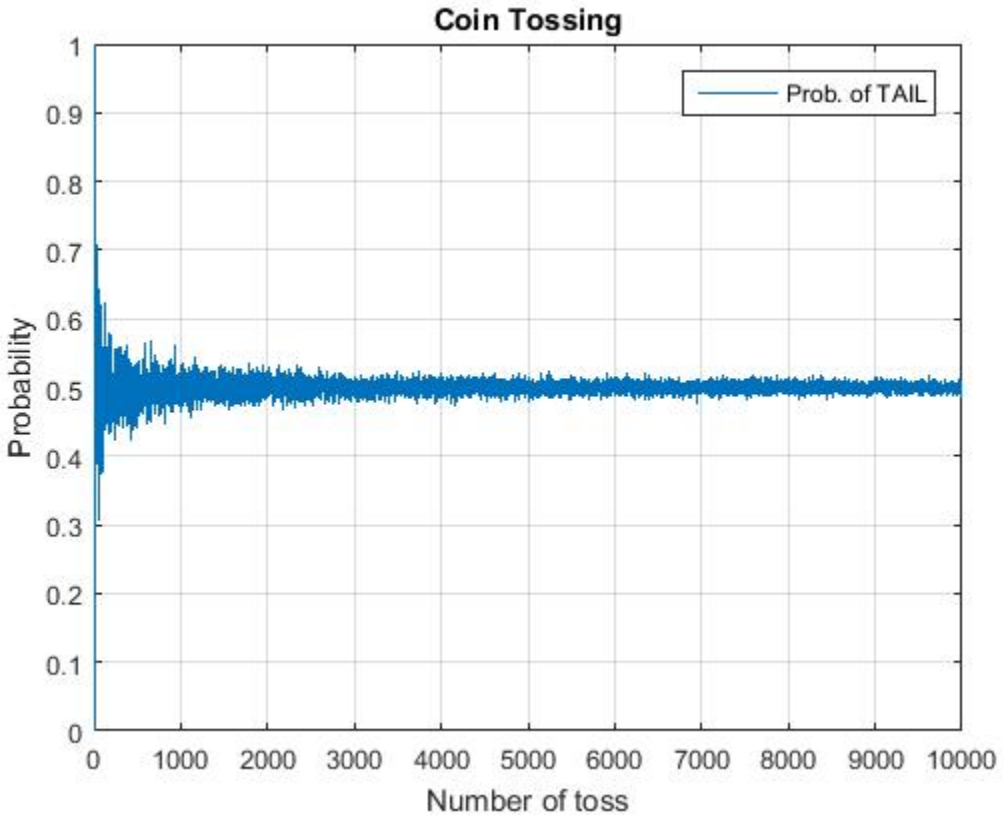
Q.1. Coin Tossing

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.

```
%clear all;
prob_1 = zeros(1,10000);
c = zeros(1,10000);
for n = 1:10000
    A = round(rand(1,n));
    count_1 = 0;
    c(n) = n;
    for i = 1:n
        if A(i) == 1    % HEAD is denoted by 0.
            count_1 = count_1 + 1;
        end
    end
    prob_1(n) = count_1 / n;
end
plot(c,prob_1);
title('Coin Tossing');
xlabel('Number of toss');
ylabel('Probability');
disp('Analysis of coin tossing');
legend('Prob. of TAIL');
grid on;
```



