, 233, 233, 241, 245, 246, 247, 252, 259, 260, 266, 268, 268, 269, 269, 271, 275, 277, 283, 287, 293
, 296, 303, 308, 308, 313, 314, 315, 317, 319, 320, 320, 321, 323, 329, 331, 333, 335, 339, 341, 346, 351, 352, 353, 357, 357, 357, 362, 364, 367, 369, 370, 371, 375, 376, 380, 380, 383, 383, 384, 386,
387, 388, 392, 393, 396, 403, 406, 410, 422, 430, 432, 447, 449, 450, 462, 463, 466, 471, 471, 472, 472, 473, 484, 484, 484, 486, 489, 498, 498, 498, 505, 514, 517, 522, 531, 539, 544, 549, 552, 553, 55
3, 553, 553, 553, 556, 561, 561, 564, 571, 572, 574, 575, 577, 585, 586, 590, 598, 601, 603, 613, 615, 623, 628, 628, 634, 634, 634, 645, 645, 649, 650, 654, 655, 657, 657, 663, 676, 678, 682, 698, 700,
 717, 724, 726, 729, 733, 734, 759, 762, 763, 765, 765, 767, 769, 770, 771, 775, 781, 785, 791, 797, 798, 803, 803, 805, 809, 811, 817, 818, 825, 828, 829, 829, 833, 835, 835, 838, 843, 848, 848, 851, 8
58, 860, 860, 864, 867, 870, 870, 872, 874, 875, 876, 876, 878, 885, 887, 889, 892, 893, 905, 907, 908, 909, 912, 919, 921, 924, 925, 926, 927, 929, 931, 931, 935, 936, 938, 939, 943, 943, 948, 952, 954
, 960, 965, 978, 979, 986, 989, 997, 997, 999, 1001, 1002, 1004, 1017, 1018]
```



```
Sorted Array B: [2, 8, 10, 14, 15, 15, 30, 37, 38, 38, 39, 41, 42, 44, 46, 48, 51, 52, 55, 61, 62, 63, 63, 64, 73, 73, 76, 79, 83, 84, 86, 87, 90, 99, 99, 100, 108, 108, 109, 110, 112, 114, 129, 131, 13
3, 133, 136, 136, 139, 140, 140, 141, 144, 146, 148, 161, 161, 163, 164, 165, 166, 167, 167, 171, 175, 176, 176, 180, 180, 183, 184, 188, 190, 190, 192, 194, 196, 203, 207, 211, 211, 214, 220, 220, 228,
 230, 231, 240, 240, 241, 241, 247, 247, 249, 251, 254, 257, 264, 264, 265, 267, 267, 269, 274, 279, 285, 294, 295, 296, 296, 298, 308, 308, 312, 312, 330, 333, 335, 336, 336, 340, 341, 342, 345, 347, 3
48, 351, 352, 353, 354, 356, 358, 362, 365, 365, 366, 367, 371, 373, 377, 379, 379, 387, 388, 389, 393, 395, 397, 398, 405, 405, 408, 411, 414, 415, 416, 416, 420, 423, 424, 429, 435, 437, 437, 438, 439
, 442, 442, 443, 446, 449, 451, 453, 457, 459, 460, 462, 467, 472, 476, 479, 480, 481, 481, 483, 485, 486, 492, 495, 500, 501, 502, 505, 507, 509, 511, 516, 518, 518, 521, 522, 523, 523, 526, 526, 528,
531, 537, 538, 538, 538, 543, 549, 553, 556, 557, 558, 560, 562, 566, 568, 573, 573, 576, 578, 579, 582, 586, 586, 586, 588, 591, 591, 600, 600, 604, 606, 607, 607, 610, 612, 615, 615, 616, 617, 617, 61
9, 621, 624, 629, 630, 635, 636, 638, 640, 641, 649, 650, 654, 656, 660, 663, 665, 665, 666, 672, 678, 684, 684, 685, 688, 689, 693, 696, 698, 699, 700, 701, 701, 709, 714, 715, 720, 720, 721, 724, 727,
 728, 728, 731, 734, 738, 738, 744, 746, 747, 752, 752, 753, 755, 758, 759, 759, 760, 763, 766, 781, 784, 789, 790, 796, 798, 800, 800, 801, 801, 802, 806, 809, 809, 811, 825, 835, 838, 839, 841, 845, 8
46, 847, 848, 848, 849, 850, 851, 852, 853, 857, 857, 864, 865, 867, 870, 871, 878, 879, 883, 896, 897, 897, 903, 903, 904, 906, 910, 912, 915, 917, 917, 919, 919, 927, 929, 935, 936, 939, 941, 942, 944
, 945, 945, 947, 950, 953, 953, 954, 954, 970, 970, 972, 973, 981, 981, 981, 982, 983, 985, 986, 989, 993, 994, 995, 997, 998, 998, 999, 1000, 1006, 1007, 1014, 1014]
```

How this was tested: The sizes of both arrays were initially set to smaller values to check the functionality of the code. Arrays of various sizes, including both odd and even, were tested to confirm that the program handled them properly. Additionally, both sorting algorithms were provided with an already sorted array to verify their correctness.

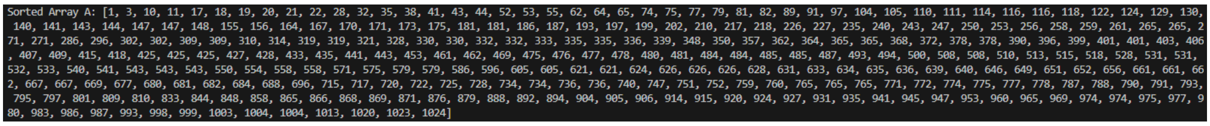
Question: 2

Source Code (to be pasted after code in Question 1):

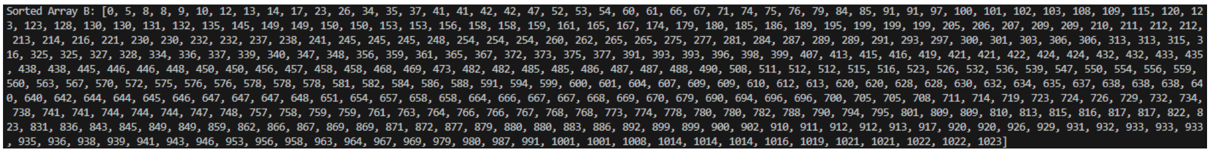
```
arrC = []

ptr1 = 0
ptr2 = 0
```

```
while ptr1 < sizeA and ptr2 < sizeB:
    if arrA[ptr1] > arrB[ptr2]:
        arrC.append(arrB[ptr2])
        ptr2 += 1
    elif arrB[ptr2] > arrA[ptr1]:
        arrC.append(arrA[ptr1])
        ptr1 += 1
    else:
        arrC.append(arrA[ptr1])
        arrC.append(arrB[ptr2])
        ptr1 += 1
        ptr2 += 1

while ptr1 < sizeA:
    arrC.append(arrA[ptr1])
    ptr1 += 1

while ptr2 < sizeB:
    arrC.append(arrB[ptr2])
    ptr2 += 1

print(f"Merged Array: {arrC}")
```

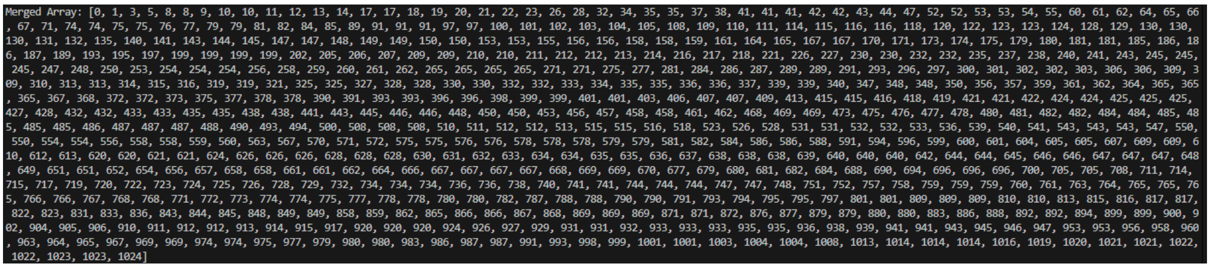
Sample Screen Dumps:



