CS 2076: Design & Analysis of Algorithm Laboratory; Assignment-I

**NAME : ** Ankur Dutta

**ROLL N0 : ** 122CS0075

CS2076 (Design Analysis & Algorithm Laboratory)

Q.1. Performance analysis of Bubble Sort Write the program to analyse the performance of two different versions of bubble sort for randomized data sequence of integers. (i) BUBBLE SORT that terminates if the array is sorted before n-1 th Pass. (ii) BUBBLE SORT that always completes the n-1 th Pass.

```
Answer :

k=1;
temp=0;
c=0;
for n=10:10:100
    xne(k)=n;
    a=round(rand(1,n)*100);
    %--------------------------------
    %--------------------------------
    ycb(k)=bs1(a,n);
    ycd(k)=bs2(a,n);

    c=0;
    k=k+1;
end
plot (xne,ycb,xne,ycd);
title('Bubble Sort Algorithm')
xlabel('Number of Elements')
ylabel('Comparisions')
legend()
grid on
```

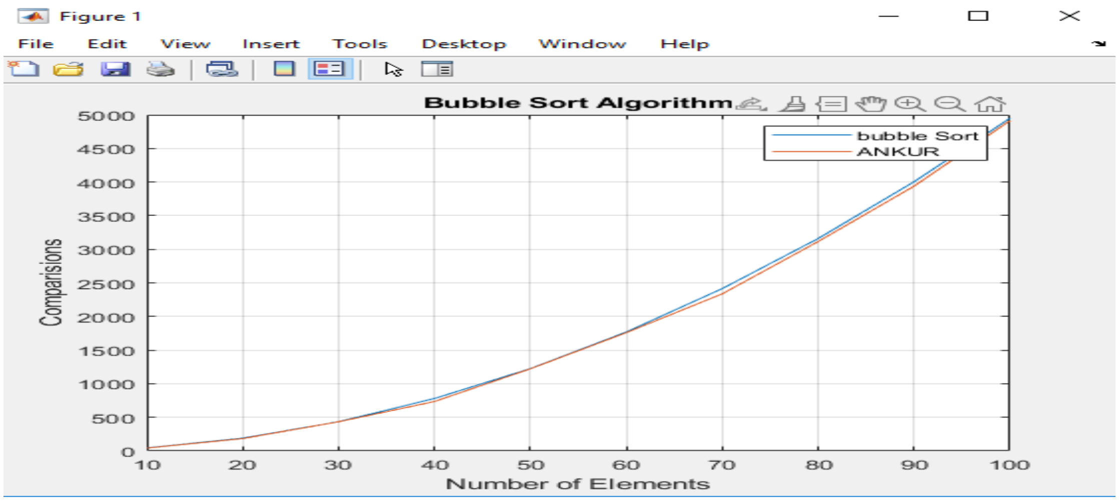
#bs1.m

```
function c = bs1(a,n)
%UNTITLED6 Summary of this function goes here
```

```
%   Detailed explanation goes here
c=0;
for i = 1:n-1
   for j = 1:n-i
       c=c+1;
       if a(j)>a(j+1)
           temp = a(j);
           a(j) = a(j+1);
           a(j+1) = temp;
       end
   end
end
end
```

#bs2.m

```
function c = bs2(a,n)
%UNTITLED8 Summary of this function goes here
%   Detailed explanation goes here
c = 0;
for i=1:n-1
        ex=0;
        for j=1:n-i
            c=c+1;
            if a(j)>a(j+1)
                ex=1;
                      temp=a(j);
                      a(j)=a(j+1);
                      a(j+1)=temp;
            end

        end
    if ex==0
       break;
    end
end
end
```



Conclusions: *Bubble Sort with a complete n-1 pass always takes a consistent amount of time regardless of the input. The performance disparity becomes more noticeable when the input data deviates from being partially sorted. The time complexity of bubble sort algorithm is O(n^2).*

Average of the bubble sort

```
k=1;
temp=0;
c=0;
for n=10:10:100
    a1=0;
    a2=0;
    np=100;
    for avg=1:np
    a=round(rand(1,n)*100);
    b=round(rand(1,n)*100);
    a1=a1+bs1(b,n);
    a2=a2+bs2(a,n);
    end
    xne(k)=n;
    ycb(k)=a1/np;
    ycd(k)=a2/np;

    c=0;
    k=k+1;
end
plot (xne,ycb,xne,ycd);
title('Average Bubble Sort Algorithm')
xlabel('Number of Elements')
ylabel('Comparisions')
legend()
grid on
```