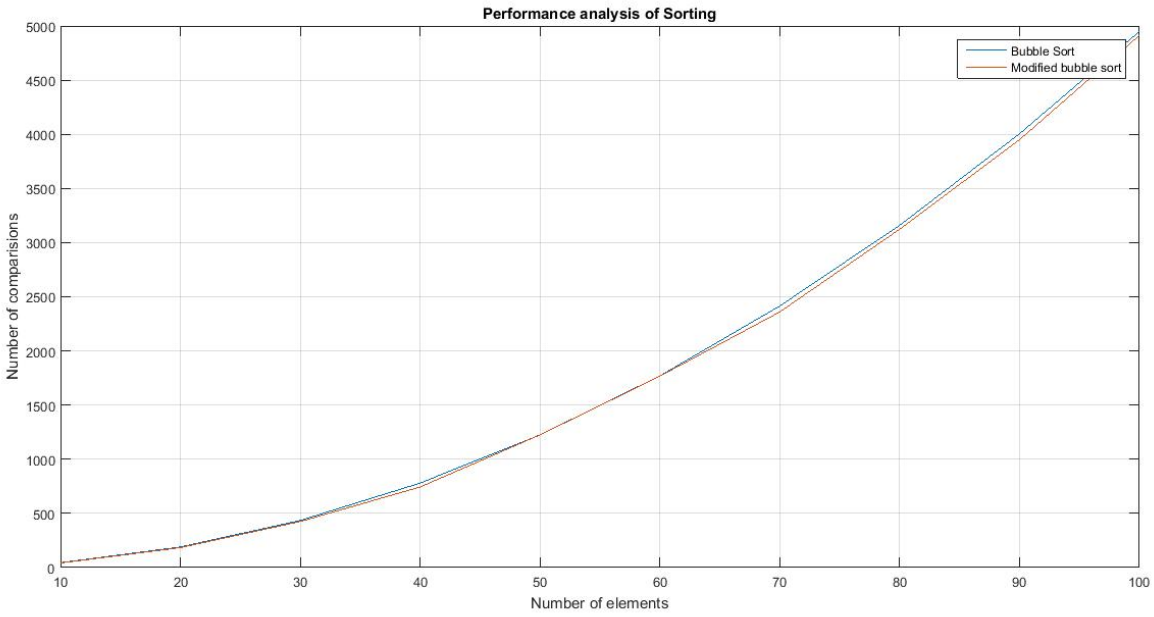
COMMENTS : When number of tosses increases the probability of getting a head tends to 0.5 i.e. Equal probability

Q.2. Performance analysis of Bubble Sort

Write the program to implement two different versions of bubble sort(BUBBLE SORT that terminates if the array is sorted before n-1th Pass. Vs. BUBBLE SORT that always completes the n-1th Pass) for randomized data sequence.

```
function [ count ] = m_bubbles( B,n )
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here
count = 0;
for i = 1:n
    flag = 0;
    for j = 1:n-i
        count = count+1;
        if B(j) > B(j+1)
            flag = 1;
            temp = B(j);
            B(j) = B(j+1);
            B(j+1) = temp;
        end
    end
    if flag == 0
      break;
    end
end

end

function [ count ] = bubbles( a,n )
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here
count = 0;
for i = 1:n
    for j = 1:n-i
        count = count+1;
        if a(j) > a(j+1)
            temp = a(j);
            a(j) = a(j+1);
            a(j+1) = temp;
        end
    end
end

end


clear all;
```

```
c = zeros(1,10);
ip = 1;

for n = 10:10:100
    A =  round(rand(1,n) * 100 );
    B = A;
    c(ip) = n;
    a(ip) = bubbles(A,n);
    b(ip) = m_bubbles(B,n);
    ip = ip + 1;
end
plot(c,a,c,b);
title('Performance analysis of Sorting');
xlabel('Number of elements');
ylabel('Number of comparisions');
disp('Analysis of bubble sort');
legend('Bubble Sort','Modified bubble sort');
grid on;
```



COMMENTS: It is observed that the bubble sort takes more time than modified bubble sort.

It is due to the fact that bubble sort always runs for the entire array even when the array is already sorted in the middle of the iterations.

But modified bubble sort stops the iteration process when in an entire iteration there is no swapping of elements.

Q.3. Average case analysis for Sorting Algorithms

For each of the data formats: random, reverse ordered, and nearly sorted, run your program say **SORTTEST** for all combinations of sorting algorithms and data sizes and complete each of the following tables. When you have completed the tables, analyze your data and determine the asymptotic behavior of 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*** . select the suitable no of elements for the analysis that supports your program.

```
function [ count ] = Insertion_sort( a,n )
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here
    count = 0;
    for j = 2:n
        k = a(j);
        i = j-1;
        while i>=1 && a(i) > k
            count = count + 1;
            a(i+1) = a(i);
            i = i - 1;
        end
        a(i+1) = k;
    end
end
```

