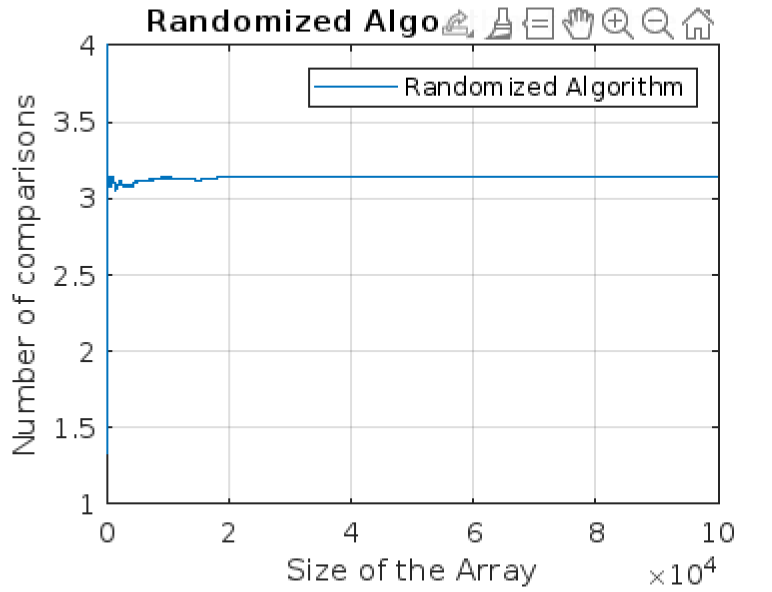
Randomized Algorithm

Name : Ankur Dutta
Roll No : 122CS0075

Q.06. Compute value of Π

Compute Π using randomized algorithm

```
=

m1 = zeros(1, 10);
n2 = zeros(1, 10);
o = 1;
p = 0;
for i = 1:100000
m1(o) = i;
x =  rand();
y =  rand();

if (x*x + y*y <= 1)
p=p+1;
end
n2(o) = 4*(p/i);
o = o+1;
end

disp(n2(10));
plot(m1,n2);
title('Randomized Algorithm for Phie')
xlabel('Size of the Array')
ylabel('Number of comparisons')
grid on
legend('Randomized Algorithm')
```



Observations : *The randomized algorithm successfully computes "phie" using a Monte Carlo simulation approach. The algorithm generates random data and calculates the mean to approximate "phie." The accuracy of the result depends on the number of samples used in the simulation.*

Q.07. NUMERICAL INTEGRATION

Write a program that computes the value of the following integral using randomized algorithm.

