

Data Structures and Algorithms : Coursework

Student Dave Galea
Lecturer Prof. John Abela
Date May 2025
Unit Code ICT 1018

1. Statement Of Completion

Question 1 – Attempted and works well.

Question 2 – Attempted and works well.

Question 3 – Attempted and works well.

Question 4 – Attempted and works well.

Question 5 – Attempted and works well.

Question 6 – Attempted and works well.

Question 7 – Attempted and works well.

Question 8 – Attempted and works well.

Question 9 – Attempted and works well.

Question 10 – Attempted and works well.

Question 11 – Attempted and works well.

Question 12 – Attempted and works well.

2. Source code, Output, and Testing

Question 1 & 2: Shell Sort, Quick Sort, and Merging of two arrays

Source Code

```
##### QUESTION 1
import random
# Creating Arrays A and B
A = [random.randint(0, 1024) for _ in range(500)]
B = [random.randint(0, 1024) for _ in range(300)]

# Sorting array A with Shell Sort
def shell_sort(arr):
    n = len(arr)
    gap = n // 2

    while gap > 0: # While gap is larger than zero
        for i in range(gap, n): # iterate through elements 'gap' positions apart
            until the end
            temp = arr[i]
            j = i

            while j >= gap and arr[j - gap] > temp:
                # While element 'gap' positions away is greater than current value
                shift forward
                arr[j] = arr[j - gap]
                j -= gap # Check for earlier elements at gap positions

            arr[j] = temp
        gap = gap // 2
```

```

    return arr

# Sorting array B with Quick Sort
# Partitioning
def partition(arr, left, right):
    pivot = arr[right] # Setting pivot as rightmost element of list (randomized, so
                        # this is fair)
    pivot_pos = left - 1
    for j in range(left, right): # runs from left to right
        if arr[j] <= pivot:
            pivot_pos = pivot_pos + 1
            (arr[pivot_pos], arr[j]) = (arr[j], arr[pivot_pos])
    # Placing pivot in correct position at the end of the loop
    (arr[pivot_pos + 1], arr[right]) = (arr[right], arr[pivot_pos + 1])
    return pivot_pos + 1

# Calling Quicksort recursively
def quick_sort(arr, left, right):
    if left < right:
        pivot_index = partition(arr, left, right) # finding pivot index from
        partition
        quick_sort(arr, left, pivot_index - 1) # Sorting elements less than pivot
        quick_sort(arr, pivot_index + 1, right) # Sorting elements greater than pivot
    return arr

shell_sort(A)
quick_sort(B, 0, len(B) - 1)

# function used to print the array into readable chunks
def print_array_in_chunks(arr, chunk_size=30):
    for i in range(0, len(arr), chunk_size):
        print(arr[i:i + chunk_size])

print("QUESTION 1: \nSorted Array A of size:", len(A))
print_array_in_chunks(A)
print("\nSorted Array B of size:", len(B))
print_array_in_chunks(B)

# TESTING
# Check if A is sorted
# All function in python which gives true if all elements of given iterable are true
print("\nA is sorted", all(A[i] <= A[i + 1] for i in range(len(A) - 1)))
# Check if B is sorted
print("B is sorted:", all(B[i] <= B[i + 1] for i in range(len(B) - 1)))

##### QUESTION 2

def merge_arrays(arr1, arr2):
    i = 0 # For A
    j = 0 # For B
    k = 0 # For C
    n = len(arr1)
    m = len(arr2)
    joined_arr = [k] * (n + m) # Create array joined size of A + B

    while i < n and j < m: # While both indexes are in bounds of their respective
        arrays:
            if arr1[i] <= arr2[j]: # Compare elements from both arrays and add smaller
                one to C
                joined_arr[k] = arr1[i] # smaller goes to A
                i = i + 1
            else:
                joined_arr[k] = arr2[j] # smaller goes to B
                j = j + 1
            k = k + 1 # increment index k of C by one after each comparison

```

```

while i < n: # If any elements left over in A (in case A is larger than B) move
    element to C
    joined_arr[k] = arr1[i]
    i = i + 1
    k = k + 1

while j < m: # Same thing for B
    joined_arr[k] = arr2[j]
    j = j + 1
    k = k + 1
return joined_arr

C = merge_arrays(shell_sort(A), quick_sort(B, 0, len(B) - 1))
print("\n\nQUESTION 2: \nSorted Array C of size:", len(C))
print_array_in_chunks(C)
print("\n C is sorted:", all(C[i] <= C[i + 1] for i in range(len(C) - 1)))