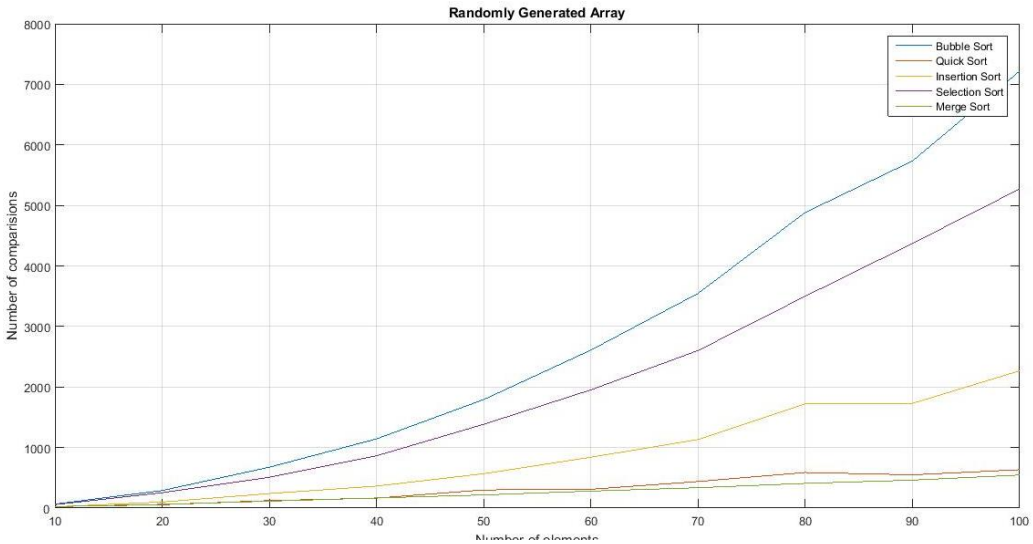
```
function [ count ] = Selection_sort( a , n )
%UNTITLED5 Summary of this function goes here
%   Detailed explanation goes here
     count = 0;
     for i = 1:n
        small = a(i);
        pos = i;
        for j = i+1:n
            count = count + 1;
            if a(j)<small
                count = count + 1;
                small = a(j);
                pos = j;
            end
            temp = a(i);
            a(i) = a(pos);
            a(pos) = temp;
        end
     end
end
```

```
function [ count ] = bubbles( a,n )
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here
count = 0;
for i = 1:n
    for j = 1:n-i
        count = count+1;
        if a(j) > a(j+1)
            temp = a(j);
            a(j) = a(j+1);
            a(j+1) = temp;
        end
    end
end

end

function [ count ] = quick( a,n )
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here
count = 0;
l = 1;
u = n;
[a,count] = quickSort(a,l,u,count);

end
function [a,count] = quickSort(a,l,u,count)
    p=0;
    if(l<u)
        [a,p,count] = part(a,l,u,count);
        [a,count] = quickSort(a,l,p-1,count);
        [a,count] = quickSort(a,p+1,u,count);
    end
end

function [a,p,k] = part(a,l,u,k)
    pivot = u;
    i = l;
    j = l;
    while i<u
        k = k+1;
        if a(i) <= a(pivot)
            temp = a(i);
            a(i) = a(j);
            a(j) = temp;
            j = j+1;
        end
        i = i+1;
    end
    temp = a(j);
    a(j) = a(i);
    a(i) = temp;
    p = j;
end
```

```
function [ count ] = mergeS( a,n )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

count = 0;
l = 1;
u = n;
[a,count]=mergeSort(a,l,u,count);

end

function [a,count] = mergeSort(a,l,u,count)
m = floor((l+u)/2);
if(l<u)
    [a,count] = mergeSort(a,l,m,count);
    [a,count] = mergeSort(a,m+1,u,count);
    [a,count] = pivot(a,l,m,u,count);
end
end

function [a,d] = pivot(a,l,m,u,d)
    s1 = m-l+1;
    s2 = u-m;
    b = zeros(1,s1);
    c = zeros(1,s2);
    for i = 1:s1
        b(i) = a(l+i-1);
    end
    for i = 1:s2
        c(i) = a(m+i);
    end
    i = 1; j = 1; k = l;
    while i <= s1 && j <= s2
        d = d+1;
        if b(i) <= c(j)
            a(k) = b(i);
            i = i+1;
        else
            a(k) = c(j);
            j = j+1;
        end
        k = k+1;
    end
    while i <= s1
        a(k) = b(i);
        i = i+1;
        k = k+1;
    end
    while j <= s2
        a(k) = c(j);
        j = j+1;
        k = k+1;
    end
end
```

//MAIN (Random Array)

