1. Sort arrays A, B and C in none decreasing order using below pseudo codes (Exchange Sort, Merge Sort and Quick Sort) then compare these algorithm in the sense of Time and Memory complexity.

2. Explain the best, average and worst case for these algorithms.

3. Try to code them in your favorite programming language and compare number of execution of main operation by inserting an operation counter to proof your analyses in time complexity.

A: 12 18 25 32 55 60

B: 18 55 32 60 12 25

C: 60 55 32 25 18 12

```
void  exchangesort (int n , keytype S[ ])
{
    index i,j;
    for (i = 1 ; i<= n -1; i++)
        for (j = i +1; j <= n ; j++)
            if ( S[j] < S[i])
               exchange S[i]  and S[j];
}
```

```
void  mergsort (int n , keytype S [ ])
   {
       const int  h = ⌈n/2⌉ , m = n – h;
       keytype U [1...h],V [1..m];
       if (n >1)  {
          copy S[1] through  S[h] to U[h];
          copy S [h + 1] through S[n] to V[1] through V[m];
          mergesort(h, U);
          mergesort(m,V);
          merge (h , m , U,V,S);
         }
    }

void  merg ( int h , int m, const keytype U[ ],
                            const keytype V[ ],
                                  keytype  S[ ] )
  {
      index i , j , k;
      i = 1; j = 1 ; k = 1;
      while (i <= h && j <= m) {
```

```
        if (U [i] < V [j]) {
            S [k] = U [i]
             i+ + ;
   }
   else {
      S [k] = V [j];
       j+ +;
    }
    k+ +;
}
if ( i > h)
    copy V [j] through V [m] to S [k]  through  S [ h + m ]
else
    copy U [i] through U [h] to S [k]  through  S [ h + m ]
}
```

```
void quicksort  (index low , index high)
 {
      index  pivotpoint;
      if   ( high >  low) {
              partition (low , high , pivotpoint)
              quicksort (low , pivotpoint – 1)
              quicksort (pivotpoint + 1 , high);
       }
  }

void partition (index low, index high)
                  index & pivotpoint)
  {
      index i , j;
      keytype pivotitem;
      pivotitem = S [low];
      j = low
      for ( i = low +1 ; i <= high; i ++)
          if ( S [i] < pivotitem )  {
           j++;
           exchange S [i]  and S [j];
      }
   pivotpoint  = j;
   exchange S [low] and S [ pivotpoint];
}
```