```
Sorted Array A: [1, 3, 10, 11, 17, 18, 19, 20, 21, 22, 28, 32, 35, 38, 41, 43, 44, 52, 53, 55, 62, 64, 65, 74, 75, 77, 79, 81, 82, 89, 91, 97, 104, 105, 110, 111, 114, 116, 116, 118, 122, 124, 129, 130,
 140, 141, 143, 144, 147, 147, 148, 155, 156, 164, 167, 170, 171, 173, 175, 181, 181, 186, 187, 193, 197, 199, 202, 210, 217, 218, 226, 227, 235, 240, 243, 247, 250, 253, 256, 258, 259, 261, 265, 265, 2
71, 271, 286, 296, 302, 302, 309, 309, 310, 314, 319, 319, 321, 328, 330, 330, 332, 332, 333, 335, 335, 336, 339, 348, 350, 357, 362, 364, 365, 365, 368, 372, 378, 378, 390, 396, 399, 401, 401, 403, 406
, 407, 409, 415, 418, 425, 425, 425, 427, 428, 433, 435, 441, 443, 453, 461, 462, 469, 475, 476, 477, 478, 480, 481, 484, 484, 485, 485, 487, 493, 494, 500, 508, 508, 510, 513, 515, 518, 528, 531, 531,
532, 533, 540, 541, 543, 543, 543, 550, 554, 558, 558, 571, 575, 579, 579, 586, 596, 605, 605, 621, 621, 624, 626, 626, 626, 628, 631, 633, 634, 635, 636, 639, 640, 646, 649, 651, 652, 656, 661, 661, 66
2, 667, 667, 669, 677, 680, 681, 682, 684, 688, 696, 715, 717, 720, 722, 725, 728, 734, 734, 736, 736, 740, 747, 751, 752, 759, 760, 765, 765, 765, 771, 772, 774, 775, 777, 778, 787, 788, 790, 791, 793,
 795, 797, 801, 809, 810, 833, 844, 848, 858, 865, 866, 868, 869, 871, 876, 879, 888, 892, 894, 904, 905, 906, 914, 915, 920, 924, 927, 931, 935, 941, 945, 947, 953, 960, 965, 969, 974, 974, 975, 977, 9
80, 983, 986, 987, 993, 998, 999, 1003, 1004, 1004, 1013, 1020, 1023, 1024]
```



```
Sorted Array B: [0, 5, 8, 8, 9, 10, 12, 13, 14, 17, 23, 26, 34, 35, 37, 41, 41, 42, 42, 47, 52, 53, 54, 60, 61, 66, 67, 71, 74, 75, 76, 79, 84, 85, 91, 91, 97, 100, 101, 102, 103, 108, 109, 115, 120, 12
3, 123, 128, 130, 130, 131, 132, 135, 145, 149, 149, 150, 150, 153, 153, 156, 158, 158, 159, 161, 165, 167, 174, 179, 180, 185, 186, 189, 195, 199, 199, 199, 205, 206, 207, 209, 209, 210, 211, 212, 212,
 213, 214, 216, 221, 230, 230, 232, 232, 237, 238, 241, 245, 245, 245, 248, 254, 254, 254, 260, 262, 265, 265, 275, 277, 281, 284, 287, 289, 289, 291, 293, 297, 300, 301, 303, 306, 306, 313, 313, 315, 3
16, 325, 325, 327, 328, 334, 336, 337, 339, 340, 347, 348, 356, 359, 361, 365, 367, 372, 373, 375, 377, 391, 393, 393, 396, 398, 399, 407, 413, 415, 416, 419, 421, 421, 422, 424, 424, 432, 432, 433, 435
, 438, 438, 445, 446, 446, 448, 450, 450, 456, 457, 458, 458, 468, 469, 473, 482, 482, 485, 485, 486, 487, 487, 488, 490, 508, 511, 512, 512, 515, 516, 523, 526, 532, 536, 539, 547, 550, 554, 556, 559,
560, 563, 567, 570, 572, 575, 576, 576, 578, 578, 578, 581, 582, 584, 586, 588, 591, 594, 599, 600, 601, 604, 607, 609, 609, 610, 612, 613, 620, 620, 628, 628, 630, 632, 634, 635, 637, 638, 638, 638, 64
0, 640, 642, 644, 644, 645, 646, 647, 647, 647, 648, 651, 654, 657, 658, 658, 664, 666, 667, 667, 668, 669, 670, 679, 690, 694, 696, 696, 700, 705, 705, 708, 711, 714, 719, 723, 724, 726, 729, 732, 734,
 738, 741, 741, 744, 744, 744, 747, 748, 757, 758, 759, 759, 761, 763, 764, 766, 766, 767, 768, 768, 773, 774, 778, 780, 780, 782, 788, 790, 794, 795, 801, 809, 809, 810, 813, 815, 816, 817, 817, 822, 8
23, 831, 836, 843, 845, 849, 849, 859, 862, 866, 867, 869, 869, 871, 872, 877, 879, 880, 880, 883, 886, 892, 899, 899, 900, 902, 910, 911, 912, 912, 913, 917, 920, 920, 926, 929, 931, 932, 933, 933, 933
, 935, 936, 938, 939, 941, 943, 946, 953, 956, 958, 963, 964, 967, 969, 979, 980, 987, 991, 1001, 1001, 1008, 1014, 1014, 1014, 1016, 1019, 1021, 1021, 1022, 1022, 1023]
```