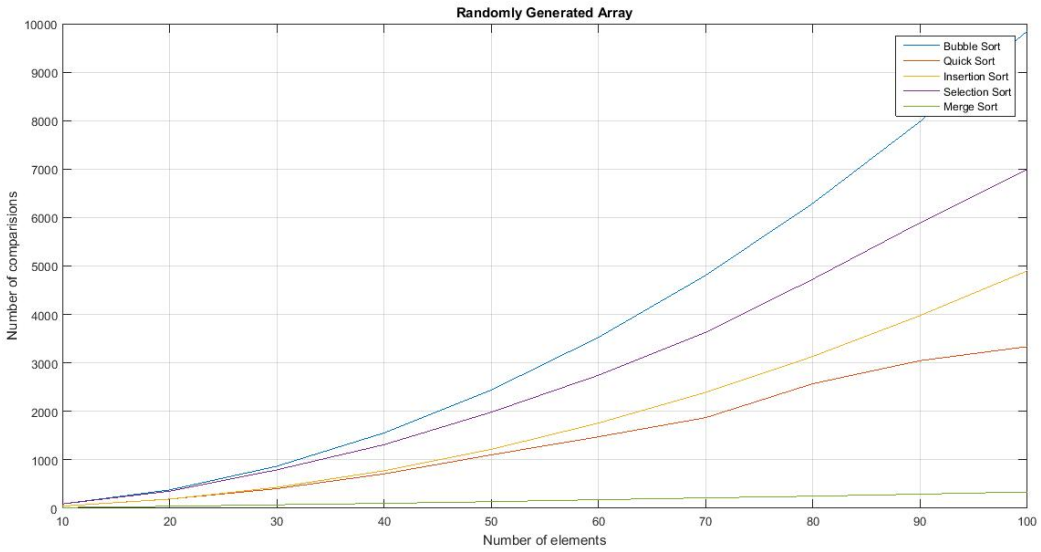
```
x = zeros(1,10);
a = zeros(1,10);
b = zeros(1,10);
c = zeros(1,10);
d = zeros(1,10);
e = zeros(1,10);
ip = 1;

for n = 10:10:100
    A =  round(rand(1,n) * 100 );
    B = A;
    C = A;
    D = A;
    E = A;
    x(ip) = n;
    a(ip) = bubbles(A,n);
    b(ip) = quick(B,n);
    c(ip) = Insertion_sort(C,n);
    d(ip) = Selection_sort(D,n);
    e(ip) = mergeS(E,n);
    ip = ip + 1;
end
plot(x,a,x,b,x,c,x,d,x,e);
title('Randomly Generated Array');
xlabel('Number of elements');
ylabel('Number of comparisions');
disp('Analysis of different sorting methods');
legend('Bubble Sort','Quick Sort','Insertion Sort','Selection Sort','Merge
Sort');
grid on;
```



COMMENSTS: This simulation shows sorting programs for random ordered arrays. Least comparisions are taken by Merge sort and highest by bubble sort.

//MAIN (Reverse Ordered)

```
x = zeros(1,10);
a = zeros(1,10);
b = zeros(1,10);
c = zeros(1,10);
d = zeros(1,10);
e = zeros(1,10);



ip = 1;

for n = 10:10:100
    arr =  round(rand(1,n) * 100 );
    A = sort(arr,'descend');
    B = A;
    C = A;
    D = A;
    E = A;
    x(ip) = n;
    a(ip) = bubbles(A,n);
    b(ip) = quick(B,n);
    c(ip) = Insertion_sort(C,n);
    d(ip) = Selection_sort(D,n);
    e(ip) = mergeS(E,n);
    ip = ip + 1;
end
plot(x,a,x,b,x,c,x,d,x,e);
title('Randomly Generated Array');
xlabel('Number of elements');
ylabel('Number of comparisions');
disp('Analysis of different sorting methods');
legend('Bubble Sort','Quick Sort','Insertion Sort','Selection Sort','Merge
Sort');
grid on;
```