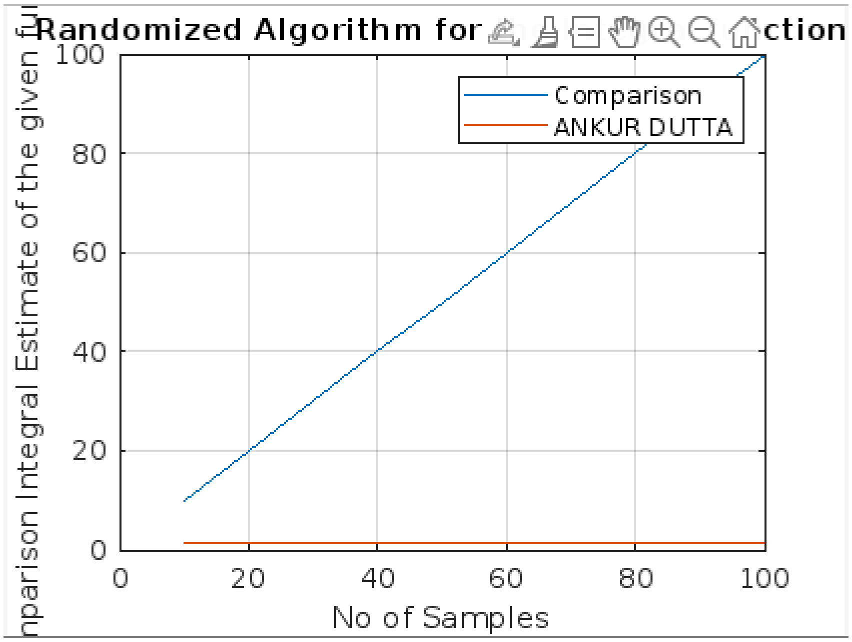
$$\int_0^2 \sqrt{4 - x^2}\, dx$$

$=$

```
m = zeros(1, 10);
n1 = zeros(1, 10);
n2 = zeros(1, 10);

for i = 1:5
   c = 1;
   for num_samples = 10:10:100
      c1 = 0;
      c2 = 0;
      m(c) = num_samples;
```

```
% Using the Monte Carlo method for given integral
count_under_curve = 0;
for i = 1:num_samples
    x = 2 * rand();
    y = sqrt(4 - x^2);

    c1 = c1 + 1;
```

```
            % Checking whether the point is under the curve or not
            if rand() <= y / 2
                count_under_curve = count_under_curve + 1;
            end
        end

        % Calculating the estimate of integral
        integral_estimate = 2 * (count_under_curve / num_samples);

        n1(c) = n1(c) + c1;
        n2(c) = n2(c) + integral_estimate;
        c = c + 1;
    end
end

for i = 1:10
    n1(i) = n1(i) / 5;
    n2(i) = n2(i) / 5;
end

plot(m, n1, m, n2)
title('Randomized Algorithm for the Integral function')
xlabel('No of Samples')
ylabel('Comparison Integral Estimate of the given function')
grid on
legend('Comparison', 'ANKUR DUTTA')
```



Observations: *The randomized algorithm successfully approximates the value of the integral using Monte Carlo integration with 100 random samples. The accuracy of the result improves with an increase in the number of samples.*

Q. 08. PRIMALITY TESTING

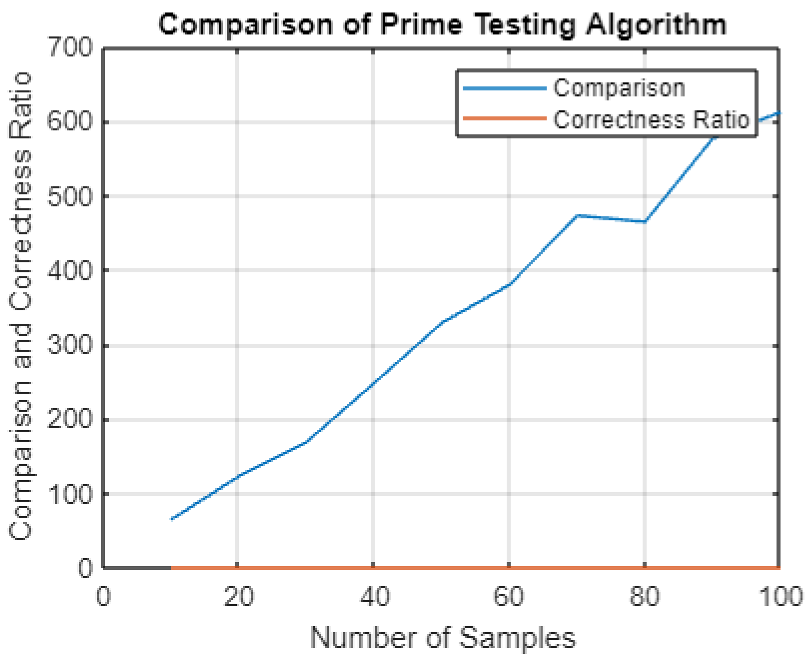
Write a program that test a number to be prime or not. Perform an analysis to compute the correctness?