```

Screenshot of the Output

```

QUESTION 1:
Sorted Array A of size: 500
[3, 3, 4, 10, 10, 11, 13, 14, 15, 15, 21, 26, 26, 31, 33, 33, 33, 34, 35, 35, 40, 42, 45, 46, 47, 48, 48, 53, 53, 54]
[55, 59, 60, 62, 68, 69, 70, 79, 88, 91, 100, 108, 110, 114, 115, 116, 116, 117, 122, 125, 126, 128, 130, 131, 134, 135, 137, 138, 139, 139]
[143, 143, 145, 146, 147, 148, 148, 153, 155, 156, 157, 158, 159, 160, 163, 165, 167, 168, 168, 169, 170, 170, 175, 175, 176, 177, 178, 180, 180, 181]
[182, 184, 189, 193, 194, 194, 195, 197, 199, 203, 205, 205, 206, 207, 208, 209, 210, 211, 212, 217, 219, 221, 221, 222, 226, 234, 237, 240, 240, 245]
[252, 254, 255, 255, 256, 258, 261, 262, 263, 263, 272, 273, 274, 274, 274, 277, 280, 281, 281, 281, 282, 283, 283, 283, 284, 285, 287, 290, 292, 292]
[295, 297, 300, 302, 303, 304, 309, 313, 314, 320, 325, 325, 325, 326, 326, 327, 329, 329, 330, 331, 332, 333, 336, 337, 339, 340, 341, 349, 351, 354]
[357, 357, 358, 359, 360, 362, 368, 368, 370, 373, 377, 380, 380, 384, 384, 385, 391, 391, 393, 393, 395, 405, 405, 406, 409, 412, 413, 413, 413, 418]
[421, 423, 428, 429, 432, 440, 441, 442, 444, 446, 452, 452, 457, 460, 464, 464, 466, 468, 474, 482, 482, 482, 483, 488, 489, 491, 495, 499, 500, 504]
[509, 511, 513, 514, 515, 517, 519, 521, 521, 525, 525, 527, 527, 529, 531, 534, 535, 535, 537, 540, 541, 544, 545, 548, 552, 553, 554, 555, 556, 559]
[561, 563, 570, 570, 572, 575, 578, 582, 584, 584, 587, 589, 590, 590, 593, 593, 594, 601, 605, 605, 607, 607, 610, 611, 611, 617, 620, 621, 623, 626]
[635, 637, 638, 640, 641, 642, 642, 643, 647, 651, 655, 657, 657, 659, 659, 662, 665, 666, 676, 678, 679, 686, 689, 693, 693, 698, 700, 700, 703, 704]
[704, 708, 710, 715, 715, 716, 718, 718, 720, 721, 721, 721, 721, 731, 731, 734, 740, 741, 742, 744, 746, 750, 750, 752, 755, 756, 757, 759, 762, 763]
[764, 765, 767, 768, 769, 772, 776, 778, 779, 779, 783, 784, 791, 792, 793, 794, 795, 795, 798, 802, 804, 806, 807, 808, 809, 809, 814, 814, 815, 818]
[820, 821, 825, 825, 827, 827, 828, 831, 831, 831, 842, 842, 843, 845, 845, 852, 854, 854, 855, 858, 860, 861, 863, 865, 866, 867, 872, 873, 874, 877]
[881, 881, 888, 889, 889, 890, 891, 891, 892, 892, 900, 901, 903, 907, 907, 910, 912, 913, 915, 919, 920, 921, 926, 927, 930, 930, 932, 938, 940, 942]
[943, 945, 945, 949, 951, 955, 957, 959, 959, 960, 960, 963, 965, 967, 967, 967, 968, 968, 968, 971, 973, 974, 976, 976, 976, 977, 977, 980, 982, 986]
[988, 989, 991, 992, 993, 994, 995, 997, 1000, 1001, 1001, 1005, 1007, 1009, 1011, 1015, 1019, 1021, 1021, 1022]

Sorted Array B of size: 300
[0, 1, 12, 14, 17, 18, 22, 25, 26, 29, 37, 37, 41, 42, 42, 48, 49, 49, 51, 52, 55, 57, 59, 62, 65, 66, 66, 70, 70, 82]
[84, 85, 86, 88, 89, 97, 99, 104, 113, 115, 120, 123, 125, 125, 128, 139, 141, 141, 142, 146, 147, 149, 155, 166, 170, 181, 182, 182, 183, 187]
[190, 196, 197, 202, 207, 222, 226, 226, 237, 239, 240, 242, 243, 245, 245, 245, 262, 263, 265, 265, 266, 266, 273, 274, 283, 284, 285, 287, 295, 298]
[300, 302, 303, 304, 305, 306, 319, 319, 321, 323, 327, 332, 340, 355, 359, 364, 368, 378, 380, 391, 392, 405, 407, 418, 419, 435, 457, 462, 466, 467]
[469, 470, 473, 481, 486, 488, 490, 501, 504, 505, 506, 507, 507, 508, 510, 513, 514, 519, 519, 520, 529, 531, 532, 536, 539, 544, 548, 551, 554, 557]
[557, 560, 562, 565, 568, 576, 584, 592, 596, 598, 599, 605, 610, 612, 612, 618, 624, 624, 633, 637, 638, 645, 650, 650, 658, 661, 662, 664, 664, 665]
[667, 669, 669, 678, 683, 686, 687, 691, 691, 692, 693, 696, 698, 708, 713, 714, 721, 722, 724, 729, 731, 741, 742, 742, 747, 750, 752, 753, 757, 758]
[760, 771, 771, 772, 772, 774, 775, 778, 786, 786, 788, 789, 789, 796, 800, 804, 806, 807, 835, 838, 839, 843, 845, 855, 856, 858, 860, 862, 867, 873]
[873, 876, 878, 882, 883, 884, 884, 887, 890, 894, 894, 901, 904, 905, 906, 908, 910, 912, 915, 919, 921, 926, 927, 931, 933, 935, 938, 939, 950, 954]
[955, 956, 957, 957, 958, 959, 960, 963, 963, 969, 972, 973, 980, 980, 981, 981, 987, 988, 990, 997, 999, 1001, 1003, 1008, 1010, 1012, 1013, 1015, 1024, 1024]

A is sorted: True
B is sorted: True