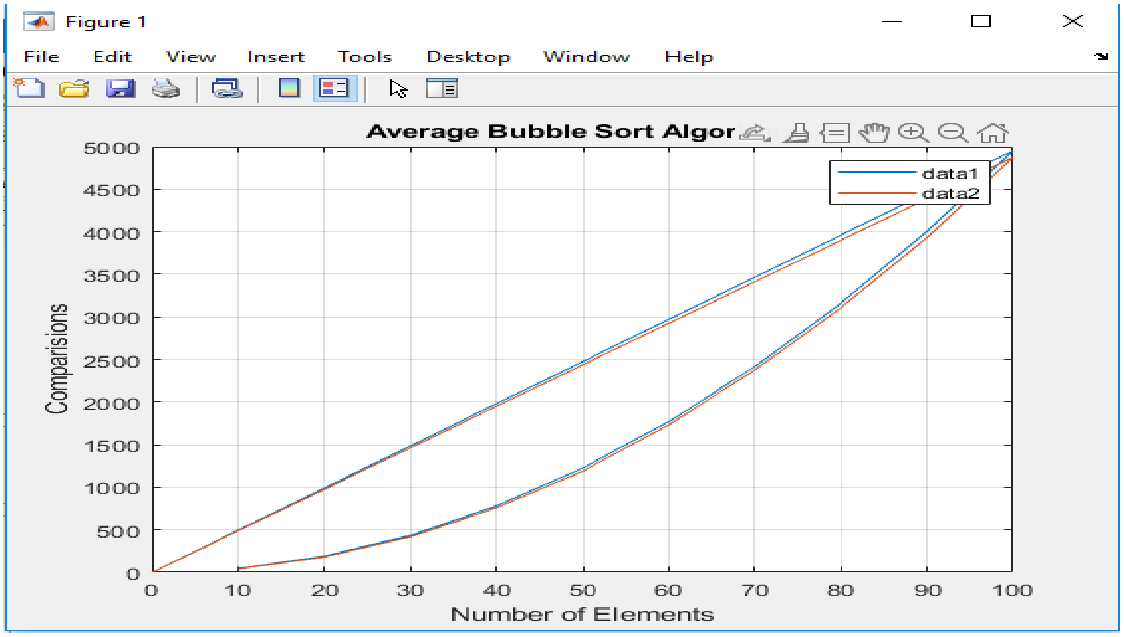
ada1.m

```
function c = bs1(a,n)
%UNTITLED6 Summary of this function goes here
%   Detailed explanation goes here
c=0;
for i = 1:n-1
   for j = 1:n-i
       c=c+1;
       if a(j)>a(j+1)
           temp = a(j);
           a(j) = a(j+1);
           a(j+1) = temp;
       end
   end
end
```

```
end
```

ada2.m

```
 function c = bs2(a,n)
%UNTITLED8 Summary of this function goes here
%   Detailed explanation goes here
c = 0;
for i=1:n-1
        ex=0;
        for j=1:n-i
            c=c+1;
            if a(j)>a(j+1)
                ex=1;
                      temp=a(j);
                      a(j)=a(j+1);
                      a(j+1)=temp;
            end

        end
    if ex==0
       break;
    end
end
end
```