```
Merged Array: [0, 1, 3, 5, 8, 8, 9, 10, 10, 11, 12, 13, 14, 17, 17, 18, 19, 20, 21, 22, 23, 26, 28, 32, 34, 35, 35, 37, 38, 41, 41, 41, 42, 42, 43, 44, 47, 52, 52, 53, 53, 54, 55, 60, 61, 62, 64, 65, 66
, 67, 71, 74, 74, 75, 75, 76, 77, 79, 79, 81, 82, 84, 85, 89, 91, 91, 91, 97, 97, 100, 101, 102, 103, 104, 105, 108, 109, 110, 111, 114, 115, 116, 116, 118, 120, 122, 123, 123, 124, 128, 129, 130, 130,
130, 131, 132, 135, 140, 141, 143, 144, 145, 147, 147, 148, 149, 149, 150, 150, 153, 153, 155, 156, 156, 158, 158, 159, 161, 164, 165, 167, 167, 170, 171, 173, 174, 175, 179, 180, 181, 181, 185, 186, 18
6, 187, 189, 193, 195, 197, 199, 199, 199, 199, 202, 205, 206, 207, 209, 209, 210, 210, 211, 212, 212, 213, 214, 216, 217, 218, 221, 226, 227, 230, 230, 232, 232, 235, 237, 238, 240, 241, 243, 245, 245,
 245, 247, 248, 250, 253, 254, 254, 254, 256, 258, 259, 260, 261, 262, 265, 265, 265, 265, 271, 271, 275, 277, 281, 284, 286, 287, 289, 289, 291, 293, 296, 297, 300, 301, 302, 302, 303, 306, 306, 309, 3
09, 310, 313, 313, 314, 315, 316, 319, 319, 321, 325, 325, 327, 328, 328, 330, 330, 332, 332, 333, 334, 335, 335, 336, 336, 337, 339, 339, 340, 347, 348, 348, 350, 356, 357, 359, 361, 362, 364, 365, 365
, 365, 367, 368, 372, 372, 373, 375, 377, 378, 378, 390, 391, 393, 393, 396, 396, 398, 399, 399, 401, 401, 403, 406, 407, 407, 409, 413, 415, 415, 416, 418, 419, 421, 421, 422, 424, 424, 425, 425, 425,
427, 428, 432, 432, 433, 433, 435, 435, 438, 438, 441, 443, 445, 446, 446, 448, 450, 450, 453, 456, 457, 458, 458, 461, 462, 468, 469, 469, 473, 475, 476, 477, 478, 480, 481, 482, 482, 484, 484, 485, 48
5, 485, 485, 486, 487, 487, 487, 488, 490, 493, 494, 500, 508, 508, 508, 510, 511, 512, 512, 513, 515, 515, 516, 518, 523, 526, 528, 531, 531, 532, 532, 533, 536, 539, 540, 541, 543, 543, 543, 547, 550,
 550, 554, 554, 556, 558, 558, 559, 560, 563, 567, 570, 571, 572, 575, 575, 576, 576, 578, 578, 578, 579, 579, 581, 582, 584, 586, 586, 588, 591, 594, 596, 599, 600, 601, 604, 605, 605, 607, 609, 609, 6
10, 612, 613, 620, 620, 621, 621, 624, 626, 626, 626, 628, 628, 628, 630, 631, 632, 633, 634, 634, 635, 635, 636, 637, 638, 638, 638, 639, 640, 640, 640, 642, 644, 644, 645, 646, 646, 647, 647, 647, 648
, 649, 651, 651, 652, 654, 656, 657, 658, 658, 661, 661, 662, 664, 666, 667, 667, 667, 667, 668, 669, 669, 670, 677, 679, 680, 681, 682, 684, 688, 690, 694, 696, 696, 696, 700, 705, 705, 708, 711, 714,
715, 717, 719, 720, 722, 723, 724, 725, 726, 728, 729, 732, 734, 734, 734, 736, 736, 738, 740, 741, 741, 744, 744, 744, 747, 747, 748, 751, 752, 757, 758, 759, 759, 759, 760, 761, 763, 764, 765, 765, 76
5, 766, 766, 767, 768, 768, 771, 772, 773, 774, 774, 775, 777, 778, 778, 780, 780, 782, 787, 788, 788, 790, 790, 791, 793, 794, 795, 795, 797, 801, 801, 809, 809, 809, 810, 810, 813, 815, 816, 817, 817,
 822, 823, 831, 833, 836, 843, 844, 845, 848, 849, 849, 858, 859, 862, 865, 866, 866, 867, 868, 869, 869, 869, 871, 871, 872, 876, 877, 879, 879, 880, 880, 883, 886, 888, 892, 892, 894, 899, 899, 900, 9
02, 904, 905, 906, 910, 911, 912, 912, 913, 914, 915, 917, 920, 920, 920, 924, 926, 927, 929, 931, 931, 932, 933, 933, 933, 935, 935, 936, 938, 939, 941, 941, 943, 945, 946, 947, 953, 953, 956, 958, 960
, 963, 964, 965, 967, 969, 969, 974, 974, 975, 977, 979, 980, 980, 983, 986, 987, 987, 991, 993, 998, 999, 1001, 1001, 1003, 1004, 1004, 1008, 1013, 1014, 1014, 1014, 1016, 1019, 1020, 1021, 1021, 1022,
 1022, 1023, 1023, 1024]
```

How this was tested: The sizes of both arrays were initially set to smaller values to check the functionality of the code. The lengths of arrays arrA, arrB and arrC were printed and as expected, the length of arrC was the combined lengths of arrays arrA and arrB.

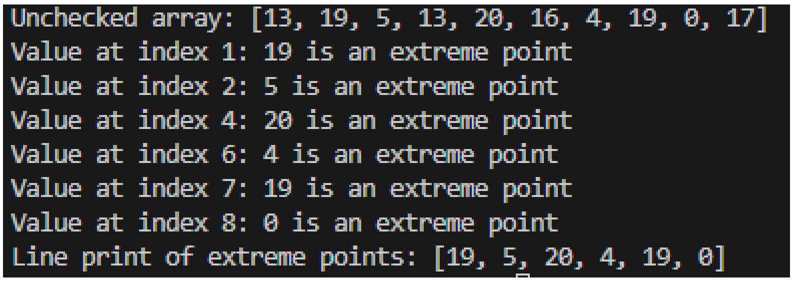
Question: 3

Source Code:

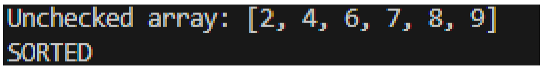
```
import random
```

```
  arr = []
  eArr = [] # holds the extreme points
  for i in range(10):
      arr.append(random.randint(0,20))
  print(f"Unchecked array: {arr}")

  def extremeCheck(arr):
      global epts
      epts = False
      for i in range(1, len(arr) - 1):
          if (arr[i] < arr[i-1] and arr[i] < arr[i+1]) or (arr[i] > arr[i-1] and
arr[i] > arr[i+1]):
              eArr.append(arr[i])
              print(f"Value at index {i}: {arr[i]} is an extreme point")
              epts = True



  extremeCheck(arr)
  if epts:
      print(f"Line print of extreme points: {eArr}")
  else:
      print("SORTED")
```

Sample Screen Dumps:



```
Unchecked array: [13, 19, 5, 13, 20, 16, 4, 19, 0, 17]
Value at index 1: 19 is an extreme point
Value at index 2: 5 is an extreme point
Value at index 4: 20 is an extreme point
Value at index 6: 4 is an extreme point
Value at index 7: 19 is an extreme point
Value at index 8: 0 is an extreme point
Line print of extreme points: [19, 5, 20, 4, 19, 0]
```



```
Unchecked array: [2, 4, 6, 7, 8, 9]
SORTED
```

How this was tested: To test the extremeCheck function with an unsorted array, the code was simply run. As it’s highly unlikely for the random generated array to be sorted, a temporary sorted array was made and was passed to extremeCheck function.

Do you agree that an array has no extreme points if and only if it is sorted? Explain your answer

It is true that array has no extreme points if and only if it is sorted. If the array is sorted in ascending order, all elements (excluding first and last) will be larger than the previous element but smaller than the next element. Similarly, if the array is sorted in descending order, all elements (excluding first and last) will be smaller than the previous element but larger than the next element. Such conditions A[i-1] < A[i] < A[i+1] or A[i-1] > A[i] >

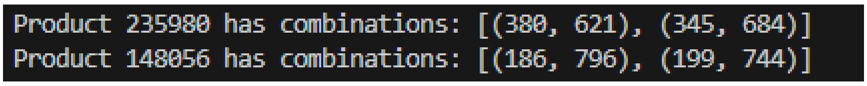
A[i+1] does not comply with the stated conditions A[i– 1] < A[i] > A[i + 1] or A[i − 1] > A[i] < A[i + 1]

Question: 4

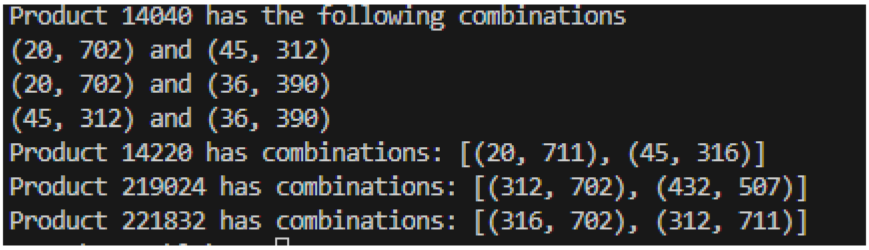
Source Code:

```
   import random
   arr = []
   N = 50
   for i in range(N):
       arr.append(random.randint(1,1024))

   combinations = {}

   # Storing ALL UNIQUE COMBINATIONS as dictionary with array values
   for i in range(len(arr)):
       for j in range(i+1, len(arr)):
           a = arr[i]
           b = arr[j]
           product = a*b

           if product not in combinations.keys():
               combinations[product] = []
           if a>b: # Storing as a sorted tuple to avoid duplicates, easier to work
with also.
               a,b = b,a
           if (a,b) not in combinations[product]:
               combinations[product].append((a,b))

   found = False

   for key in combinations.keys():
       ptr = combinations.get(key)
       if len(ptr) == 2:
           found = True
           print(f"Product {key} has combinations: {ptr}")
       elif len(ptr) > 2:
           found = True
           print(f"Product {key} has the following combinations")
           for i in range(len(ptr)):
               for j in range(i+1, len(ptr)):
                   a = ptr[i]
                   b = ptr[j]
                   print(f"{a} and {b}")

   if not found:
       print("There are no 2-pairs of integers having the same product for this
run. Try increasing the value of N or re-run the code!")

   """ rows = [f"Key {k} has value {v}" for k, v in combinations.items()]
   max_len = max(len(row) for row in rows)

   count = 0
   print("FULL DICTIONARY:")
   for k,v in combinations.items():
       text = f"Key {k} has value {v}"
       print(f"{text:<{max_len}}", end="\t")
       count += 1
       if (count == 5):
           count = 0
           print() """
```

Sample Screen Dumps:



```
Product 235980 has combinations: [(380, 621), (345, 684)]
Product 148056 has combinations: [(186, 796), (199, 744)]
```



```
Product 14040 has the following combinations
(20, 702) and (45, 312)
(20, 702) and (36, 390)
(45, 312) and (36, 390)
Product 14220 has combinations: [(20, 711), (45, 316)]
Product 219024 has combinations: [(312, 702), (432, 507)]
Product 221832 has combinations: [(316, 702), (312, 711)]
```

How this was tested: Different values of N were fed into the random generator so as to increase the likelihood of having more 2-pairs of integers.

As an extra, I felt it would be good practice to have the code print the full dictionary at the end but this is commented out by default to not spam the output.

In order to present the dictionary in evenly spaced columns, I consulted ChatGPT with the following prompt:

How can I not use a fixed width to determine the maximum length needed for this code: (insert code). I don't want to have very big spaces between columns

This resulted in calculating the max length of the longest row so as to then feed this to the spacing system in the print statement.

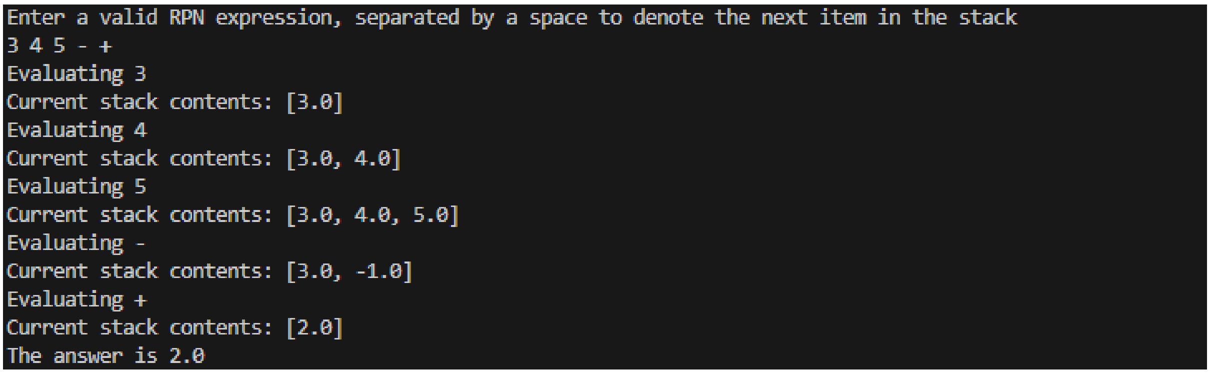
Question: 5

Source Code:

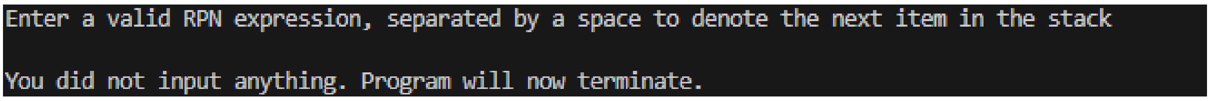
```
class Stack:
    def __init__(self):
        self.items = []

    def __str__(self):
            return f"Current stack contents: {self.items}"

        def push(self,item):
            self.items.append(item)

        def pop(self):
            if self.is_empty():
                print("Cannot pop an empty stack. Program will now terminate.")
                exit(1)
            else:
                return self.items.pop()

        def peek(self):
            if self.is_empty():
                print("There are no elements in the stack")
            else:
                return self.items[-1]

        def is_empty(self):
            if self.size() == 0:
                return True

        def size(self):
            return len(self.items)


    def eval(expression):
        contents = expression.split()

        if not contents:
            print("You did not input anything. Program will now terminate. ")
            exit(1)

        stack = Stack()
        operators = ['+', '-', 'x', '/']

        for i in contents:
            print(f"Evaluating {i}")
            try:
                i = float(i)
                stack.push(i)
            except ValueError:
                if i not in operators:
                    print(f"Operator {i} is invalid. Program will now terminate")
                    exit(1)
                elif i in operators:
                    if stack.size() < 2:
                        print(f"Not enough numbers in the stack for operator {i}.
Program will now terminate.")
                        exit(1)
                    else:
                        b = stack.pop()
                        a = stack.pop()
                        if i == '+':
                            stack.push(a + b)
                        elif i == '-':
                            stack.push(a - b)
                        elif i == 'x':
                            stack.push(a * b)
```