```

```

QUESTION 2:
Sorted Array C of size: 800
[0, 1, 3, 3, 4, 10, 10, 11, 12, 13, 14, 14, 15, 15, 17, 18, 21, 22, 25, 26, 26, 26, 29, 31, 33, 33, 33, 34, 35, 35]
[37, 37, 40, 41, 42, 42, 42, 45, 46, 47, 48, 48, 48, 49, 49, 51, 52, 53, 53, 54, 55, 55, 57, 59, 59, 60, 62, 62, 65, 66]
[66, 68, 69, 70, 70, 70, 79, 82, 84, 85, 86, 88, 88, 88, 89, 91, 97, 99, 100, 104, 108, 110, 113, 114, 115, 115, 116, 116, 117, 120, 122]
[123, 125, 125, 125, 126, 128, 128, 130, 131, 134, 135, 137, 138, 139, 139, 139, 141, 141, 142, 143, 143, 145, 146, 146, 147, 147, 148, 148, 149, 153]
[155, 155, 156, 157, 158, 159, 160, 163, 165, 166, 167, 168, 168, 169, 170, 170, 170, 175, 175, 176, 177, 178, 180, 180, 181, 181, 182, 182, 182, 183]
[184, 187, 189, 190, 193, 194, 194, 195, 196, 197, 197, 199, 202, 203, 205, 205, 206, 207, 207, 208, 209, 210, 211, 212, 217, 219, 221, 221, 222, 222]
[226, 226, 226, 234, 237, 237, 239, 240, 240, 240, 242, 243, 245, 245, 245, 252, 254, 255, 255, 256, 258, 261, 262, 262, 263, 263, 263, 265, 265]
[266, 266, 272, 273, 273, 274, 274, 274, 274, 277, 280, 281, 281, 281, 282, 283, 283, 283, 283, 284, 284, 285, 285, 287, 287, 290, 292, 292, 295, 295]
[297, 298, 300, 300, 302, 302, 303, 303, 304, 304, 305, 306, 309, 313, 314, 319, 319, 320, 321, 323, 325, 325, 326, 326, 327, 327, 329, 329, 330]
[331, 332, 332, 333, 336, 337, 339, 340, 340, 341, 349, 351, 354, 355, 357, 357, 358, 359, 359, 360, 362, 364, 368, 368, 368, 370, 373, 377, 378, 380]
[380, 380, 384, 384, 385, 391, 391, 391, 392, 393, 393, 395, 405, 405, 405, 406, 407, 409, 412, 413, 413, 413, 418, 418, 419, 421, 423, 428, 429, 432]
[435, 440, 441, 442, 444, 446, 452, 452, 457, 457, 460, 462, 464, 464, 466, 466, 467, 468, 469, 470, 473, 474, 481, 482, 482, 482, 483, 486, 488, 488]
[489, 490, 491, 495, 499, 500, 501, 504, 504, 505, 506, 507, 507, 508, 509, 510, 511, 513, 513, 514, 514, 515, 517, 519, 519, 519, 519, 520, 521, 521, 525]
[525, 527, 527, 529, 529, 531, 531, 532, 534, 535, 535, 536, 537, 539, 540, 541, 544, 544, 545, 548, 548, 551, 552, 553, 554, 554, 555, 556, 557, 557]
[559, 560, 561, 562, 563, 565, 568, 570, 570, 572, 575, 576, 578, 582, 584, 584, 584, 587, 589, 590, 590, 592, 593, 593, 594, 596, 598, 599, 601, 605]
[605, 605, 607, 607, 610, 610, 611, 611, 612, 612, 617, 618, 620, 621, 623, 624, 624, 626, 633, 635, 637, 637, 638, 638, 640, 641, 642, 642, 643, 645]
[647, 650, 650, 651, 655, 657, 657, 658, 659, 659, 661, 662, 662, 664, 664, 665, 665, 666, 667, 669, 669, 676, 678, 678, 679, 683, 686, 686, 687, 689]
[691, 691, 692, 693, 693, 693, 696, 698, 698, 700, 700, 703, 704, 704, 708, 708, 710, 713, 714, 715, 715, 716, 718, 718, 720, 721, 721, 721, 721, 721]
[722, 724, 729, 731, 731, 731, 734, 740, 741, 741, 742, 742, 742, 744, 746, 747, 750, 750, 750, 752, 752, 753, 755, 756, 757, 757, 758, 759, 760, 762]
[763, 764, 765, 767, 768, 769, 771, 771, 772, 772, 772, 774, 775, 776, 778, 778, 779, 779, 783, 784, 786, 786, 788, 789, 789, 791, 792, 793, 794, 795]
[795, 796, 798, 800, 802, 804, 804, 806, 806, 807, 807, 808, 809, 809, 814, 814, 815, 818, 820, 821, 825, 825, 827, 827, 828, 831, 831, 831, 835, 838]
[839, 842, 842, 843, 843, 845, 845, 845, 852, 854, 854, 855, 855, 856, 858, 858, 860, 860, 861, 862, 863, 865, 866, 866, 867, 867, 872, 873, 873, 874]
[876, 877, 878, 881, 881, 882, 883, 884, 884, 887, 888, 889, 889, 890, 890, 891, 891, 892, 892, 894, 894, 900, 901, 901, 903, 904, 905, 906, 907, 907]
[908, 910, 910, 912, 912, 913, 915, 915, 919, 919, 920, 921, 921, 926, 926, 927, 927, 930, 930, 931, 932, 933, 935, 938, 938, 939, 940, 942, 943, 945]
[945, 949, 950, 951, 954, 955, 955, 956, 957, 957, 957, 958, 959, 959, 959, 960, 960, 960, 963, 963, 963, 965, 967, 967, 967, 968, 968, 968, 969, 971]
[972, 973, 973, 974, 976, 976, 976, 977, 977, 980, 980, 980, 981, 981, 982, 986, 987, 988, 988, 989, 990, 991, 992, 993, 994, 995, 997, 997, 999, 1000]
[1001, 1001, 1001, 1003, 1005, 1007, 1008, 1009, 1010, 1011, 1012, 1013, 1015, 1015, 1019, 1021, 1021, 1022, 1024, 1024]

C is sorted: True

Process finished with exit code 0