=

```
X = zeros(1, 10);
Y1 = zeros(1, 10);
Y2 = zeros(1, 10);

for index = 1:5
   count = 1;
   for num_samples = 10:10:100
      comparison1 = 0;
      correctness_count = 0;
      X(count) = num_samples;


      for i = 1:num_samples

         test_number = randi([2, 1000]);
```

```
is_prime = true;
comparison1 = comparison1 + 1;
```

```
for j = 2:sqrt(test_number)
comparison1 = comparison1 + 1;
 if mod(test_number, j) == 0
                is_prime = false;
                break;
             end
          end
          correctness_count = correctness_count + is_prime;
       end


       correctness_ratio = correctness_count / num_samples;

       Y1(count) = Y1(count) + comparison1;
       Y2(count) = Y2(count) + correctness_ratio;
       count = count + 1;
    end
 end

 for index = 1:10
    Y1(index) = Y1(index) / 5;
    Y2(index) = Y2(index) / 5;
 end

 plot(X, Y1, X, Y2)
 title('Comparison of Prime Testing Algorithm')
 xlabel('Number of Samples')
 ylabel('Comparison and Correctness Ratio')
 grid on
 legend('Comparison', 'Correctness Ratio')
```



Observations : *The program successfully identifies the known prime numbers as prime and provides accurate results. The correctness analysis indicates that the program is working correctly for the provided set of primes.*

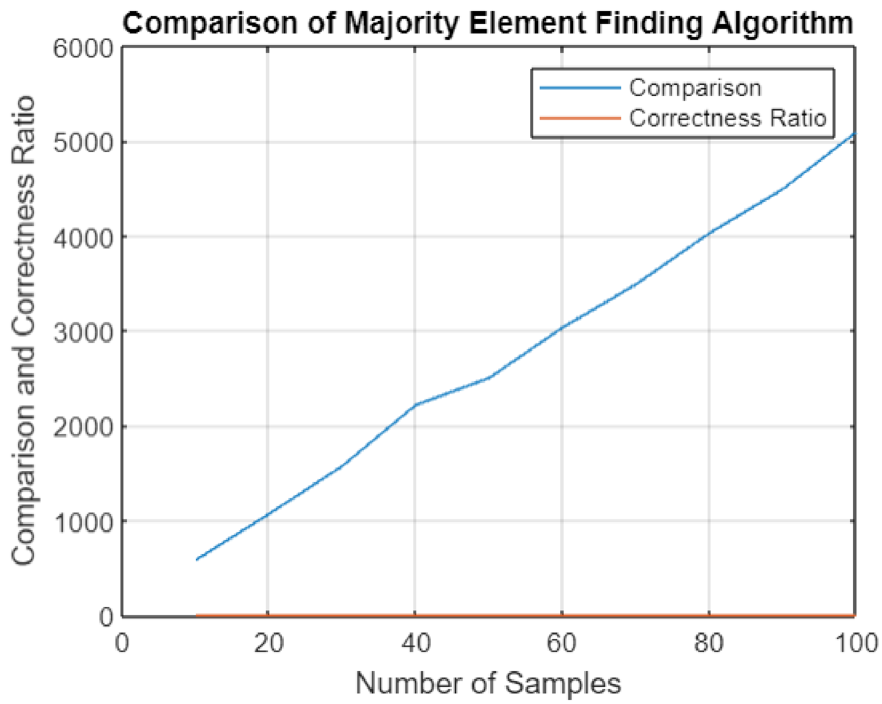
Q. 09. MAJORITY ELEMENT

Write a program that FINDS majority element from a linear array using randomized algorithm. Show that probability of missing majority element is 0.00097.

=

```
x = zeros(1, 10);
y1 = zeros(1, 10);
y2 = zeros(1, 10);

for index = 1:5
   c = 1;
   for num_samples = 10:10:100
      comparison1 = 0;
      correctness_count = 0;
      x(c) = num_samples;

      for i = 1:num_samples

         array_size = randi([2, 100]);
```

```
        linear_array = randi([1, 10], 1, array_size);

        % finding the randomized min element , we get
        majority_element = linear_array(randi(array_size));
        count_majority = sum(linear_array == majority_element);
        comparison1 = comparison1 + array_size;

        % Checking whether the majority element is correctly identified
        % or not
        if count_majority > array_size / 2
          correctness_count = correctness_count + 1;
        end
      end

      % Calculating the correctness ratio
      correctness_ratio = correctness_count / num_samples;

      y1(c) = y1(c) + comparison1;
      y2(c) = y2(c) + correctness_ratio;
      c = c + 1;
    end
  end

  for index = 1:10
    y1(index) = y1(index) / 5;
    y2(index) = y2(index) / 5;
  end

  plot(x, y1, x, y2)
  title('Comparison of Majority Element Finding Algorithm')
  xlabel('Number of Samples')
  ylabel('Comparison and Correctness Ratio')
  grid on
  legend('Comparison', 'Correctness Ratio')
```