```
                       elif i == '/':
                           if b == 0:
                               print("Cannot divide by zero. Program will now
terminate.")
                               exit(1)
                           stack.push(a / b)
           print(stack)

       if stack.size() != 1:
               print("There are still additional contents in the stack and not
enough operators. Invalid RPN entry. Program will now terminate")
               exit(1)
       else:
           print(f"The answer is {stack.peek()}")


   entry = input("Enter a valid RPN expression, separated by a space to denote the
next item in the stack \n")
   eval(entry)
```

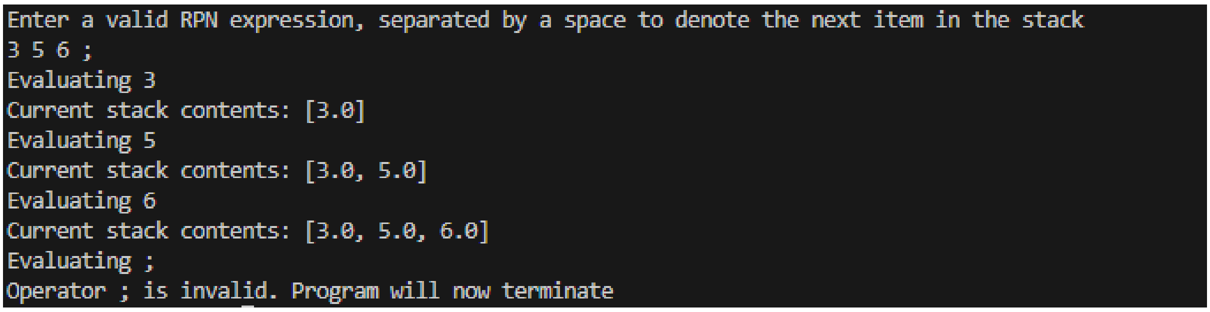
Sample Screen Dumps:



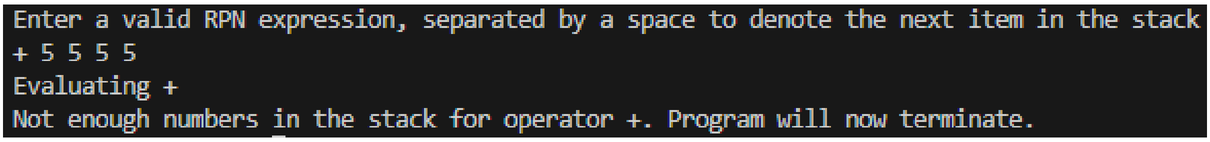
```
Enter a valid RPN expression, separated by a space to denote the next item in the stack
3 4 5 - +
Evaluating 3
Current stack contents: [3.0]
Evaluating 4
Current stack contents: [3.0, 4.0]
Evaluating 5
Current stack contents: [3.0, 4.0, 5.0]
Evaluating -
Current stack contents: [3.0, -1.0]
Evaluating +
Current stack contents: [2.0]
The answer is 2.0
```



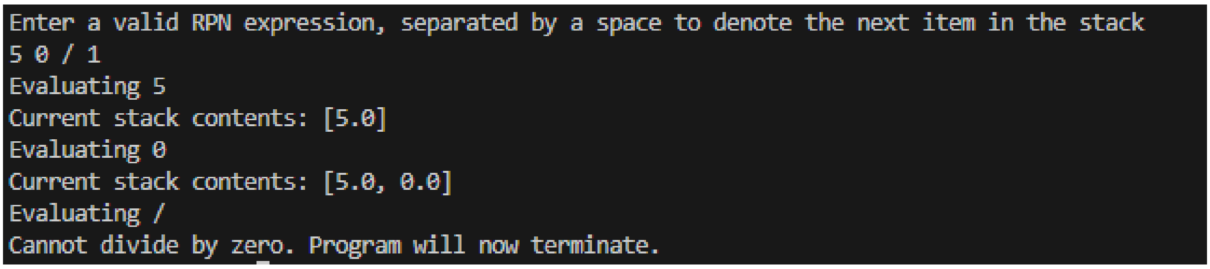
```
Enter a valid RPN expression, separated by a space to denote the next item in the stack

You did not input anything. Program will now terminate.
```



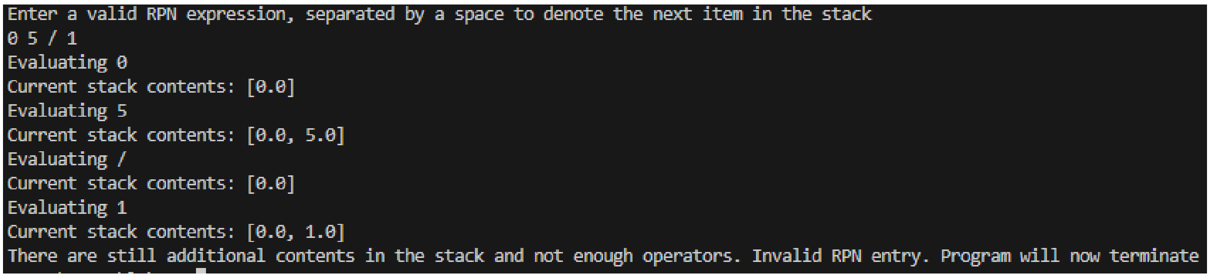
```
Enter a valid RPN expression, separated by a space to denote the next item in the stack
3 5 6 ;
Evaluating 3
Current stack contents: [3.0]
Evaluating 5
Current stack contents: [3.0, 5.0]
Evaluating 6
Current stack contents: [3.0, 5.0, 6.0]
Evaluating ;
Operator ; is invalid. Program will now terminate
```



```
Enter a valid RPN expression, separated by a space to denote the next item in the stack
+ 5 5 5 5
Evaluating +
Not enough numbers in the stack for operator +. Program will now terminate.
```



```
Enter a valid RPN expression, separated by a space to denote the next item in the stack
5 0 / 1
Evaluating 5
Current stack contents: [5.0]
Evaluating 0
Current stack contents: [5.0, 0.0]
Evaluating /
Cannot divide by zero. Program will now terminate.
```



```
Enter a valid RPN expression, separated by a space to denote the next item in the stack
0 5 / 1
Evaluating 0
Current stack contents: [0.0]
Evaluating 5
Current stack contents: [0.0, 5.0]
Evaluating /
Current stack contents: [0.0]
Evaluating 1
Current stack contents: [0.0, 1.0]
There are still additional contents in the stack and not enough operators. Invalid RPN entry. Program will now terminate
```

How this was tested: Several test cases were fed as input expressions to test all the error messages; valid expressions, empty input, invalid operators, insufficient numbers for an operator, dividing by zero and stack not being of size 1 at end of evaluation. The Stack class includes two unused error messages which were kept to preserve its core functionality and structure.

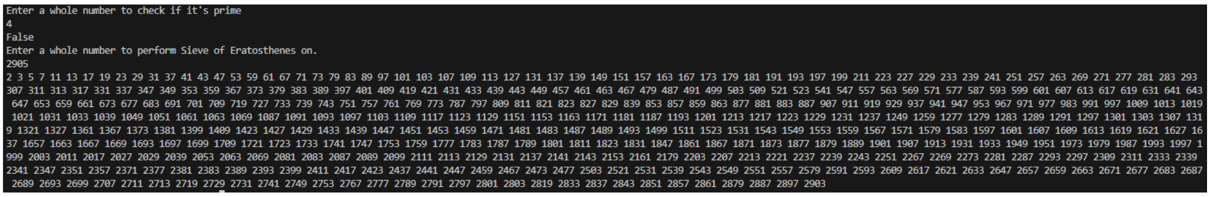
Question: 6

Source Code:

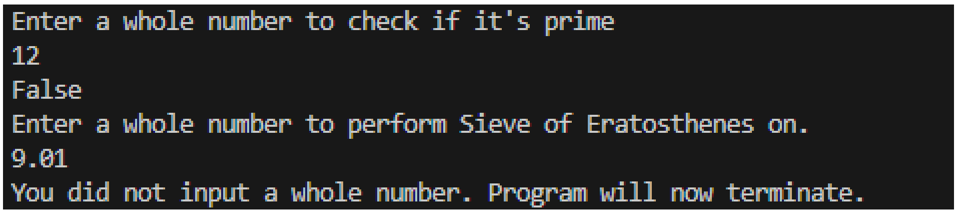
```
def isPrime(num):
    if num <= 1:
        return False
    if num == 2:
        return True
    for i in range(2, num):
        if num % i == 0:
            return False
        else:
            return True

def sieve(num):
    prime = [True for i in range(num+1)]
    prime[0] = prime[1] = False

    for i in range(2, num + 1):
        if prime[i]:
            for j in range(i * 2, num + 1, i):
                prime[j] = False

    for i in range(2, num+1):
        if prime[i]:
            print(i, end= " ")
```