```

Testing

Testing was done with use of the `all` function in python, which gives "True" if all elements of a given iterable are true. By giving the iterable as `A [i] <= A [i + 1]`, this ensured that every array produced in both Question 1 and 2 was in fact sorted.

Question 3: Finding Minimum and Maximum in an Array

Source Code

```

import random
A = [random.randint(0, 1024) for _ in range(10)] # Short array for quick testing
B = [1, 2, 3, 4, 5, 6, 7, 10, 11, 55] # Example of sorted array for testing
def find_extremes(arr):
    extremes = []
    for i in range(1, len(arr) - 1): # A[i] is not the first nor the last element of
        A
        # Check if i is larger or smaller than both neighbours
        if (arr[i - 1] < arr[i] > arr[i + 1]) or (arr[i - 1] > arr[i] < arr[i + 1]):
            extremes.append(arr[i]) # if it is, add it to the extremes array
    return extremes if extremes else "NONE, IT IS SORTED"

print("Array A Extremes:", find_extremes(A))

# TESTING
print("Array B Extremes:", find_extremes(B)) #This will always output sorted
# ANSWER TO QUESTION
print("I agree that an array has no extreme points if and only if it is sorted, as
      for an array to\n"
      "not have extreme points, it must be strictly ascending or descending; hence
      sorted")

```

Screenshot of the Output

```
Array A Extremes: [307, 913, 70, 843, 102]
Array B Extremes: NONE, IT IS SORTED
I agree that an array has no extreme points if and only if it is sorted, as for an array to
not have extreme points, it must be strictly ascending or descending; hence sorted

Process finished with exit code 0
```

Testing

Testing was done by ensuring that the part of the function printing that the algorithm is sorted works. This was done by feeding the function an array which is known to be sorted.