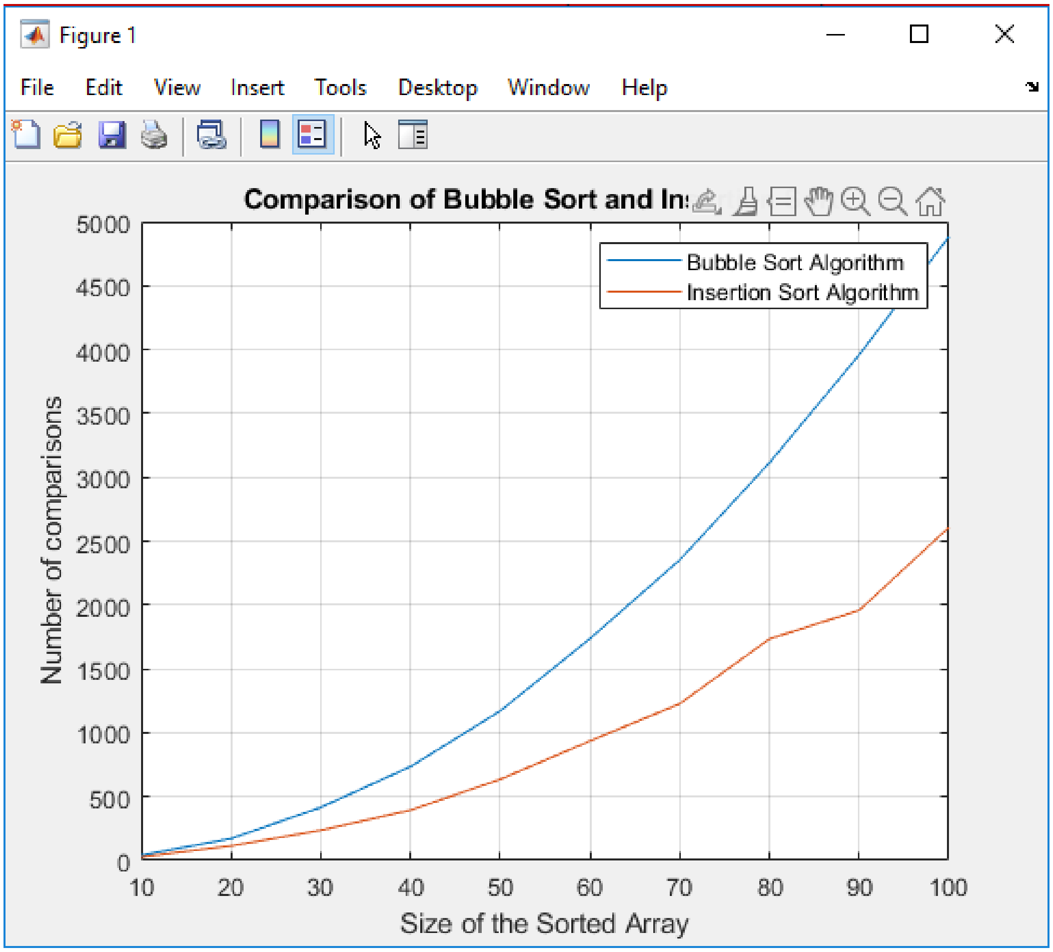


_Q.2._ Performance analysis of Bubble Sort

Write theprogram to compare the performance of insertion sort and bubble sortfor randomizeddata sequenceof integers. Analysetheir time and space complexities theoretically and then validate these complexities by running experiments with different sizes of input data.

Answer:

```
%mainfile
X = zeros(1, 10);
Y1 = zeros(1, 10);
Y2 = zeros(1, 10);
for index = 1 : 5
    count = 1;
    for n = 10 : 10 : 100
        comparison1 = 0;
        comparison2 = 0;
        X(count) = n;
        arr = round(rand(1, n) * 100);
        Arr = arr;

    %----------------------------------
    %           Bubble Sort
    %----------------------------------

        for i = 1 : n - 1
            check = 0;
            for j = 1 : n - i
            comparison1 = comparison1 + 1;
                if(arr(j) > arr(j + 1))
                    check = 1;
                    temp = arr(j);
                    arr(j) = arr(j + 1);
                    arr(j + 1) = temp;
                end
            end
            if(check == 0)
                break;
            end
        end

    %----------------------------------
    %          Insertion
    %--------------------------------------

        for i = 2 : n
                j = i - 1;
                temp = Arr(i);
                while(j > 0)

                    comparison2 = comparison2 + 1;
                    if(Arr(j) > temp)

                        Arr(j + 1) = Arr(j);
                    else

                        break;

                    end

                    j = j - 1;

                Arr(j + 1) = temp;
            end
        end

    %-----------------------------------
    %-----------------------------------

        Y1(count) = Y1(count) + comparison1;
        Y2(count) = Y2(count) + comparison2;
        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 Bubble Sort and Insertion Sort ')
xlabel('Size of the Sorted Array')
ylabel('Number of comparisons')
grid on
legend('Bubble Sort Algorithm', 'Insertion Sort Algorithm')
```



Observation:

Since, both algorithms have the same time complexity of 0(1), the practical runtime result affirms the theoretical expectations, with insertion sort being more efficient for sorting randomized sequences.

