Ex: 3

Date: 27 / 1 / 25

Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam - 603 110 (An Autonomous Institution, Affiliated to Anna University, Chennai)

#### ICS1412: ALGORITHMS LABORATORY

#### Assignment 3 - Divide and Conquer

1. Given a list L of n numbers, an inversion is defined as a pair (L[i], L[j]) such that i < j and L[i] > L[j]. For example, if L = [3, 2, 8, 1], then (3, 2), (8, 1), (2, 1), (3, 1) are the inversions in L. Modify the algorithm of Mergesort to count inversions in a given list. Analyze the time complexity of the code.

### Algorithm:

1. Count Toversions using modified merge sort	
- hundren same merge count arr, the pain mid high	)
invalize i, j, p +0	
instalize invocant to	
co Act inche intertos des annota	
lest = an [low:mid+1]	
right = an [midtl: high+1]	
while is mid and is right do	
Chine 12 min & on []	
see the an[r] ← lest [i]	
i ← i+1	
else = 7	
an Ex] < right[j]	
Haar	
inv count + inv count + (mid-la	0
and the second second	
j-41	
D 4 RH	
return invocant	
( ary low high)	
Function merge sort (orr, low, high) Introduze introduce introduce introduce	
it sous region / 2	
mid = (low thigh)/2	

-	inv count ← inv count + merge 30rt (m, b,
	inv count & inv count + merge sort (or
	Alas, midtly to
	inx count + inx count + merge count (arr,
	low, mid, high
	return inv count



Ex: 3

Date: 27 / 1 / 25

#### Time Complexity:

#### Code:

```
def merge count(A, low, mid, high):
    i=0
    j=0
    k=low
    inv count=0
    left=A[low:mid+1]
    right=A[mid+1:high+1]
    while i<len(left) and j<len(right):</pre>
         if left[i] <= right[j]:</pre>
             A[k]=left[i]
             i+=1
         else:
             A[k]=right[j]
             inv count+= (mid - low + 1 - i)
             j += 1
         k+=1
    while (i<len(left)):
         A[k]=left[i]
         i+=1
         k+=1
    while (j<len(right)):</pre>
         A[k]=right[j]
         j+=1
```



NAME: A.S.Tritthik Thilagar Reg no: 3122237001057 Ex: 3 Date: 27 / 1 / 25 k+=1return inv\_count def merge sort count (A, low, high): inv count= $\overline{0}$ if(low < high):</pre> mid=(low + high)//2inv count+=merge sort count(A,low,mid) inv count+=merge sort count(A, mid+1, high) inv\_count+=merge count(A,low,mid,high) return inv count 1 = [3, 2, 8, 1]print("NUMBER OF INVERSIONS: ", merge sort count(1,0,len(1)-1))

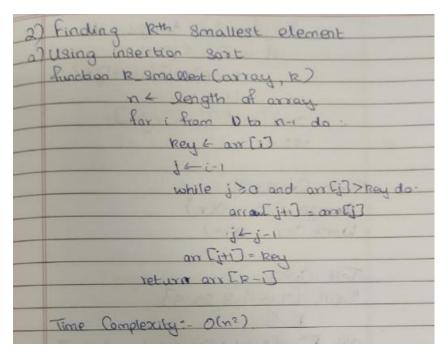
### Sample Testcase:

NUMBER OF INVERSIONS: 4

2. Find the  $k^{th}$  smallest element in an unsorted list using insertion sort. Then, find the same by modifying the divide-and-conquer algorithm of Quicksort. Compare the time complexities of both the algorithms.

#### Algorithm:

### I) Insertion Sort

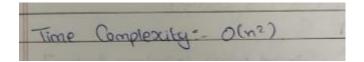




Ex: 3

Date: 27 / 1 / 25

#### Time Complexity:



#### Code:

```
def kth_smallest_insertion_sort(arr, k):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr[k - 1]
arr = [7, 10, 4, 3, 20, 15]
k = 3
print("k-th smallest element using Insertion Sort:", kth_smallest_insertion_sort(arr[:], k))
```

### Output:

k-th smallest element using Insertion Sort: 7

II) Quick Sort

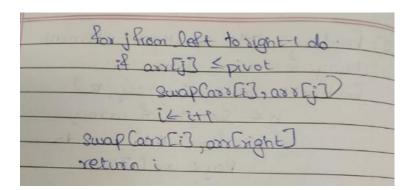
Algorithm:



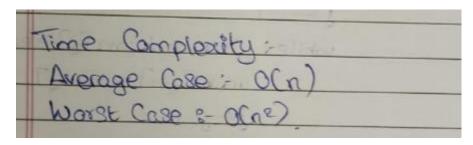
Ex: 3

Date: 27 / 1 / 25

function quickSort Earr, left, right, E)
if left signit:
pivot < parlation (arr, let, right)
if pivot=R:
return arx [pivat]
else if pivot >R
reture quicksort (arr, left, pinot-),
else: R)
roturn quick sort (arr, pivotal, right),
k)
function partition (arx, left, right)
Cariginal or a sovia
it Jeft



# Time Complexity:



#### Code:

```
def partition(arr, left, right):
    pivot = arr[right]
    i = left
    for j in range(left, right):
        if arr[j] <= pivot:
            arr[i], arr[j] = arr[j], arr[i]</pre>
```



```
Reg no: 3122237001057
Ex: 3
Date: 27 / 1 / 25
            i += 1
    arr[i], arr[right] = arr[right], arr[i]
    return i
def quicksort(arr, left, right, k):
    if left <= right:</pre>
        pivot index = partition(arr, left, right)
        if pivot index == k:
            return arr[pivot index]
        elif pivot index < k:</pre>
            return quicksort(arr, pivot index + 1, right, k)
        else:
            return quicksort(arr, left, pivot index - 1, k)
def kth smallest quicksort(arr, k):
    return quicksort(arr[:], 0, len(arr) - 1, k - 1)
arr = [7, 10, 4, 3, 20, 15]
k = 3
print("k-th smallest element using QuickSort:",
kth smallest quicksort(arr, k))
```

#### Sample Testcase:

NAME: A.S.Tritthik Thilagar

k-th smallest element using QuickSort: 7

- 3. Given a list A of size n, find the sum of elements in a subset A' of A such that the elements of A' are contiguous and has the largest sum among all such subsets. Please note that:
  - the subset should be having elements that are contiguous in the original list.
  - the input list may have negative values.
  - the algorithm should be based on divide and conquer strategy.

## Algorithm:

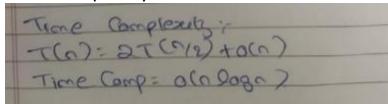


Ex: 3

Date: 27 / 1 / 25

	0
3.	Maximum Subarray Sum
	Sunction max crossing sum (arr, left, mid, right)
	Jest sum <-00
	Sum ← 0
-	& Occe
	for i from mid to left, step=-1, do:
	Sum + Sum tons Ci3
	if sum> loft sum
	left 3cm = Sum
	right Sum 60
	3um 60
	for i from midt to right +1 do:
	Sum 4 Sum + arr []
	if sum Sright sum
	right sun = Sum
	vetura lat sum tright sum
	function max subarray sum (arr, left, right)
	il left right
	return conflets
	mid = (left tright)/2
	left max & max subarray sum (ar, left)
	mid)
	right more & mare Subarray Sur (cos, mode)
	cossing 6 mase clossing sum (arr, left it, and
	return max Cleft max, right max, ight
	C(055105@3m

# Time Complexity:



#### Code:

```
def max_crossing_sum(arr, left, mid, right):
    left_sum = float('-inf')
    total = 0
    for i in range(mid, left - 1, -1):
        total += arr[i]
        left sum = max(left sum, total)
```



```
NAME: A.S.Tritthik Thilagar
Reg no: 3122237001057
Ex: 3
Date: 27 / 1 / 25
    right sum = float('-inf')
    total = 0
    for i in range(mid + 1, right + 1):
        total += arr[i]
        right_sum = max(right sum, total)
    return max(left sum + right sum, left sum, right sum)
def max subarray sum(arr, left, right):
    if left == right:
        return arr[left]
    mid = (left + right) // 2
    left_max = max_subarray_sum(arr, left, mid)
    right max = max subarray sum(arr, mid + 1, right)
    crossing max = max crossing sum(arr, left, mid, right)
    return max(left_max, right_max, crossing max)
arr = [-2, 1, -3, 4, -1, 2, 1, -5, 4]
print ("Maximum contiguous subarray sum:",
max subarray sum(arr, 0, len(arr) - 1))
```

### Sample Testcase:

Maximum contiguous subarray sum: 6

# Learning Outcome:

We Learnt how to use different Divide and Conquer Algorithms.