Question 4: Identifying Pairs with the Same Product

Source Code

```
import random
from collections import defaultdict

random_array = [random.randint(1, 1024) for _ in range(50)]

def find_pairs_same_product(arr):
    pairs_same_product = []
    product_map = defaultdict(list)
    seen_combinations = []

    for i in range(len(arr)): # Iterating over all pairs of i and j
        for j in range(i + 1, len(arr)):
            a = arr[i]
            b = arr[j]
            product = a * b

            for (c,d) in product_map[product]: # If any pair has the same product
                group = {a, b, c, d} # Creating a set to be checked for length, if 4,
                    all unique
                if len({a, b, c, d}) == 4: # If length is 4, all are unique due to
                    nature of set

                    if group not in seen_combinations:
                        pairs_same_product.append(((a, b), (c, d), product))
                        seen_combinations.append(group) # Mark as seen to avoid
                            duplicates

            product_map[product].append((a, b)) # Add pair only if another pair has
                same product

    return pairs_same_product

def print_results(arr): # Function for printing a fail case with no pairs found
    pairs = find_pairs_same_product(arr)
    if pairs:
        for pair1, pair2, product in pairs:
            print(f"pair 1:{pair1}, pair 2:{pair2}. Product is: {product}")
    else:
        print("No matching pairs found.")
print("Results for random array:")
print_results(random_array)

# TESTING: Done with two different arrays: one known to have no matching products,
    and one with known matching products
matching_array = [2, 16, 1, 32, 8, 4]
```

```

nomatching_array = [1, 2, 3, 5, 7]
print("\nTESTING: Results with a matched array:")
print_results(matching_array)

print("\nTESTING: Results with an array known to have no matches. Expected result is
      no matches found:")
print_results(nomatching_array)

```

Screenshot of the Output

```

Results for random array:
pair 1:(475, 164), pair 2:(820, 95). Product is: 77900
pair 1:(920, 205), pair 2:(820, 230). Product is: 188600
pair 1:(920, 120), pair 2:(230, 480). Product is: 110400
pair 1:(95, 120), pair 2:(475, 24). Product is: 11400
pair 1:(205, 480), pair 2:(820, 120). Product is: 98400
pair 1:(164, 120), pair 2:(820, 24). Product is: 19680
pair 1:(120, 861), pair 2:(504, 205). Product is: 103320
pair 1:(861, 480), pair 2:(820, 504). Product is: 413280

TESTING: Results with a matched array:
pair 1:(16, 1), pair 2:(2, 8). Product is: 16
pair 1:(16, 4), pair 2:(2, 32). Product is: 64
pair 1:(1, 32), pair 2:(2, 16). Product is: 32
pair 1:(1, 8), pair 2:(2, 4). Product is: 8
pair 1:(32, 4), pair 2:(16, 8). Product is: 128
pair 1:(8, 4), pair 2:(2, 16). Product is: 32
pair 1:(8, 4), pair 2:(1, 32). Product is: 32

TESTING: Results with an array known to have no matches. Expected result is no matches found:
No matching pairs found.

Process finished with exit code 0

```

Testing

Testing was done with two different arrays : one known to have no matching products , and one with known matching products. This ensured that the function worked as expected.

Question 5: Stack Implementation Using Abstract Data Type

Source Code

```

def evaluate_rpn(expr):
    stack = []

    for item in expr.split():
        if item.isdigit(): #If the item from the split is a number
            stack.append(float(item)) # Add the item to the stack
        else: #If the item is not a number (operand)
            b = stack.pop() # Popping in LIFO format
            a = stack.pop()
            print(f"Operation: {a} {item} {b}")

```

```

        if item == '+':
            result = a + b
        elif item == '-':
            result = a - b
        elif item == 'x':
            result = a * b
        elif item == '/':
            result = a / b
        result = round(result, 2)
        stack.append(result)
    print("Stack:", stack)
    return stack[0]