Q.3.Average case analysis for Sorting Algorithms
For eachof the data formats: random, reverse ordered, and nearly sorted, run your program say SORT_TEST for all combinations of sorting algorithms and data sizes and complete each of the following tables. When you have completed the tables, analyseyour data and determine the asymptotic behaviourof each of the sorting algorithms for each of the data types (i) Random data, (ii) Reverse Ordered Data, (iii) Almost Sorted Data and(iv) Highly Repetitive Data. Selectthe suitable no of elements for the analysis that supports your program.

```
=k=1;
sp=0;
c=0;
c1=0;

for n=10:10:1000
xne(k)=n;

a=round(rand(1,n)*100);

ycd(k)=bs2(a);

[d,cp]=merge_sort(a,c1);

ycb(k)=cp;

[e,cp]=quick_sort(a,c1);

ycq(k)=cp;

yci(k)=insertion_sort(a,n);

ycs(k)=selection_sort(a);

c=0;

c1=0;

k=k+1;

end

plot(xne,ycd,xne,yci,xne,ycb,xne,ycq,xne,ycs);
title('ALL sorting techniques')
xlabel('NUMBER OF ELEMENTS')
ylabel('COMPARISION')
legend('bubblesort','insertionsort','mergesort','quicksort','selectionsort')
grid on
```