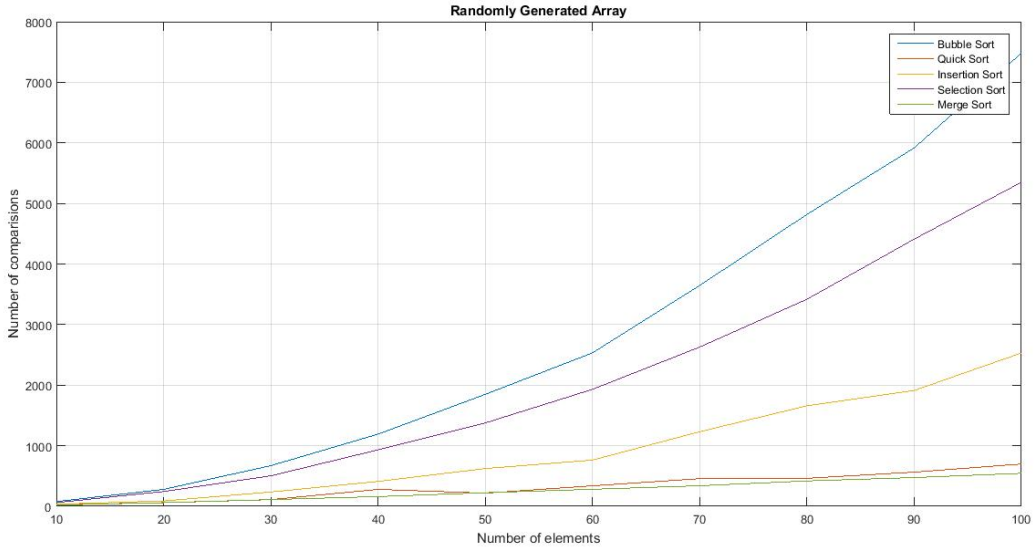


COMMENTS: This simulation takes into account a reverse ordered array and sorts it in % different techniques.
The merge sort being the best among them and bubbles the worst.

//MAIN (Almost Sorted)

```
x = zeros(1,10);
a = zeros(1,10);
b = zeros(1,10);
c = zeros(1,10);
d = zeros(1,10);
e = zeros(1,10);
ip = 1;

for n = 10:10:100
    A =  round(rand(1,n) * 100 );
    bubbles(A,n/2);
    B = A;
    C = A;
    D = A;
    E = A;
    x(ip) = n;
    a(ip) = bubbles(A,n);
    b(ip) = quick(B,n);
    c(ip) = Insertion_sort(C,n);
    d(ip) = Selection_sort(D,n);
    e(ip) = mergeS(E,n);
    ip = ip + 1;
end
plot(x,a,x,b,x,c,x,d,x,e);
title('Randomly Generated Array');
xlabel('Number of elements');
ylabel('Number of comparisions');
disp('Analysis of different sorting methods');
```

```
legend('Bubble Sort','Quick Sort','Insertion Sort','Selection Sort','Merge
Sort');
grid on;
```

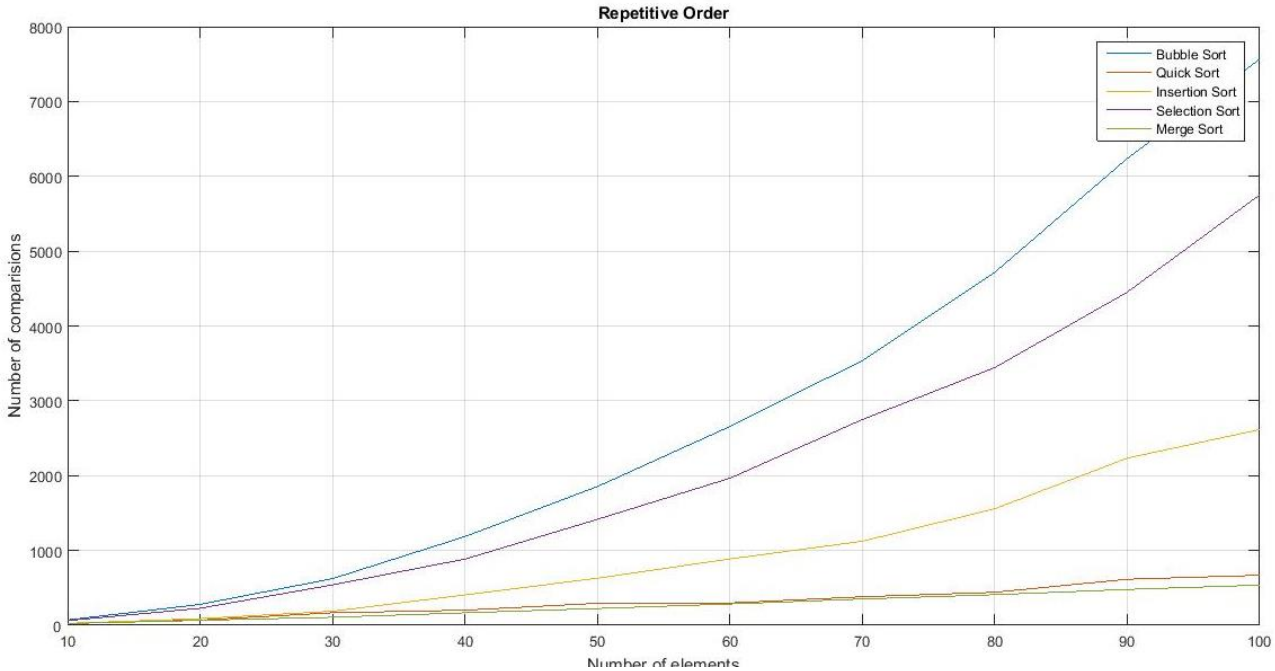


COMMENTS: This simulation takes into account an almost sorted array. First a random array is sorted to its half length using bubble sort the all the techniques are applied on the resulting array.