```

```
evaluate_rpn("4 6 + 2 x 3 / 8 - 10 +")
```

Screenshot of the Output

```

Stack: [4.0]
Stack: [4.0, 6.0]
Operation: 4.0 + 6.0
Stack: [10.0]
Stack: [10.0, 2.0]
Operation: 10.0 x 2.0
Stack: [20.0]
Stack: [20.0, 3.0]
Operation: 20.0 / 3.0
Stack: [6.67]
Stack: [6.67, 8.0]
Operation: 6.67 - 8.0
Stack: [-1.33]
Stack: [-1.33, 10.0]
Operation: -1.33 + 10.0
Stack: [8.67]

Process finished with exit code 0

```

Testing

Testing was done using an already existing RPN calculator on the internet and comparing the result:

<https://www.alcula.com/calculators/rpn/>

Question 6: Prime Number Generation Using Sieve of Eratosthenes

Source Code

```
import math

def is_prime(n): # Boolean function as it returns either True or False
    if n <= 1: # 1 and 0 are not primes; return false
        return False
    for i in range(2, int(math.sqrt(n) + 1)): # check if n is divisible by any number
        from 2 up to its sqrt
        if n % i == 0:
            return False
    return True

def sieve_erat(limit):
    primes = [True for _ in range(limit + 1)] # Assume all numbers are prime
    n = 2 # First prime number is 2

    while n * n <= limit: # checks all n from n to sqrt limit
        if primes[n]:
            for multiple in range(n * n, limit + 1, n):
                primes[multiple] = False
            n += 1
    primes_non_boolean = [num for num in range(2, limit+1) if primes[num]] # convert
        from boolean values to actual numbers
    return primes_non_boolean

#Part 1
n = int(input("PART 1: PRIME CHECKER \nEnter a number to check if it is prime: "))
if is_prime(n):
    print(f"{n} is prime")
else:
    print(f"{n} is not prime")

#Part 2
limit = int(input("\nPART 2: SIEVE OF ERATOSTHENES:\nEnter a number to find all prime
    numbers up to that number: "))
print(f"The following numbers are prime:\n", sieve_erat(limit))

# Explaining Optimizations made
print("\nOptimizations made:\nIn both algorithms, checks were made only till n
    reached the square root of the limit."
    "\nthis is because once this is passed, all smaller factors have been checked
    previously, and checking\n"
    "would be a waste of time.")

# Testing
print("\nTESTING:\nTesting will be done by passing values of the sieve algorithm
    through the is_prime algorithm")

for number in sieve_erat(limit):
    if not is_prime(number):
        print(f"Result: {number} is not prime, therefore test has failed.")
    else:
        print("Result: All values are prime.")
        break
```


Screenshot of the Output

```
PART 1: PRIME CHECKER
Enter a number to check if it is prime: 25
25 is not prime

PART 2: SIEVE OF ERATOSTHENES:
Enter a number to find all prime numbers up to that number: 50
The following numbers are prime:
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]

Optimizations made:
In both algorithms, checks were made only till n reached the square root of the limit.
this is because once this is passed, all smaller factors have been checked previously, and checking
would be a waste of time.

TESTING:
Testing will be done by passing values of the sieve algorithm through the is_prime algorithm
Result: All values are prime.

Process finished with exit code 0
```

Testing

The testing of the Sieve of Eratosthenes algorithm was performed by feeding the outputted array into the function which checks if all arrays in a number are prime. If all numbers in the array were prime, the Sieve algorithm was successfully implemented.

Question 7: Generating the Collatz Sequence

Source Code

```
import csv
def collatz_generator(n):
    col_sequence = []
    while n != 1:
        col_sequence.append(n)
        if n % 2 == 0: # following rules of the collatz sequence
            n = n // 2
        elif n % 2 == 1:
            n = 3 * n + 1
    return col_sequence

with open('collatz_sequence.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(collatz_generator(512))

# Testing
print("Testing was done by comparing to the result in a paper in the"
      "\n'International Journal of Mathematics and Mathematical Sciences':"
      "\n'Novel Theorems and Algorithms Relating to the Collatz Conjecture'")
```

Screenshot of the Output

```
collatz_sequence.csv 1 512,256,128,64,32,16,8,4,2

Testing was done by comparing to the result in a paper in the
'International Journal of Mathematics and Mathematical Sciences':
'Novel Theorems and Algorithms Relating to the Collatz Conjecture'

Process finished with exit code 0
```

Testing

Testing was done by comparing to the result in a paper in the International Journal of Mathematics and Mathematical Sciences: 'Novel Theorems and Algorithms Relating to the Collatz Conjecture'.