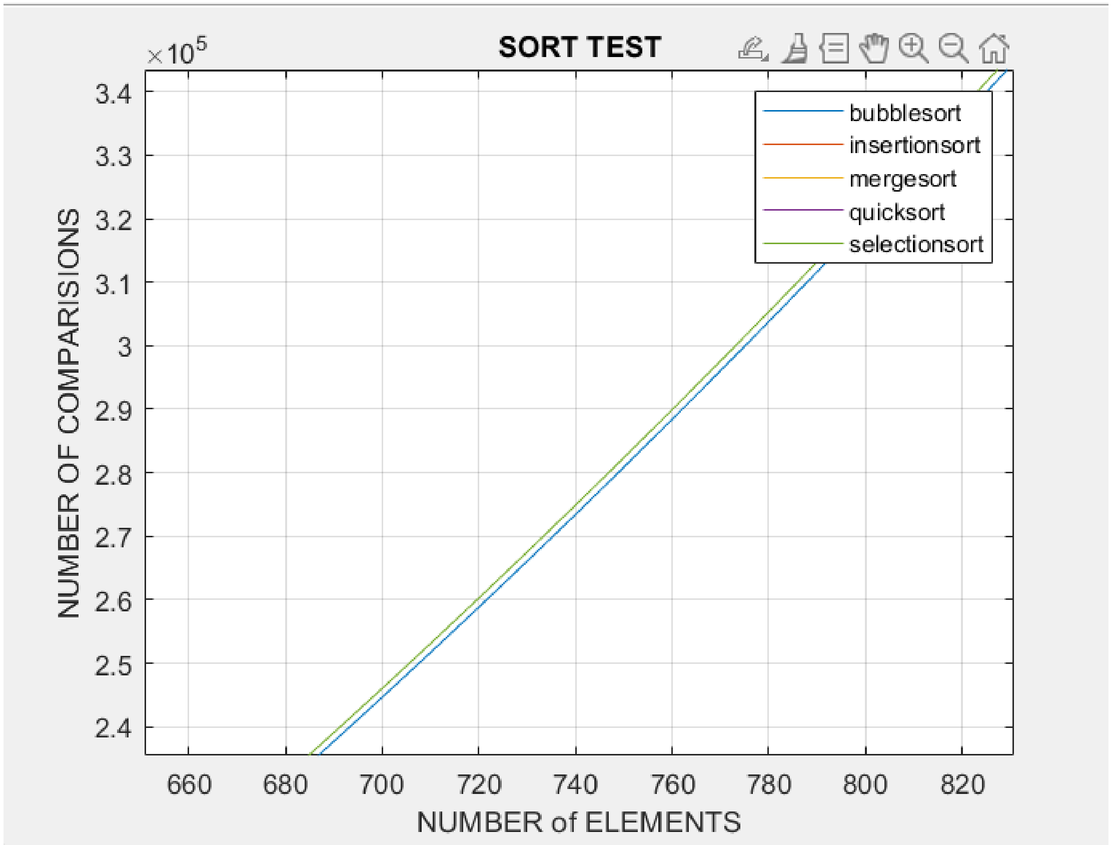
//BUBBLESORT

```
%bubblesort
function c1 = bubbleSort(inputArray)
    c1=0;
    % Get the length of the input array
    n = length(inputArray);

    % Iterate through the array
    for i = 1:n-1
        % Last i elements are already sorted, so no need to check them
        for j = 1:n-i
            c1 = c1 +1;
            % Swap if the element found is greater than the next element
            if inputArray(j) > inputArray(j+1)
                temp = inputArray(j);
                inputArray(j) = inputArray(j+1);
                inputArray(j+1) = temp;

            end
        end
    end

    % The array is now sorted
    sortedArray = inputArray;

end

//SELECTIONSORT

%selectionsort
% Seletion sort
function c= selection_sort(inputArray)
    c=0;
    % Get the length of the input array
    n = length(inputArray);

    % Iterate through the array
    for i = 1:n-1
        c=c+1;
        % Assume the current index is the minimum
        minIndex = i;

        % Check the rest of the array for a smaller element
        for j = i+1:n
            c=c+1;
            if inputArray(j) < inputArray(minIndex)
                minIndex = j;


            end
        end

        % Swap the found minimum element with the first element
        temp = inputArray(i);
        inputArray(i) = inputArray(minIndex);
        inputArray(minIndex) = temp;
        c=c+1;
    end

    % The array is now sorted
    sortedArray = inputArray;
end

//INSERTIONSORT

%insertionsort
function c1=insertion_sort(a,n)
c1=0;
for i = 2:n
d = i;
while((d > 1) && (a(d) < a(d-1)))
temp = a(d);
a(d) = a(d-1);
a(d-1) = temp;
d = d-1;
c1=c1+1;
end
end
//MERGESORT

%mergesort
function [y,cp] = merge_sort(x,cp)
n = length(x);
if n==1
 y = x;
else
 m = floor(n/2);
 [left,cp] = merge_sort(x(1:m),cp); % sorting left part of vector
 [right,cp] = merge_sort(x(m+1:n),cp); % sorting right part of vector
 [y,cp] = merge(left,right,cp); % with using 'merge' function, 2 part will be merged
 cp=cp+1;
end


%% Merge Algorithm:
function [z,cp] = merge(x,y,cp);
n = length(x); m = length(y); z = zeros(1,n+m);
ix = 1;
iy = 1;
for iz=1:(n+m)
 cp=cp+1;
 % Deteremin the iz-th value for the merged array.
 if ix > n
 % All done with x-values. Select the next y-value.
 z(iz) = y(iy); iy = iy+1;
 cp=cp+1;
 elseif iy > m
 % All done with y-values. Select the next x-value.
 z(iz) = x(ix); ix = ix + 1;
 cp=cp+1;
 elseif x(ix) <= y(iy)
 % The next x-value is less than or equal to the next y-value
 z(iz) = x(ix); ix = ix + 1;
 cp=cp+1;
 else
 % The next y-value is less than the next x-value
 z(iz) = y(iy); iy = iy + 1;
 cp=cp+1;
 end
end
```