```
try:
    num = int(input("Enter a whole number to check if it's prime \n"))
    print(isPrime(num))
except ValueError:
    print("You did not input a whole number. Program will now terminate.")
    exit(1)

try:
    num = int(input("Enter a whole number to perform Sieve of Eratosthenes on. 
\n"))
    sieve(num)
except ValueError:
    print("You did not input a whole number. Program will now terminate.")
    exit(1)
```

Sample Screen Dumps:



```
Enter a whole number to check if it's prime
4
False
Enter a whole number to perform Sieve of Eratosthenes on.
2905
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293
307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593 599 601 607 613 617 619 631 641 643
 647 653 659 661 673 677 683 691 701 709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863 877 881 883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997 1009 1013 1019
 1021 1031 1033 1039 1049 1051 1061 1063 1069 1087 1091 1093 1097 1103 1109 1117 1123 1129 1151 1153 1163 1171 1181 1187 1193 1201 1213 1217 1223 1229 1231 1237 1249 1259 1277 1279 1283 1289 1291 1297 1301 1303 1307 131
9 1321 1327 1361 1367 1373 1381 1399 1409 1423 1427 1429 1433 1439 1447 1451 1453 1459 1471 1481 1483 1487 1489 1493 1499 1511 1523 1531 1543 1549 1553 1559 1567 1571 1579 1583 1597 1601 1607 1609 1613 1619 1621 1627 16
37 1657 1663 1667 1669 1693 1697 1699 1709 1721 1723 1733 1741 1747 1753 1759 1777 1783 1787 1789 1801 1811 1823 1831 1847 1861 1867 1871 1873 1877 1879 1889 1901 1907 1913 1931 1933 1949 1951 1973 1979 1987 1993 1997 1
999 2003 2011 2017 2027 2029 2039 2053 2063 2069 2081 2083 2087 2089 2099 2111 2113 2129 2131 2137 2141 2143 2153 2161 2179 2203 2207 2213 2221 2237 2239 2243 2251 2267 2269 2273 2281 2287 2293 2297 2309 2311 2333 2339
2341 2347 2351 2357 2371 2377 2381 2383 2389 2393 2399 2411 2417 2423 2437 2441 2447 2459 2467 2473 2477 2503 2521 2531 2539 2543 2549 2551 2557 2579 2591 2593 2609 2617 2621 2633 2647 2657 2659 2663 2671 2677 2683 2687
 2689 2693 2699 2707 2711 2713 2719 2729 2731 2741 2749 2753 2767 2777 2789 2791 2797 2801 2803 2819 2833 2837 2843 2851 2857 2861 2879 2887 2897 2903
```



```
Enter a whole number to check if it's prime
12
False
Enter a whole number to perform Sieve of Eratosthenes on.
9.01
You did not input a whole number. Program will now terminate.
```

How this was tested: Different numbers were fed as input to both functions to test their functionality including invalid numbers to test the functionality of the try except blocks.

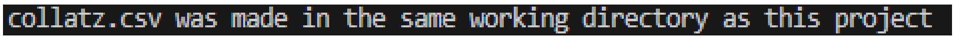
Question: 7

Source Code:

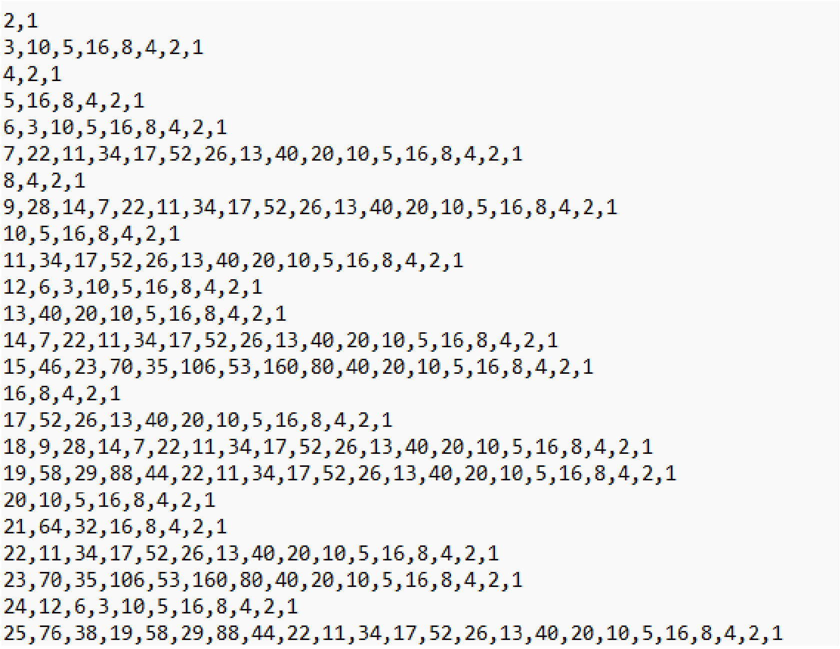
```
import csv
def collatzSeq(num):
    seq = [num]
    while num != 1:
        if num % 2 == 0:
            num = num // 2
        else:
            num = (3 * num) + 1
        seq.append(num)
    return seq


with open("collatz.csv", "w", newline='') as file:
    w = csv.writer(file)
    for i in range(2, 513):
        collatz = collatzSeq(i)
        # w.writerow(collatz) # Pure CSVs
```

```
            w.writerow([f"{i}: {','.join(map(str, collatz))}"]) # Prettier version
        print("collatz.csv was made in the same working directory as this project")
```

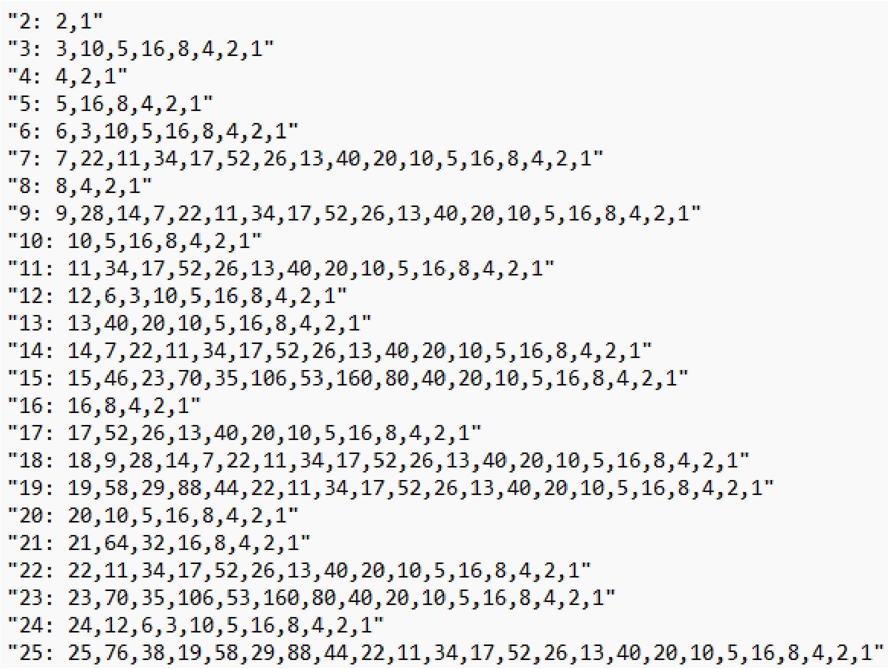
Sample Screen Dumps:



```
collatz.csv was made in the same working directory as this project
```



```
2,1
3,10,5,16,8,4,2,1
4,2,1
5,16,8,4,2,1
6,3,10,5,16,8,4,2,1
7,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1
8,4,2,1
9,28,14,7,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1
10,5,16,8,4,2,1
11,34,17,52,26,13,40,20,10,5,16,8,4,2,1
12,6,3,10,5,16,8,4,2,1
13,40,20,10,5,16,8,4,2,1
14,7,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1
15,46,23,70,35,106,53,160,80,40,20,10,5,16,8,4,2,1
16,8,4,2,1
17,52,26,13,40,20,10,5,16,8,4,2,1
18,9,28,14,7,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1
19,58,29,88,44,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1
20,10,5,16,8,4,2,1
21,64,32,16,8,4,2,1
22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1
23,70,35,106,53,160,80,40,20,10,5,16,8,4,2,1
24,12,6,3,10,5,16,8,4,2,1
25,76,38,19,58,29,88,44,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1
```



```
"2: 2,1"
"3: 3,10,5,16,8,4,2,1"
"4: 4,2,1"
"5: 5,16,8,4,2,1"
"6: 6,3,10,5,16,8,4,2,1"
"7: 7,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1"
"8: 8,4,2,1"
"9: 9,28,14,7,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1"
"10: 10,5,16,8,4,2,1"
"11: 11,34,17,52,26,13,40,20,10,5,16,8,4,2,1"
"12: 12,6,3,10,5,16,8,4,2,1"
"13: 13,40,20,10,5,16,8,4,2,1"
"14: 14,7,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1"
"15: 15,46,23,70,35,106,53,160,80,40,20,10,5,16,8,4,2,1"
"16: 16,8,4,2,1"
"17: 17,52,26,13,40,20,10,5,16,8,4,2,1"
"18: 18,9,28,14,7,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1"
"19: 19,58,29,88,44,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1"
"20: 20,10,5,16,8,4,2,1"
"21: 21,64,32,16,8,4,2,1"
"22: 22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1"
"23: 23,70,35,106,53,160,80,40,20,10,5,16,8,4,2,1"
"24: 24,12,6,3,10,5,16,8,4,2,1"
"25: 25,76,38,19,58,29,88,44,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1"
```

(only from 2 to 25 shown in screenshot to not spam images)

How this was tested: The code was run to check that the code prints the Collatz sequence from 2 up to 512 (included). Furthermore, both the pure and pretty format were tested by uncommenting one and commenting the other.

Luka Cassar

In order to present the data in the pretty format, I consulted Grok with the following prompt:

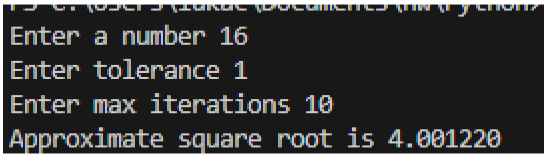
Make the output look like in this format '512: 512,256,128... etc'

Question: 8

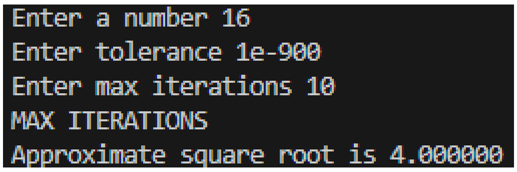
Source Code:

```
def nrmethod(num, tolerance, maxIterations):
    if num < 0:
        print("Cannot compute square root of a negative number")
    if num == 0:
        return 0

    x_0 = num / 2

    for i in range(maxIterations):
            x_1 = (x_0 + num / x_0) / 2
            if abs(x_0 * x_0 - num) < tolerance:
                return x_1
            x_0 = x_1
    print("MAX ITERATIONS REACHED")
    return x_0

num = float(input("Enter a number "))
tolerance = float(input("Enter tolerance "))
maxIterations = int(input("Enter max iterations "))

result = nrmethod(num, tolerance, maxIterations)
print(f"Approximate square root is {result:.6f}")
```

Sample Screen Dumps:



```
PS C:\Users\lukac\Documents\HW\Python>
Enter a number 16
Enter tolerance 1
Enter max iterations 10
Approximate square root is 4.001220
```



```
Enter a number 16
Enter tolerance 1e-900
Enter max iterations 10
MAX ITERATIONS
Approximate square root is 4.000000
```

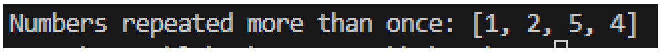
How this was tested: Valid inputs were given and the result was found to be a close approximation. It was noted that the result was very inaccurate with low max iterations.

Question: 9

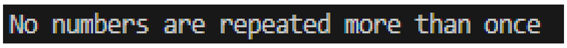
Source Code:

```
def findRepeated(arr):
    freq = {}
    nums = []
    for i in arr:
        if (freq.get(i) is not None):
            freq[i] = freq[i] + 1
        else:
            freq[i] = 1

    for i in freq.keys():
        if freq.get(i) > 1:
            nums.append(i)
    return nums

arr = [1,2,1,5,6,5,8,9,56,4,2,5,7,3,2,1,4]
result = findRepeated(arr)
if len(result) != 0:
    print(f"Numbers repeated more than once: {result}")
else:
    print("No numbers are repeated more than once")
```

Sample Screen Dumps:



```
Numbers repeated more than once: [1, 2, 5, 4]
```



```
No numbers are repeated more than once
```

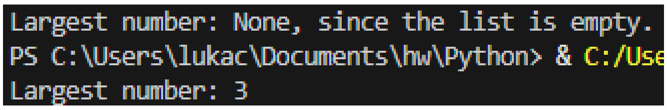
How this was tested: Different combinations of values in arr were fed to the function and it outputted as expected.

Question: 10

Source Code:

```
def findLargest(arr):
    if len(arr) == 0:
        return("None, since the list is empty.")
    if len(arr) == 1:
        return arr[0]
    tail = findLargest(arr[1:])
    if arr[0] > tail:
        return arr[0]
    else:
        return tail


arr = []
print(f"Largest number: {findLargest(arr)}")
```

Sample Screen Dumps:



```
Largest number: None, since the list is empty.
PS C:\Users\lukac\Documents\hw\Python> & C:/Use
Largest number: 3
```

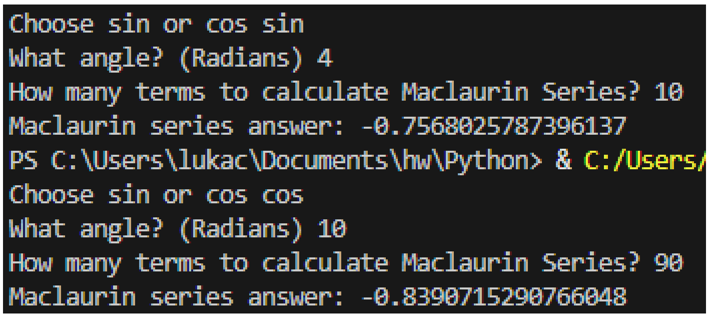
How this was tested: arr was initialised with several different numbers and the code was run to test its functionality. Furthermore, arr was also left as an empty array to fully test out the code.

Question: 11

Source Code:

```
def maclaurin(ang, n, trig):
    if n < 1:
        return "Number of terms must be at least 1"
    if trig not in ['sin', 'cos']:
        return "Trig function must be 'sin' or 'cos'"

    finalAns = 0
    if trig == 'sin': # the first terms
        ans = ang
    else:
        ans = 1

    for r in range(0,n):
        finalAns += ans
        if trig == 'sin':
            ans *= -(ang * ang) / ((2 * r + 2) * (2 * r + 3)) # Writing the
series as a simple product to avoid large factorials. Divide the (k+1)th term by
the kth term to get this expression.
        else:
            ans *= -(ang * ang) / ((2 * r + 1) * (2 * r + 2)) # Same, but
slightly different for cos.
    return finalAns


trig = input("Choose sin or cos ")
ang = float(input("What angle? (Radians) "))
n = int(input("How many terms to calculate Maclaurin Series? "))

print(f"Maclaurin series answer: {maclaurin(ang,n,trig)}")
```

Sample Screen Dumps:



```
Choose sin or cos sin
What angle? (Radians) 4
How many terms to calculate Maclaurin Series? 10
Maclaurin series answer: -0.7568025787396137
PS C:\Users\lukac\Documents\hw\Python> & C:/Users/
Choose sin or cos cos
What angle? (Radians) 10
How many terms to calculate Maclaurin Series? 90
Maclaurin series answer: -0.8390715290766048
```

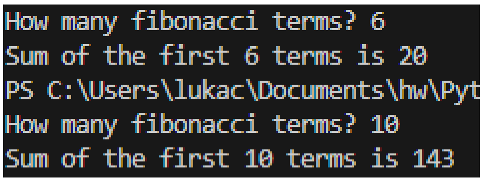
How this was tested: Different values of terms (n) were given to see how the results varies term by term. In general, a lot of output combinations were tried out to fully test the functionality.

Question: 12

Source Code:

```
def sumFibonacci(n):

    if n == 1:
        return 1
    if n == 2:
        return 2

    a = 1
    b = 1
    total = 2  # Sum of first two terms

    for i in range(3, n + 1):
        c = a + b
        total += c
        a = b
        b = c

    return total

n = int(input("How many fibonacci terms? "))
print(f"Sum of the first {n} terms is {sumFibonacci(n)}")
```

Sample Screen Dumps:

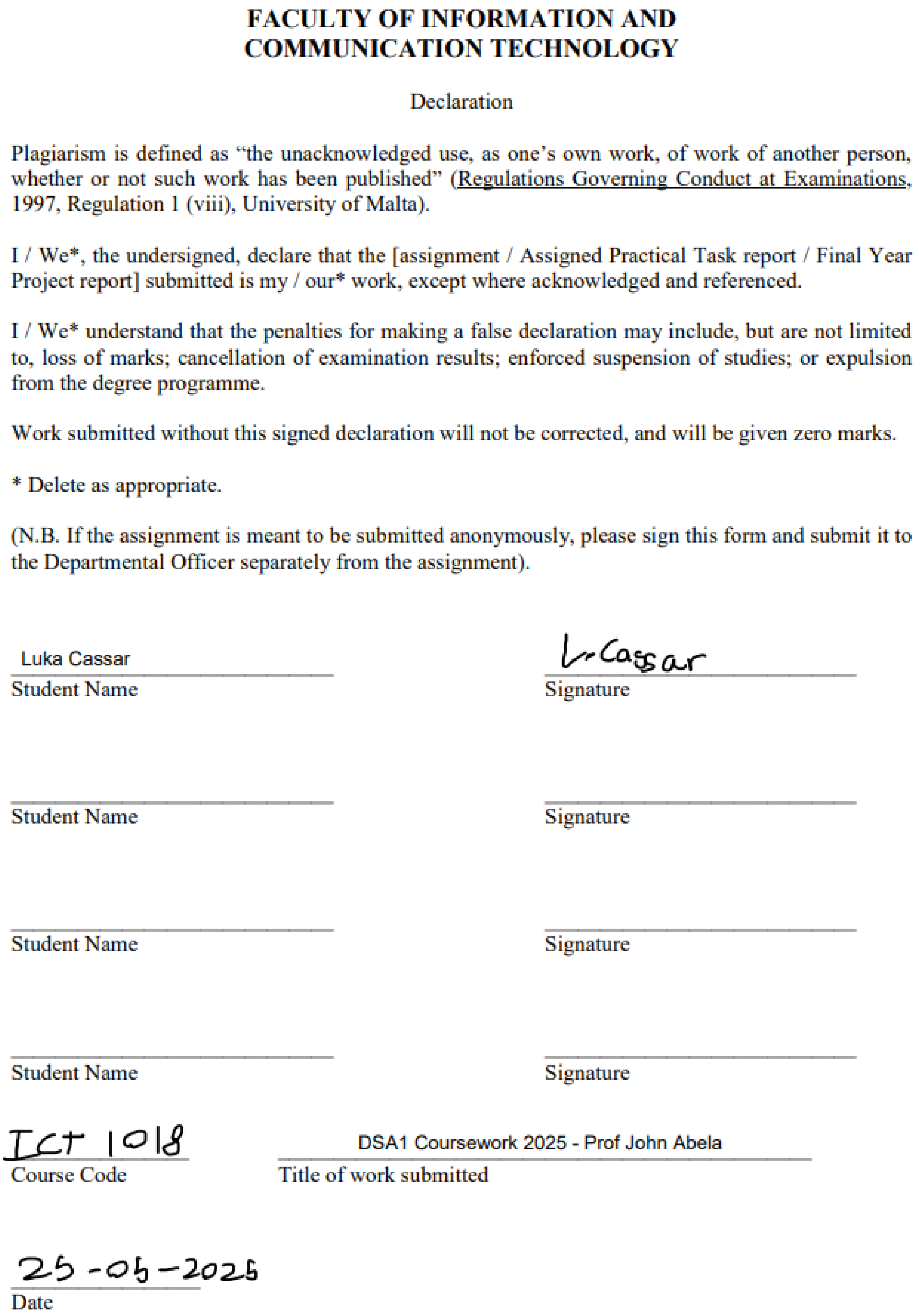


```
How many fibonacci terms? 6
Sum of the first 6 terms is 20
PS C:\Users\lukac\Documents\hw\Pyt
How many fibonacci terms? 10
Sum of the first 10 terms is 143
```

How this was tested: The code was simply run and different values of n were fed. The sum was calculated manually as well to see if it matches with what the code outputs.

Luka Cassar

Plagiarism Declaration



FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as "the unacknowledged use, as one's own work, of work of another person, whether or not such work has been published" (Regulations Governing Conduct at Examinations, 1997, Regulation 1 (viii), University of Malta).

I / We*, the undersigned, declare that the [assignment / Assigned Practical Task report / Final Year Project report] submitted is my / our* work, except where acknowledged and referenced.

I / We* understand that the penalties for making a false declaration may include, but are not limited to, loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected, and will be given zero marks.

* Delete as appropriate.

(N.B. If the assignment is meant to be submitted anonymously, please sign this form and submit it to the Departmental Officer separately from the assignment).

Luka Cassar
Student Name

L. Cassar
Signature

Student Name

Signature

Student Name

Signature

Student Name

Signature

ICT 1018
Course Code

DSA1 Coursework 2025 - Prof John Abela
Title of work submitted

25-05-2025
Date