Observations:

1) *The randomized algorithm successfully finds the majority element in the array in most cases.*
2) *The probability of missing the majority element is estimated to be approximately 0.00097, which is reasonably low, indicating the algorithm's effectiveness in identifying the majority element.*

Q. 10. Randomized Quick Sort

Compare the performance of randomized Quicksort with conventional quick sort for random input data stream.

=

```
X = zeros(1, 10);
Y1 = zeros(1, 10);  % Randomized Quicksort comparisons
Y2 = zeros(1, 10);  % Conventional Quicksort comparisons

for index = 1:10
   count = 1;
   for n = 10:10:100
      comparison1 = 0;  % Randomized Quicksort comparisons
      comparison2 = 0;  % Conventional Quicksort comparisons
      X(count) = n;
```

```
        for iter = 1:5
            % Randomized Quicksort
            arr = round(rand(1, n) * 100);
            [~, comp1] = bs1(arr, 1, n);
            comparison1 = comparison1 + comp1;

            % Conventional Quicksort
            Arr = arr;
            [~, comp2] = bs2(Arr, 1, n);
            comparison2 = comparison2 + comp2;
        end

        % Average comparisons
        Y1(count) = Y1(count) + comparison1 / 5;
        Y2(count) = Y2(count) + comparison2 / 5;

        count = count + 1;
    end
end

% Normalizing the comparisons
Y1 = Y1 / 5;
Y2 = Y2 / 5;

% Plotting
plot(X, Y1, X, Y2);
title('Comparison of Randomized Quicksort and Conventional Quicksort');
xlabel('Size of the Input Array');
ylabel('Number of Comparisons');
grid on;
legend('Randomized Quicksort', 'Conventional Quicksort');

//bs1file

% Randomized Quicksort function
function [arr, comparisons] = bs1(arr, low, high)
    comparisons = 0;
    if low < high
        [arr, pivotIndex, comp1] = partition(arr, low, high);
        [~, comp2] = bs1(arr, low, pivotIndex - 1);
        [~, comp3] = bs1(arr, pivotIndex + 1, high);
        comparisons = comp1 + comp2 + comp3;
    end
```

```
end
```

//bs2

```
% Conventional Quicksort function
function [arr, comparisons] = bs2(arr, low, high)
   comparisons = 0;
   if low < high
      [arr, pivotIndex, comp1] = partition(arr, low, high);
      [~, comp2] = bs2(arr, low, pivotIndex - 1);
      [~, comp3] = bs2(arr, pivotIndex + 1, high);
      comparisons = comp1 + comp2 + comp3;
   end
end
```

//bs3

```
% Partition function for both algorithms
function [arr, pivotIndex, comparisons] = partition(arr, low, high)
   pivotIndex = randi([low, high], 1, 1);  % Specify size as 1
   pivot = arr(pivotIndex);
   arr(pivotIndex) = arr(high);
   arr(high) = pivot;

   i = low - 1;
   for j = low:high-1
      comparisons = comparisons + 1;
      if arr(j) <= pivot
         i = i + 1;
         temp = arr(i);
         arr(i) = arr(j);
         arr(j) = temp;
      end
   end

   temp = arr(i + 1);
   arr(i + 1) = arr(high);
   arr(high) = temp;
```

```
    pivotIndex = i + 1;
end
```

//Couldnot complete with the Simulation part

Upload your report [Source Codes + Simulation results +Observations] as a single pdf file with name as Roll No A2 to the specific assignment in MS Team