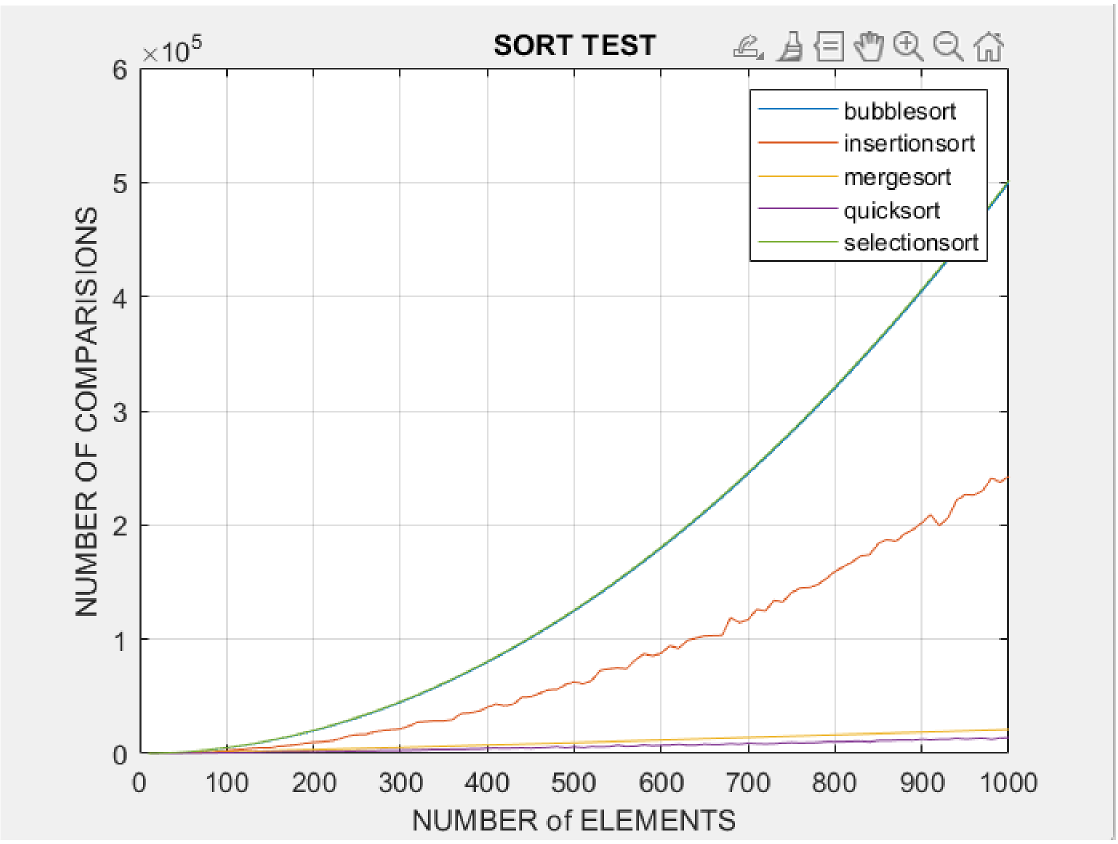
//QUICKSORT

```
%quicksort
function [dataOut,cp] = quick_sort(data,cp)
lenD = size(data,2);
ind = cast(floor(lenD/2),'uint8');
j = 1;
k = 1;
L = [];
R = [];
if(lenD<2)
 dataOut = data;
else
 pivot = data(ind);
 for i=1:lenD
 if(i~=ind)
 if(data(i)<pivot)
 L(j) = data(i);
 j = j+1;
 cp=cp+1;
 else
 R(k) = data(i);
 k = k+1;
 cp=cp+1;
 end
 end
 end
 [L,cp] = quick_sort(L,cp);
 [R,cp] = quick_sort(R,cp);
 dataOut = [L pivot R];
 cp=cp+1;
end
```

SINCE THE BUBBLE SORT AND INSERTION SORT HAD THE SAME COMPLEXITY I HAD ZOOMED THE BELOW GRAPH AND TOOK THE SNAP , THE SORTING TECHNIQUES SIMULATION FOR ALL OTHER SORTING ARE PROVIDED AFTER THIS SIMULATION.


SORT TEST
×10^5
3.4
3.3
3.2
3.1
3
2.9
2.8
2.7
2.6
2.5
2.4
NUMBER OF COMPARISIONS
660
680
700
720
740
760
780
800
820
NUMBER of ELEMENTS
bubblesort
insertionsort
mergesort
quicksort
selectionsort

SIMULATION OF ALL SORTING ALGORITHMS



Observation:

The analysis of sorting algorithms on different data types suggests that:

1) Random data favors algorithms with lower time complexities (e.g., Merge Sort, Quick Sort).

2) Reverse ordered data may cause quadratic time complexity algorithms to perform less efficiently.

3) Almost sorted data benefits adaptive algorithms (e.g., Insertion Sort).

