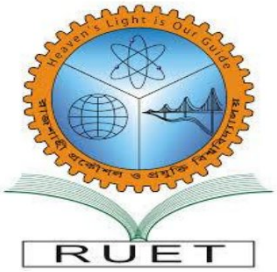
Heaven's Light is Our Guide



Rajshahi University of Engineering and Technology

Department of Computer Science and Engineering

Course No: CSE.2202

Course Title: Sessional based on CSE.2201 (Computer Algorithms)

Lab Report No: 03

Lab Report On: Comparison of Marge Sort and Quick Sort Algorithm.

Submitted By

Md. Ariful Islam

Roll No: 1803046

Section: A

Department: CSE

Submitted To

Dr. Md. Ali Hossain

Associate Professor

Dept. of CSE,RUET

Date: 06-03-2021

Title: Finding out the time and space complexity of the Marge Sort approach and compareing it with the performance of Quick Sort algorithm for (i) Best (ii) Average and (iii) Worst cases.

Introduction:

(I) Marge sort is an algorithm based on divide and conquer strategy. In this, the elements are split into two sub-array ($n/2$) again and again until only one element is left. It uses additional storage for sorting auxiliary array. In this ~~element~~ three arrays were used, ~~two~~ of them are for sorting each half and third one for storing the the two half and marge these.
Finally the all sub array are marged in 'n' element size of the array.

The best, average and worst cases ~~is~~ complexity is $O(n * \log n)$ and additional 'n' space is required

(II) Quick sort is an algorithm based on divide and conquer strategy. In this the array of elements is divided into parts repeatedly until it is not possible to divide further. It uses a a key element on pivot for partitioning the elements. One left partition contains all the elements that are smaller than pivot and one right partition contains all the elements that are greater than pivot.

The best and average case complexity is $O(n * \log n)$ and the worst case complexity is $O(n^2)$.

Algorithm:

(I) For Marge Sort:

```
(i) MargeSort (low, high) {
        if (low < hight){
            mid := ⌊ (low+hight)/2 ⌋ ;
            MargeSort (low, mid);
            MargeSort (mid+1 , high);
            Marge (low, mid, high);
        }
}
```

(ii) Merge (low, mid, high) {

```
h := low ;  i := low ;  j := high ;
while ( (h ≤ mid) and (j ≤ high) ) do {
    if ( a[h] ≤ a[j] ) then {
        b[i] := a[h] ; h := h+1 ;
    }
    else {
        b[i] := a[j] ; j := j+1;
    }
    i := i+1 ;
}
if (h > mid) then
    for k := j to high do {
        b[i] := a[k] ;  i := i+1 ;
    }
else
    for k := h to mid do {
        b[i] := a[k] ;  i := i+1 ;
    }
for k := low to high do a[k] := b[k] ;
}
```

(II) For Quick Sort :

(i) QuickSort (left , right) {

```
if ( left < right ) {

    j := Partition ( left , right+1 );
    QuickSort ( left , j-1 );
    QuickSort ( j+1 , right );

}
```