//MAIN(Repetitive Elements)

```
x = zeros(1,10);
a = zeros(1,10);
b = zeros(1,10);
c = zeros(1,10);
d = zeros(1,10);
e = zeros(1,10);
ip = 1;

for n = 10:10:100
    arr =  round(rand(1,n) * 100 );
    A = cat(2,arr,arr);
    A = cat(2,A,A);
    B = A;
    C = A;
    D = A;
    E = A;
    x(ip) = n;
    a(ip) = bubbles(A,n);
    b(ip) = quick(B,n);
    c(ip) = Insertion_sort(C,n);
    d(ip) = Selection_sort(D,n);
    e(ip) = mergeS(E,n);
    ip = ip + 1;
end
```

```
plot(x,a,x,b,x,c,x,d,x,e);
title('Repetitive Order');
xlabel('Number of elements');
ylabel('Number of comparisions');
disp('Analysis of different sorting methods');
legend('Bubble Sort','Quick Sort','Insertion Sort','Selection Sort','Merge
Sort');
grid on;
```



COMMENTS: This simulation takes in to account a highly repetitive array. Merge and Quick being almost taking similar time and bubble taking longest time.

Q.4. Variants of QUICK SORT

Compare the performance of **variants of quick sort** algorithm for instance characteristic n=10, ..., 1000. Use the finding from Q3, [cross-over point where insertion sort shows the better performance over quick sort] Modify your sorting algorithm in the previous problem to stop partitioning the list in QUICKSORT when the size of the (sub)list is less than or equal to 12 and sort the remaining sublist using INSERTIONSORT. Your counter will now have to count compares in both the partition function and in each iteration of INSERTIONSORT. Again, run the

experiment for 50 iterations and record the same set of statistics. Compare your results for the two different sorting techniques and comment upon your results.

//QUICK SORT WITH LAST PIVOT

```
function [ count ] = quick( a,n )
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here
count = 0;
l = 1;
u = n;
[a,count] = quickSort(a,l,u,count);

end
function [a,count] = quickSort(a,l,u,count)
    p=0;
    if(l<u)
        [a,p,count] = part(a,l,u,count);
        [a,count] = quickSort(a,l,p-1,count);
        [a,count] = quickSort(a,p+1,u,count);
    end
end

function [a,p,k] = part(a,l,u,k)
    pivot = u;
    i = l;
    j = l;
    while i<u
        k = k+1;
        if a(i) <= a(pivot)
            temp = a(i);
            a(i) = a(j);
            a(j) = temp;
            j = j+1;
        end
        i = i+1;
    end
    temp = a(j);
    a(j) = a(i);
    a(i) = temp;
    p = j;
end
```

//QUICK SORT WITH FIRST PIVOT

```
function [ count ] = first_quick( a,n )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

count = 0;
l = 1;
u = n;
[a,count] = s_quickSort(a,l,u,count);

end
```

```
function [a,count] = s_quickSort(a,l,u,count)
    p = 0;
    if l<u
        [a,p,count] = s_partition(a,l,u,count);
        [a,count] = s_quickSort(a,l,p-1,count);
        [a,count] = s_quickSort(a,p+1,u,count);
    end
end

function [a,p,count] = s_partition(a,l,u,count)
    pivot = l;
    i = u;
    j = u;
    while i>l
        count = count+1;
        if a(i) >= a(pivot)
            temp = a(i);
            a(i) = a(j);
            a(j) = temp;
            j = j-1;
        end
        i = i-1;
    end
    temp = a(j);
    a(j) = a(i);
    a(i) = temp;
    p = j;
end
```

//QUICK SORT WITH PIVOT AT RANDOM POINT

```
function [ count ] = rand_quick( a,n )
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here
count = 0;
l = 1;
u = n;
[a,count] = rand_quickSort(a,l,u,count);
end

function [a,count] = rand_quickSort(a,l,u,count)
    p = 0;
    if l<u
        [a,p,count] = rand_partition(a,l,u,count);
        [a,count] = rand_quickSort(a,l,p-1,count);
        [a,count] = rand_quickSort(a,p+1,u,count);
    end
end

function [a,p,count] = rand_partition(a,l,u,count)

    pivot = l+round(rand(1)*(u-l));
    i = l;
    j = l;
```

```
    while i <= u

        if i ~= pivot
            count = count+1;
        if a(i) <= a(pivot)
            temp = a(i);
            a(i) = a(j);
            a(j) = temp;
            if j == pivot
                pivot = i;
            end
            j = j+1;
        end
        end
        i = i+1;

    end
    temp = a(j);
    a(j) = a(pivot);
    a(pivot) = temp;
    p = j;
end
```