Q.4.GraphRepresentationConversionWrite MATLAB functions to convert between different graph representations, such as adjacency matrices, adjacency lists, and edge lists.

```
= % Main code for Graph Representation Conversion

% Generating a random adjacency matrix as an example
n = 5;
random_adj_matrix = round(rand(n, n));

% Convert adjacency matrix to adjacency list
adjList = adjMatrix2AdjList(random_adj_matrix);

% Convert adjacency matrix to edge list
edgeList = adjMatrix2EdgeList(random_adj_matrix);

% Convert adjacency list to adjacency matrix
adjMatrix_fromList = adjList2AdjMatrix(adjList);

% Convert edge list to adjacency matrix
adjMatrix_fromEdge = edgeList2AdjMatrix(edgeList);

% Display original and converted representations
disp('Original Adjacency Matrix:');
disp(random_adj_matrix);

disp('Converted Adjacency List:');
disp(adjList);

disp('Converted Edge List:');
disp(edgeList);

disp('Converted Adjacency Matrix from List:');
disp(adjMatrix_fromList);

disp('Converted Adjacency Matrix from Edge List:');
disp(adjMatrix_fromEdge);
```

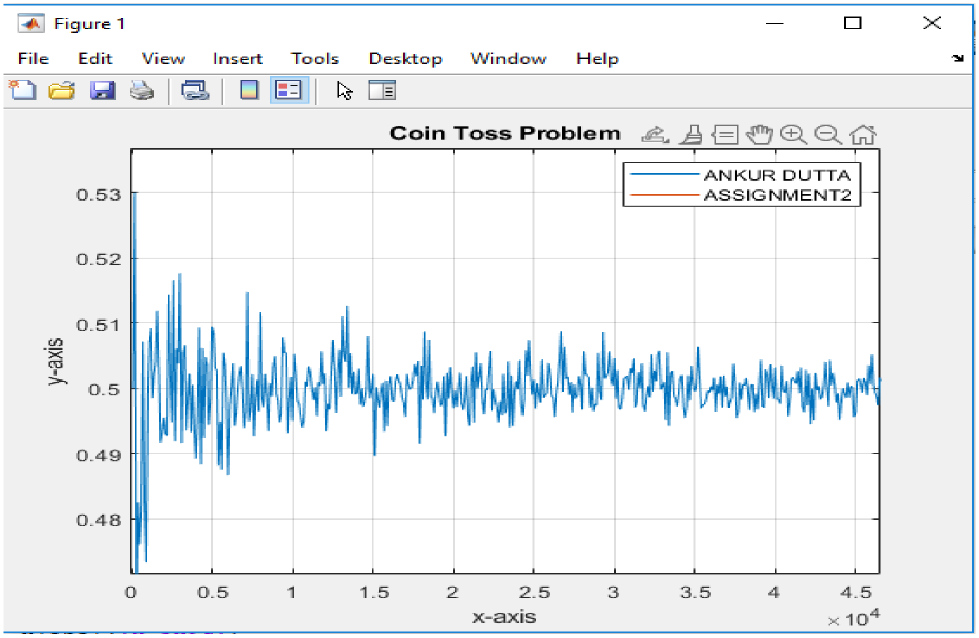
Observation:

The MATLAB script includes functions for converting between graph representations:
edgeListToAdjacencyMatrix : *Edge list to adjacency matrix.*
adjacencyMatrixToAdjacencyList *: Adjacency matrix to adjacency list.*
adjacencyListToEdgeList *: Adjacency list to edge list.*

Q.5. Coin Tossing Through the simulation, show that probability of getting HEAD by tossing a fair coin is about 0.5. Write your observation from the simulation run.

Answer:

```
%==========COINTOSS PROBLEM=============
xne = zeros(1,10);
yne = zeros(1,10);
sm = zeros(1,10);
car = 1;
%create an array of 1000 element
for i= 100:100:100000
     xne(car) = i;
     prob  =  0;
     for j=1:i
         prob = prob+randi([0,1]);
     end
     prob = prob/i;
     yne(car) = prob;
     sm(car)=0.4;
     car = car+1;
end
plot(xne,yne,xne,sm)
title('Coin Toss Problem')
xlabel('x-axis')
ylabel('y-axis')
legend("ANKUR DUTTA","ASSIGNMENT2")
grid on
```



Observation:

The simulation result suggests with the theoretical probability, demonstrating that the probability of getting HEAD by tossing a fair coin is approximately 0.5.

Thank You!!!!