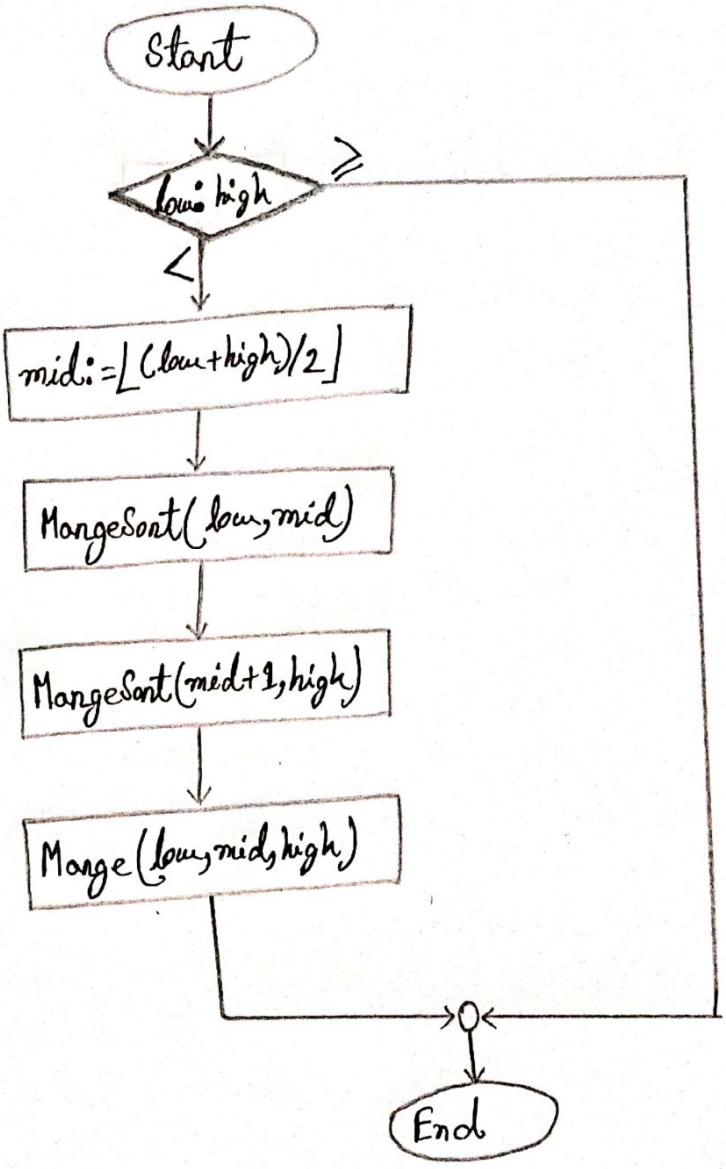
(ii) Partition (m , p) {

```
v := a[m] ; i := m ; j := p ;
repeat {
    repeat {
        i := i+1 ; }
    until ( a[i] ≥ v );
    repeat {
        j := j-1 ; }