//FIRST QUICK THEN INSERTION

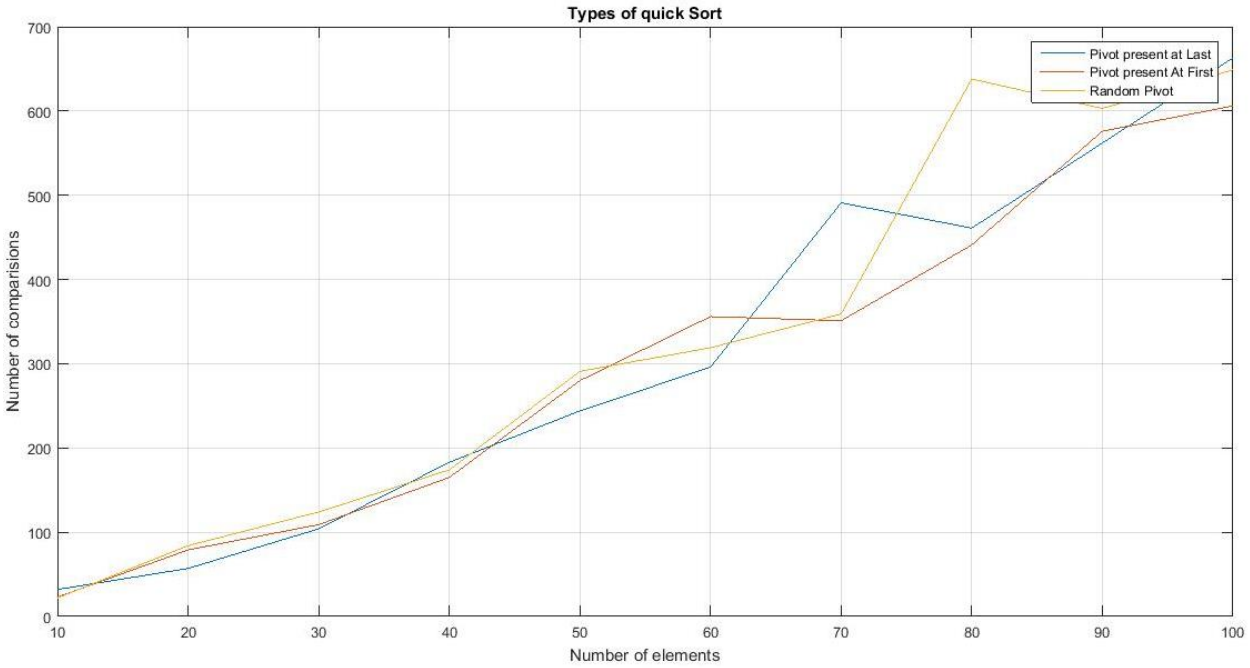
```
function [ count ] = m_quick( a,n )
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here

count = 0;
l = 1;
u = n;
[a,count] = quickSort(a,l,u,count);

end

function [a,count]=quickSort(a,l,u,count)
    p=0;
    if(l<u)
        if(u-l+1<=12)
            [a,count]=insertionSort(a,l,u,count);
        else
        [a,p,count]=partition(a,l,u,count);
        [a,count]=quickSort(a,l,p-1,count);
        [a,count]=quickSort(a,p+1,u,count);
        end
    end
end

function [a,p,count]=partition(a,l,u,count)
    pivot = u;
    i = l;
    j = l;
    while i<u
        count = count+1;
```

```
        if a(i) <= a(pivot)
            temp = a(i);
            a(i) = a(j);
            a(j) = temp;
            j = j+1;
        end
        i = i+1;
    end
    temp = a(j);
    a(j) = a(i);
    a(i) = temp;
    p = j;
end

function [a,count]=insertionSort(a,l,u,count)
    for i = l+1:u
    key = a(i);
    j=i-1;
    while j >= l && a(j)>key
        count = count+1;
        a(j+1) = a(j);
        j = j-1;
    end
    if j~=l-1
    count = count+1;
    end
    a(j+1) = key;
    end
end
```