Question 8: Newton-Raphson Method for Root Finding

Source Code

```
def find_sqrt(n, guess, tolerance): # n = no. to find sqrt of 2
    next_guess = (guess + n / guess) / 2
    if abs(guess - next_guess) < tolerance:
        print("Final guess is:", next_guess)
        return next_guess
    return find_sqrt(n, next_guess, tolerance) # Implementing recursion to repeat the
        step

find_sqrt(327, 327/2, 0.00001) # Parameters equal to those in
# https://www.geeksforgeeks.org/find-root-of-a-number-using-newtons-method/

# Testing
print("Testing was done by comparing result to that found in link above. Result
    matches.")
```

Screenshot of the Output

```
Final guess is: 18.083141320025124

Testing was done by comparing result to that found in link above. Result matches.

Process finished with exit code 0
```

Testing

Testing was done by comparing the result to the result found in <https://www.geeksforgeeks.org/find-root-of-a-number-using-newtons-method/> by using the same parameters used in the site.

Question 9: Detecting Repeated Integers in a List

Source Code

```
arr_has_repeats = [1, 7, 7, 2, 3, 4, 4, 5, 6, 1, 55, 0, 0, 32, 3] # Repeated values:
    4, 7, 1, 0, 3
arr_no_repeats = [34, 15, 28, 2, 3, 44, 4]
# Try 1 (unsuccessful: although this performs the task, time complexity is O(n^2))
def find_repeats_try1(arr):
    repeats = []
```

```

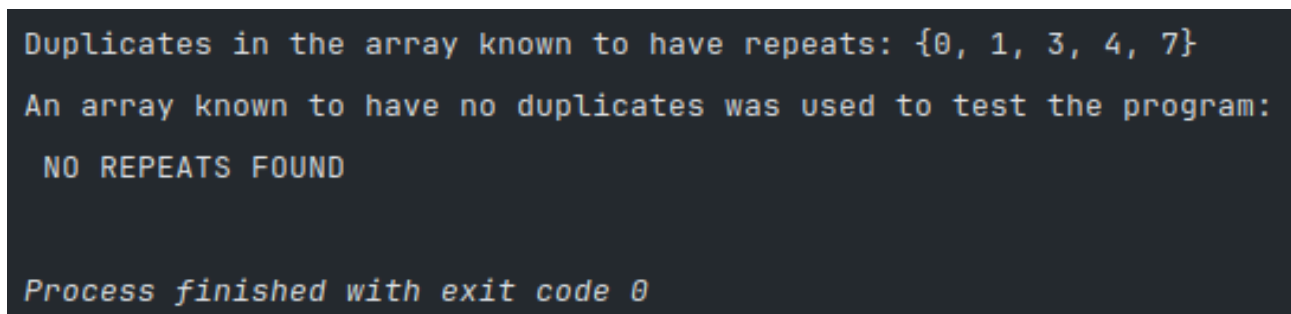
    for i in range(len(arr)):
        for j in range(len(arr)):
            if (i != j # if not comparing with itself
                and arr[i] == arr[j] # and value at i position is the same as that at
                    another
                and arr[i] not in repeats): # and has not already been listed as a repeat
                    repeats.append(arr[i]) # add it to the repeats array
    return repeats if repeats else "NO REPEATS FOUND"

# Try 2 (successful: O(n) time complexity)
def find_repeats(arr):
    seen = set()
    repeats = set()
    for number in arr:
        if number not in seen:
            seen.add(number)
        else:
            repeats.add(number)
    return repeats if repeats else "NO REPEATS FOUND"

print(f"Duplicates in the array known to have repeats:", find_repeats(arr_has_repeats
))
# Testing
print(f"An array known to have no duplicates was used to test the program:\n",
      find_repeats(arr_no_repeats))

```

Screenshot of the Output



```

Duplicates in the array known to have repeats: {0, 1, 3, 4, 7}
An array known to have no duplicates was used to test the program:
NO REPEATS FOUND

Process finished with exit code 0

```

Testing

Testing was done to ensure that the function also works for when the array has no repeated integers. This was done by feeding the function an array known to have no repeats.

Note

Two attempts were made in solving this question. Both were successful in completing the task, which is why I left both of them in the source code, however the first try has a time complexity of $O(n^2)$ while the second try has a time complexity of $O(n)$, and is therefore better.

