NAME: A.S.Tritthik Thilagar
Reg no: 3122237001057
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 Inversions using modified merge sort

```
-> function merge_count(arr, low, mid, high)
      initialize i, j, k ← 0
      initialize inv_count ← 0
      left = arr [low : mid+1]
      right = arr [mid+1 : high+1]
      while i ≤ mid and j ≤ right do:
          & arr[i] ≤ arr[j]
              arr[k] ← left[i]
              i ← i+1
          else
              arr[k] ← right[j]
              inv_count ← inv_count + (mid-low
                                      +1-i)
              j ← j+1
          k ← k+1
      return inv_count
function merge_sort (arr, low, high)
                     initialize inv_count=0
      if low < high:
         mid = (low + high) / 2
             inv count ← inv count + merge_sort (arr, low,
                                                  mid)
             inv count ← inv count + merge_sort (arr,
                                                  mid+1, high)
             inv count ← inv count + merge_count (arr,
                                                  low, mid, high)
      return inv count
```

NAME: A.S.Tritthik Thilagar
Reg no: 3122237001057
Ex: 3
Date: 27 / 1 / 25

Time Complexity:

Time Complexity:-

$$T(n) = 2T(n/2) + \Theta(n)$$

$$= 2(2T(n/4) + \Theta(n/2)) + \Theta(n)$$

$$\cancel{T(n)\ 2k} = 4T(n/4) + 2cn \quad (c \to const)$$

$$= 8T(n/8) + 3cn$$

$$\Rightarrow T(n) = 2^k T(n/2^k) + kcn$$

cont till $n/2^k = 1 \Rightarrow k = \log n$

$$\Rightarrow T(n) = 2^{\log_2 n} (T(1)) + cn \cdot \log n$$

$$\Rightarrow T(n) = n + c(n \log n)$$

$$\therefore \boxed{T(n) = O(n \log n)}$$

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):
        if left[i]<=right[j]:
            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)):
        A[k]=right[j]
        j+=1
```

NAME: A.S.Tritthik Thilagar
Reg no: 3122237001057
Ex: 3
Date: 27 / 1 / 25

```
        k+=1
    return inv_count
def merge_sort_count(A,low,high):
    inv_count=0
    if(low < high):
        mid=(low + high)//2
        inv_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
l=[3,2,8,1]
print("NUMBER OF INVERSIONS: ",merge_sort_count(l,0,len(l)-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

2) Finding k^{th} smallest element
a) Using insertion sort

```
function k_smallest(array, k)
        n ← length of array
        for i from 0 to n-1 do:
            key ← arr[i]
            j ← i-1
            while j≥0 and arr[j]>key do:
                array[j+1] = arr[j]
                j ← j-1
            arr[j+1] = key
      return arr[k-1]
```

Time Complexity:- $O(n^2)$

NAME: A.S.Tritthik Thilagar
Reg no: 3122237001057
Ex: 3
Date: 27 / 1 / 25

Time Complexity:

Time Complexity :- $O(n^2)$

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:

NAME: A.S.Tritthik Thilagar
Reg no: 3122237001057
Ex: 3
Date: 27 / 1 / 25

```
Function quickSort (arr, left, right, k)
    if left ≤ right :
        pivot ← partition (arr, left, right)
    if pivot = k :
        return arr[pivot]
    else if pivot > k
        return quickSort (arr, left, pivot-1, k)
    else :
        return quickSort (arr, pivot+1, right, k)

Function partition (arr, left, right)
    pivot ← arr[right]
    i ← left
```

```
    for j from left to right-1 do
        if arr[j] ≤ pivot
            swap(arr[i], arr[j])
            i ← i+1
    swap (arr[i], arr[right])
    return i
```

Time Complexity:

Time Complexity :-
Average Case :- $O(n)$
Worst Case :- $O(n^2)$

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]
```

NAME: A.S.Tritthik Thilagar
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:
        pivot_index = partition(arr, left, right)
        if pivot_index == k:
            return arr[pivot_index]
        elif pivot_index < k:
            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:
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:

NAME: A.S.Tritthik Thilagar
Reg no: 3122237001057
Ex: 3
Date: 27 / 1 / 25

3. Maximum Subarray Sum

```
function max crossing sum(arr, left, mid, right)
    left sum ← -∞
    sum ← 0
    for i from mid to left, step=-1, do:
        Sum ← sum + arr[i]
        if sum > left sum
            left sum = sum
    right sum ← -∞
    sum ← 0
    for i from mid+1 to right+1 do:
        sum ← sum + arr[i]
        if sum > right sum
            right sum = sum
    return left sum + right sum
function 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 crossing sum(arr, left, mid, right)
    return max(left max, right max, crossing sum)
```

Time Complexity:

Time Complexity:
$T(n) = 2T(n/2) + O(n)$
Time Comp = $O(n \log n)$

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.