    until ( a[j] ≤ v );
    if ( i < j ) then  t := a[i] ; a[i] := a[j] ; a[j] := t ;
} until ( i ≥ j )
a[m] := a[j] ;  a[j] := v ;
return j ;
}
```

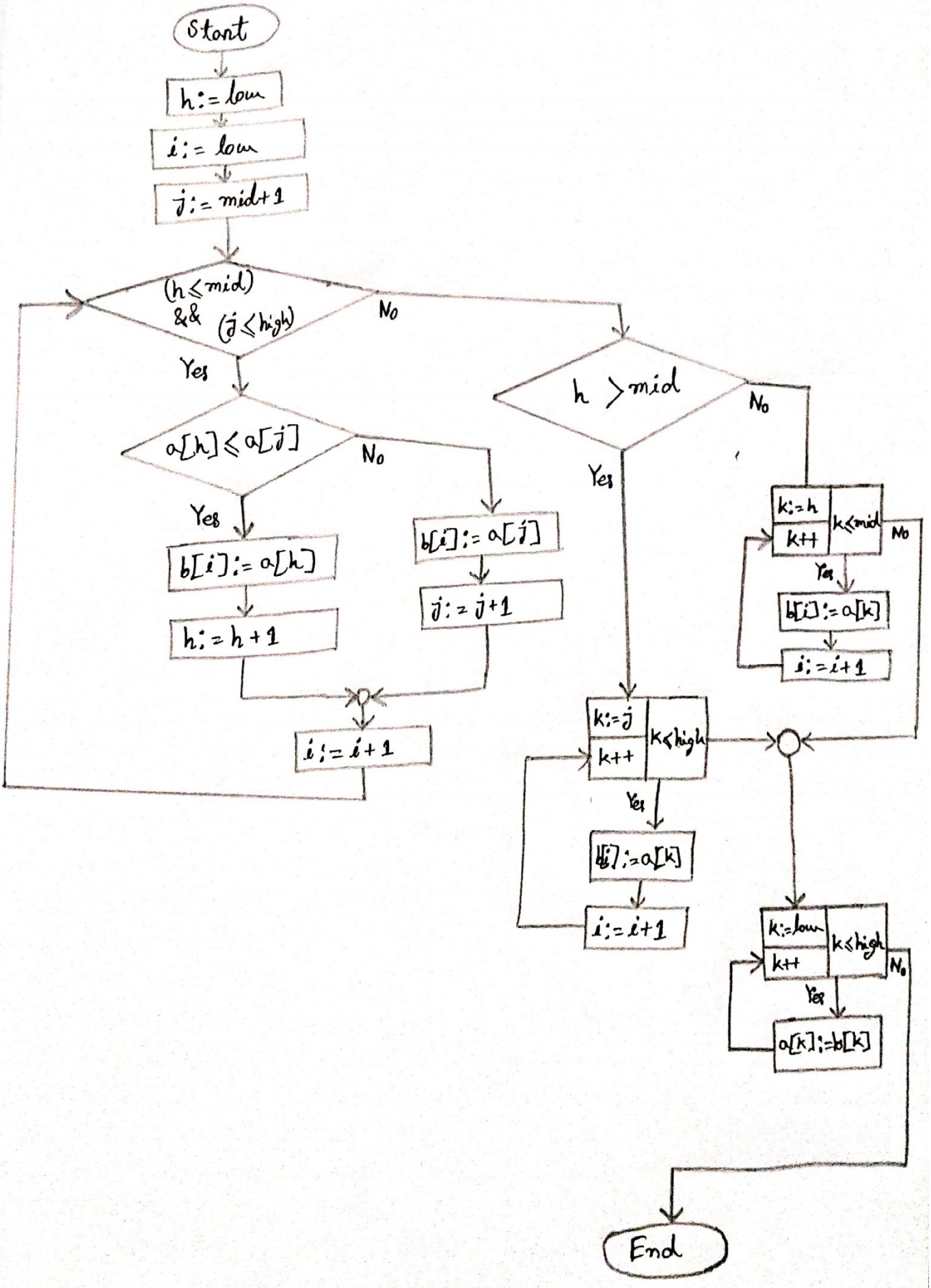
Flow Chart :