Question 10: Finding the Largest Number Using Recursion

Source Code

```

def find_largest_number(arr, i = 0, highest = None):
    if i == len(arr): # Base Case: If reach the end of the list, return highest
        return highest
    if highest is None: # Case for the first run through (no highest set yet)
        highest = arr[i] # Set highest to the first number in array
    else:
        next_guess = arr[i] # Next guess will be current number in array
        if next_guess > highest: # if higher than previous number, new highest
            highest = next_guess

```

```

    return find_largest_number(arr, i+1, highest) # Recursive call

fifty_is_highest = [1, 49, 50, 22, 2, 33]
print("50 is the expected highest number. Program outputs:", find_largest_number(
    fifty_is_highest))

# Testing
print("\nThe program is tested by inputting and printing numerous arrays with known
    largest numbers")

test_arr1 = [1, 89, 23, 5, 465, 5]
print("465 is the expected largest number. Program outputs:", find_largest_number(
    test_arr1))
test_arr2 = [1, 89, 89, 89, 90]
print("90 is the expected largest number. Program outputs:", find_largest_number(
    test_arr2))

```

Screenshot of the Output

```

50 is the expected highest number. Program outputs: 50

The program is tested by inputting and printing numerous arrays with known largest numbers
465 is the expected largest number. Program outputs: 465
90 is the expected largest number. Program outputs: 90

Process finished with exit code 0

```

Testing

Testing was done by feeding the function arrays with known largest numbers, and seeing that the result matches.

Question 11: Approximating Cosine and Sine Using the Maclaurin Series

Source Code

```

import math

def cos(x, term_count): # Calculating Cosine
    while x > math.pi: # Keeping X in range of [-pi, pi]
        x -= 2 * math.pi
    while x < -math.pi:
        x += 2 * math.pi
    # FIRST TERM
    sign = 1
    numerator = 1
    denominator = 1
    result_cos = sign * numerator / denominator
    # Rest of terms
    for n in range(1, term_count):
        numerator *= x * x
        denominator *= (2 * n - 1) * (2 * n) # use previous denom in calculation
        sign *= -1 # Alternating signs
        result_cos += sign * numerator / denominator
    return result_cos

def sin(x, term_count): # Calculating sine
    while x > math.pi: # Keeping X in range of [-pi, pi]
        x -= 2 * math.pi
    while x < -math.pi:
        x += 2 * math.pi
    #FIRST TERM

```

```

    sign = 1
    numerator = x
    denominator = 1
    result_sin = sign * numerator / denominator
    # Rest of terms
    for n in range(1, term_count):
        numerator *= x * x
        denominator *= (2 * n + 1) * (2 * n)
        sign *= -1
        result_sin += sign * numerator / denominator
    return result_sin

print("Cos function calculation for first 5 terms and x = 5:", cos(50, 5))
print("Sin function calculation for first 5 terms and x = 5:", sin(50, 5))
# Testing
print("\nTesting: Comparing with in built functions of python:")
print("Cos:", math.cos(50))
print("Sin:", math.sin(50))

```

Screenshot of the Output

```

Cos function calculation for first 5 terms and x = 5: 0.9649660284925927
Sin function calculation for first 5 terms and x = 5: -0.26237485370393837

Testing: Comparing with in built functions of python:
Cos: 0.9649660284921133
Sin: -0.26237485370392877

Process finished with exit code 0

```

Testing

Testing was done by comparing the outputs of the functions approximating sine and cosine with the inbuilt functions in python.

Question 12: Generating the Fibonacci Sequence

Source Code

```

def first_n_fibonacci_numbers(n):
    sequence = []
    x = 0 # Starting values of x and y of the sequence
    y = 1
    for i in range(n):
        sequence.append(x)
        temp = x # use temp for old x as x will be updated in next step, and thus
                # affect y
        x = y
        y = temp + y
    return sequence

print(first_n_fibonacci_numbers(20))

# Testing
# Below is the first 20 numbers of the fibonacci sequence
# pasted from wikipedia. It will be used to compare the function's output
fibonacci_wikipedia = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610,
                        987, 1597, 2584, 4181]

```

```
print(f"wikipedia array is:\n", fibonacci_wikipedia)
```

Screenshot of the Output

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181]  
wikipedia array is:  
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181]  
  
Process finished with exit code 0
```

Testing

Testing was done visually by comparing side by side the output of the function and the array copied from Wikipedia.

3. Use of Generative AI

ChatGPT was used during this assignment. This was mainly the case for questions 4 and 11.

For Question 4, it was used to generate reasoning to use a dictionary data structure, as it did not come to mind when trying to implement it incorrectly in the first few attempts.

For Question 11, it was used to understand that x had to be limited to the range $[-\pi, \pi]$, as the output was always growing, and I could not understand why.

4. 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).

Dave Galea
Student Name


Signature

Student Name

Signature

Student Name

Signature

Student Name

Signature

ICT-1018
Course Code

Data Structures and Algorithms
Title of work submitted

10th May 2025
Date