Question#1

Write a C program which can input some random data into an array and sort the array in ascending order using mergesort.

Question#2

Write a C program which can input some random data into an array and sort the array in ascending order using quicksort. [You may use any of the following two partition techniques: Hoare's partition or Lomuto's partition]

Additional Task

Additional Task#1

Write a program which can input huge numbers of random data (at least 50,000) into an array and sort the array in ascending order using both mergesort and quicksort. Show the time required for each algorithm.

Additional Task#2

Write a program which can show time required for mergesort and quicksort to sort a large array (at least 30,000) in reverse order i.e. sort an ascending sorted array into descending order.

Additional Task#3

Write a program which can implement quicksort using both Hoare's partition and Lomuto's partition. Display number of interchange operation executed in those two different partition techniques]

Appendix:

You may read this article:

https://www.geeksforgeeks.org/hoares-vs-lomuto-partition-scheme-quicksort/

```
QUICKSORT(A, p, r)
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, r)
```

Lomuto's partition

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

Hoare's partition

```
HOARE-PARTITION(A, p, r)
 1  x = A[p]
 2  i = p − 1
 3  j = r + 1
 4  while TRUE
 5      repeat
 6          j = j − 1
 7      until A[j] ≤ x
 8      repeat
 9          i = i + 1
10      until A[i] ≥ x
11      if i < j
12          exchange A[i] with A[j]
13      else return j
```

```
MERGE-SORT(A, p, r)
1.  if p < r
2.      q = ⌊(p + r)/2⌋
3.      MERGE-SORT(A, p, q)
4.      MERGE-SORT(A, q+1, r)
5.      MERGE(A, p, q, r)
```

```
MERGE(A, p, q, r)
1.   n_1 = q − p + 1
2.   n_2 = r − q
3.   let L[1..n_1 + 1] and R[1..n_2 + 1] be new arrays
4.   for i = 1 to n_1
5.       L[i] = A[p + i − 1]
6.   for j = 1 to n_2
7.       R[j] = A[q + j]
8.   L[n_1 + 1] = ∞
9.   R[n_2 + 1] = ∞
10.  i = 1
11.  j = 1
12.  for k = p to r
13.      if L[i] ≤ R[j]
14.          A[k] = L[i]
15.          i = i + 1
16.      else A[k] = R[j]
17.          j = j + 1
```