① For Mange Sort :

ⓘ Mange Sort

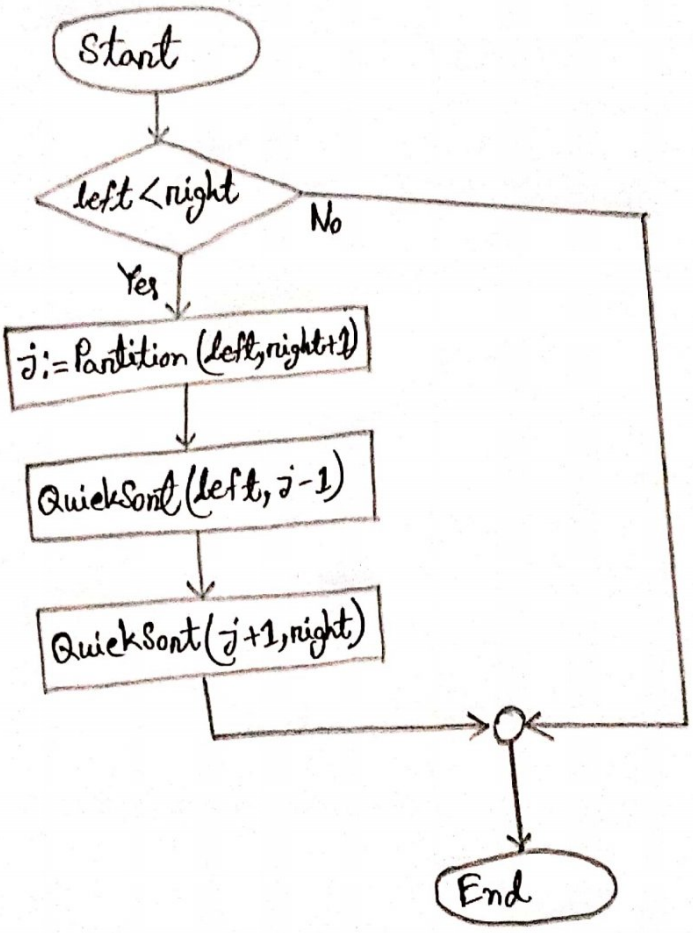


(ii) Merge

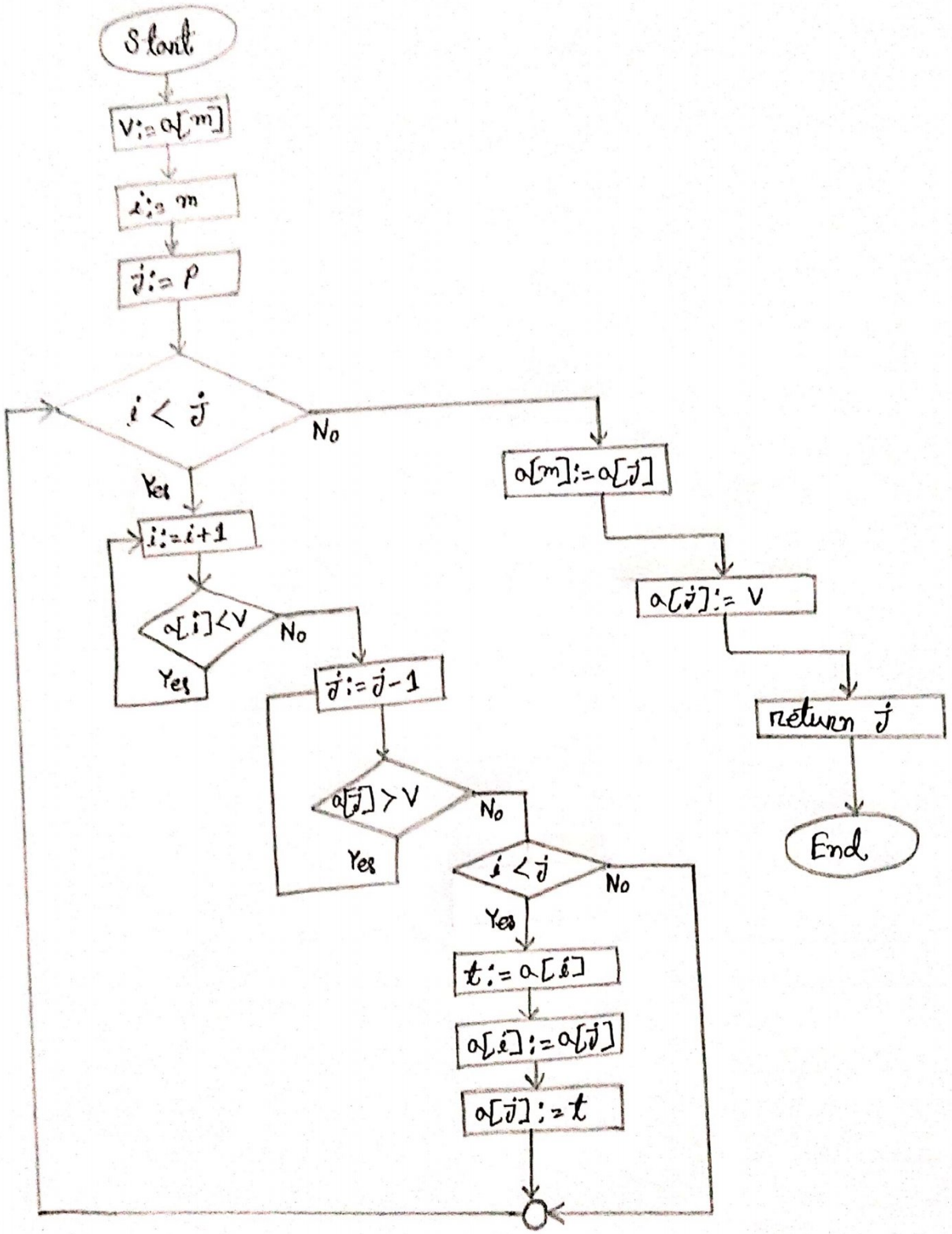

Start
h := low
i := low
j := mid+1
(h ≤ mid) && (j ≤ high)
No
Yes
a[h] ≤ a[j]
No
Yes
b[i] := a[h]
h := h + 1
b[i] := a[j]
j := j+1
i := i+1
h > mid
No
Yes
k := h
k++
k ≤ mid
No
Yes
b[i] := a[k]
i := i+1
k := j
k++
k ≤ high
Yes
b[i] := a[k]
i := i+1
k := low
k++
k ≤ high
No
Yes
a[k] := b[k]
End

(11) For Quick Sort:

(i) QuickSort



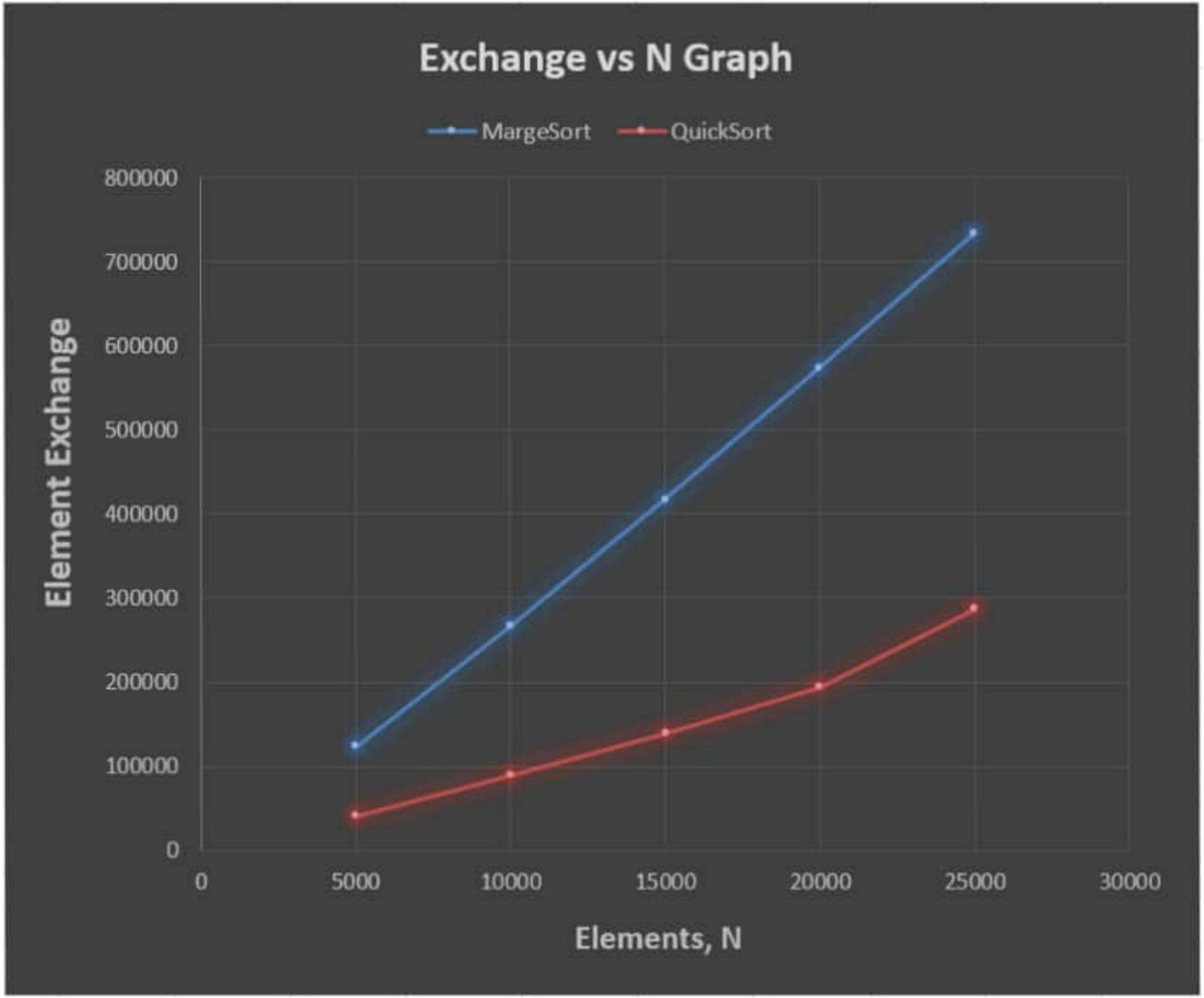
(ii) Partition :



Output :

n	Merge Sort		Quick Sort	
5000	Exchange	123616	Exchange	40877
	Time	2993 μs	Time	995 μs
	Space (ll)	10000 bytes	Space (ll)	5000 bytes
10000	Exchange	267232	Exchange	89425
	Time	4992 μs	Time	2026 μs
	Space (ll)	20000 bytes	Space (ll)	10000 bytes
15000	Exchange	417232	Exchange	139649
	Time	5552 μs	Time	2991 μs
	Space (ll)	30000 bytes	Space (ll)	15000 bytes
20000	Exchange	574464	Exchange	194030
	Time	7000 μs	Time	3989 μs
	Space (ll)	40000 bytes	Space (ll)	20000 bytes
25000	Exchange	734464	Exchange	287842
	Time	8041 μs	Time	5068 μs
	Space (ll)	50000 bytes	Space (ll)	25000 bytes

Graph :


Exchange vs N Graph
MargeSort
QuickSort
800000
700000
600000
500000
400000
300000
200000
100000
0
Element Exchange
0
5000
10000
15000
20000
25000
30000
Elements, N

Conclusion: From the sample output and graph we can see that the number of steps in merge sort is required more than the number of steps in quick sort.

And the required time is greater in merge sort than quick sort.

We have used array to store data. The merge sort may have worked faster in linked list.