//MAIN_I

```
x = zeros(1,10);
A = zeros(1,10);
B = zeros(1,10);

ip = 1;

for n = 10:10:100
    arr = round(rand(1,n)*100);
    P = arr;
    Q = arr;

    %fprintf('\nArray : %d',P);

    x(ip) = n;
    A(ip) = quick(arr,n);
    B(ip) = first_quick(P,n);
    C(ip) = rand_quick(Q,n);

    ip = ip + 1;
end
plot(x,A,x,B,x,C);
title('Types of quick Sort');
xlabel('Number of elements');
```

```
ylabel('Number of comparisions');
disp('Analysis of sortings');
legend('Pivot present at Last','Pivot present At First','Random Pivot');
grid on;
```



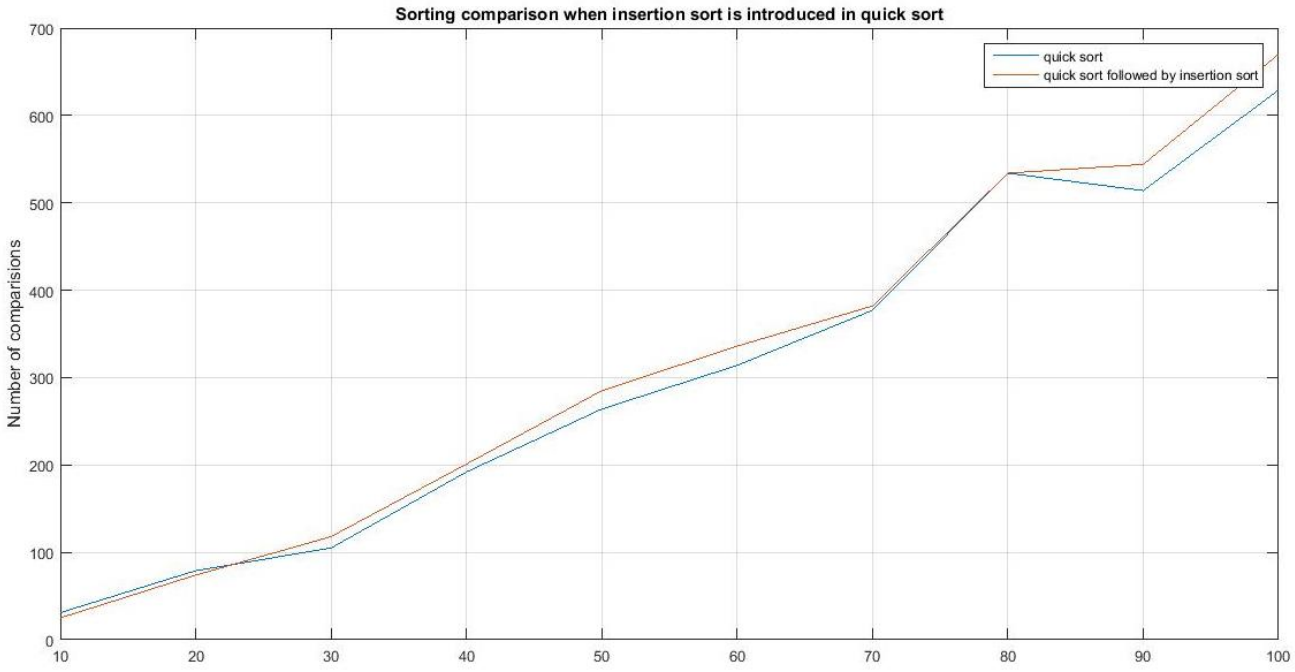
COMMENTS: From this simulation we see the pivot point doesn't affect the type of quick sort as the results are fairly the same. They roughly bring the same results in the sorting algorithm.

So basically we can take the pivot point in any position regardless of the array that we are using.

//MAIN_II

```
x = zeros(1,10);
A = zeros(1,10);
B = zeros(1,10);

ip = 1;

for n = 10:10:100
    arr = round(rand(1,n)*100);
    P = arr;

    %fprintf('\nArray : %d',P);

    x(ip) = n;
    A(ip) = quick(arr,n);
    B(ip) = m_quick(P,n);
```

```
    ip = ip + 1;
end
plot(x,A,x,B);
title('Sorting comparison when insertion sort is introduced in quick sort');
xlabel('Number of elements');
ylabel('Number of comparisions');
disp('Analysis of sortings');
legend('quick sort','quick sort followed by insertion sort');
grid on;
```



COMMENTS: From this simulation we observe that the quick sort with insertion sort takes more time than the quick sort and this can be proved from question 3 that insertion sort works better in case of lesser elements than the quick sort and quick sort works better in case of huge number of elements.