Name :- SHISHU
REG :- 2020CA089

Assignment 2

Ques1 Recursive Procedure of insertion sort.

```
void insertionsort ( arr [] , n ) {
    if  n <= 1
            return
        insertionsort (arr, n-1)
        last  =  arr [n-1]
          J   = n-2;
      while ( J >0 and arr[J] > last )
            arr [J+1]  =  arr[J]
               J --
        arr[J+1] = last
}
```

$$\boxed{T[n] = T(n-1) + n}$$ Ans

Ques 2 :-

Algo :-

```
Sumsearch (arr[], length, sum) {
    mergesort (arr, 1, length)
      for (i=1 to length)
          index = Binarysearch (arr, x- arr[i])
          if(intex ≠-1) and index ≠ i
                return true
      return false
```

Ques 3 :-

(a) Sorting sublists :- for input of size K insertion sort run on $O(K^2)$ worst case time so worst case time to sort n/K sublists each of length K will be n/K $O(K^2)$ - $\Theta(nK)$

NAME :- SHISHU REG :- 2020CA089

(b) merging sublists:-

We have n elements devided into n/k sorted sublists each of length k. To mearge these n/k sorted sublists of length n. we have to take two sublists at a time and continus to merge them.

This will result in $lg(n/k)$ steps. And in every step we are eventially going to compare n elements. So the whole Process will run a $\theta(n \log(n/k))$.

(c) Largest value of k:

For the modified algorithim to have the same asymptotic running time as standerd merge sort $O(nk + nlg(n/k))$ must be same as $O(n\log n)$.

To satisfy the condition k can't grow faster then logn Asymtotically if it does then because of the nk from the algorithm will run at worst asymtotic time then $\theta(n\log n)$.

So, lets assume, $k = \theta(\log n)$

$\theta(nk + n\log(n/k)) = \theta(nk + nlgn - n\log k)$.

$= \theta(n\log n + n\log n - n\log \log n))$

$= \theta(n \log n)$

neglect loglogn very small.

(d) Practical value of k:-

To determine Practical value of k. It has to be largest input size for which insertion sort runs faster then merge sort to get exact value we need to calculate the exact running time expression with constant factor and use the method described In.

Quest4 ⓐ Given Array

	2	3	8	6	1
Index	1	2	3	4	5

5 inversion :-> (1,5), (2,5), (3,4), (3,5) and (4,5) { used index have

ⓑ Given Array :- 1, 2, 3 ...... n

Array will be most no. of inversion mao array sorted in descending order

no of int

no. of inversion $= \frac{n(n+1)}{2}$ $\left\{ \begin{array}{ll} 1^{st} \rightarrow & n-1 \\ 2^{nd} \rightarrow & n-2 \\ \vdots & \\ n^{m}-1 - & 1 \end{array} \right.$

ⓒ If no. of Inverstion in an array is more than inner loop of insertion sort run more time run more time So the higher the no. of inversions in an array, the longer insertion sort will take to sort the array.

ⓓ Algorithms to count inversion

```
count_Inversions (A, P, r)
  {
      if (P ≥ r)
          return 0
      q = (P+r) / 2
      left = cont_Inversions (A, P, q)
      right = cont_Inversions (A, q+1, r)
      total = left + right + merge (A, P, q, r)
      return total;
  }

merge (A, P, q ; r) {
      n1 = q-P+1
      n2 = r-q
   int L1 [1 --- n1] & L2 [1 ---- n2]
   for (i = 1 to n1)
      l[i] = A[P+i-1]
      for j = 1 to n2
      R[j] = A[q+j]
    L1 [n1+1] = ∞
     L2 [n2+1] = ∞
     i = 1
     j = 1
```

NAME: SHISHU
REG : 2020CA089

```
count = 0
for(k = p   to   r) {
    if (L[i] ≤ R[j]) {
        A[K] = L[j]
            i = i+1
    }
    else {
        count = count + ($n_1$ - i+2)
        A[K] = R[j]
            j = j+1
    }
}
return count;
}
```

Quen 5

(1) Insertion sort, merge sort, are stable and Heapsort, and quicksort are not.

(2) To make my Sorting algorithm stable we can preprocess, replacing each element of an array with an ordered pair. Pre first entry will be the value of the element and second value will be the index of the element.

eg. Array [2|1|1|3|4|4|4] would become [(2,1)|(1,2)|(1,3)|(3,4)|(4,5)|(4,6)|(4,7)]

2020 CA089

SHISHU

NAME: SHISHU
REG : 2020CA089

We now interpret $(i,j) < (k,m)$
if $(i<k)$ or
$i = k$ and
$j < m$.

This dobles the space requirement bot the running time will be asymptotically unchanged.

Ques 6

Step1: Run through the list of integers and convert each one to base n.

Step 2: Radix Sort
∵ Each number will have at most $\log_n n^3 = 3$ digit. so will be only 3 passes.

Step 3: For each pass, there are n possible values which can be taken on, so we can use counting sort to sort then in $O(n)$.