**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Computer Engineering Department**

Program: B.Tech. Sem V

**Course: Design and Analysis of Algorithms**

**List of Experiments**

w.e.f. 1th Jul 2020

**Faculty:** Abhay Kolhe.

LAB Manual

PART B

|  |  |
| --- | --- |
| Roll No. B032 | Name: Naman Garg |
| Class : B Tech CS B | Batch : B2 |
| Date of Experiment: 21/07/2020 | Date of Submission 7/8/2020 |
| Grade : | Time of Submission: |
| Date of Grading: |  |

**B.1 Software Code written by student:**

# naman garg B032

# lab 2 DAA

# AIM: Implementation of Linear Search and Binary Search Technique..

#swaps and comparisons are global variables

swaps=0

comparisons=0

#function for partitioning the array

def partition\_q(arr,low,high):

    global comparisons

    global swaps

    localComparisons=0

    localSwaps=0

    pivot=arr[high]

    print("the pivot element is : ",pivot)

    # i use the last elemnt as pivot

    i=low-1

    for j in range(low,high):

        #incrementing comps,swaps

        comparisons+=1

        localComparisons+=1

        #if element is smaller than pivot

        if(arr[j]<pivot):

            i+=1

            arr[j],arr[i]=arr[i],arr[j] #swapping the elements (toohigh,toolow)

            localSwaps+=1

            swaps+=1

    arr[i+1],arr[high]=arr[high],arr[i+1]

    #printing the current state of the array

    print("Current list : ",arr)

    print("Number of swaps: ",localSwaps)

    print("Number of comparisons: ",localComparisons)

    print('\n')

    #returning the partition

    return i+1

#quicksort recursive func

def quickSort(arr,low,high):

    global comparisons

    if(low<high):

        comparisons+=1

        part=partition\_q(arr,low,high)

        quickSort(arr,low,part-1)

        quickSort(arr,part+1,high)

if \_\_name\_\_ == "\_\_main\_\_":

    userArray= list(map(int,input("Enter list of numbers to sort: ").split()))

    n = len(userArray)

    quickSort(userArray,0,n-1)

    print("Sorted array :")

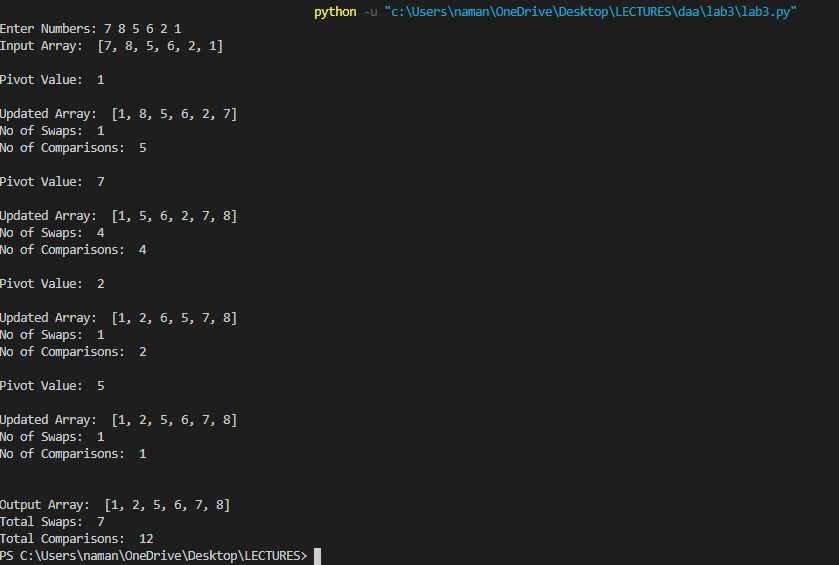
    print(userArray)

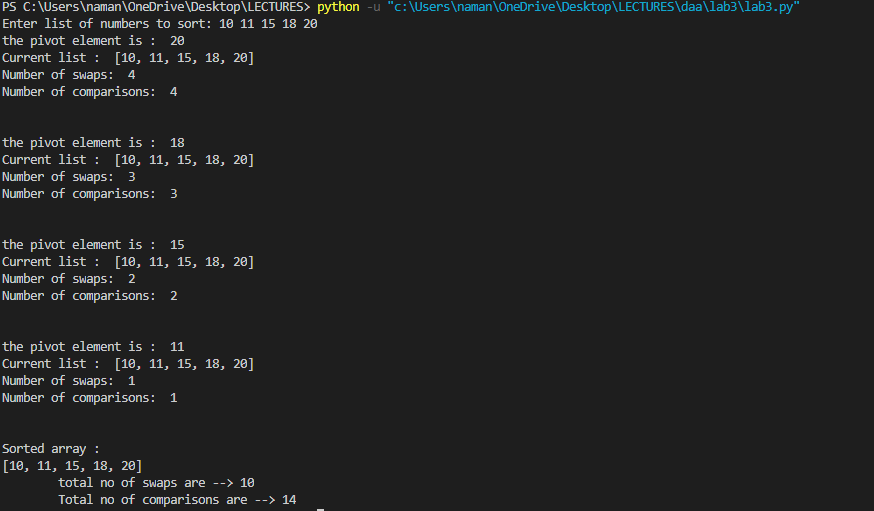
    print("\ttotal no of swaps are -->",swaps)

    print("\tTotal no of comparisons are -->",comparisons)

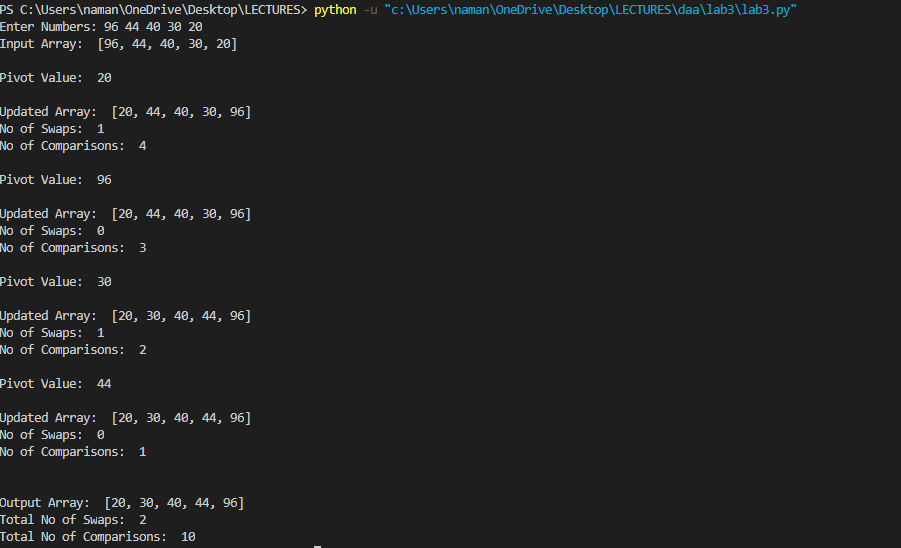
**B.2 Input and Output:**

**Unsorted array::**

****

**Sorted array::**

**Perfectly unsorted array**

**B.3 Observations and learning:**

**B.4 Conclusion:**

**B.5 Question of Curiosity**

Q.1 Comment on the performance of Quick Sort, after filling up the following table.

|  |  |  |
| --- | --- | --- |
| **Array Size(n)** | **Data** | **No. of Comparisons** |
| 4 | 5, 3, 15, 8 | 4 |
| 8 | 1, 3, 3, 4, 5, 6, 9, 19 | 28 |
| 16 | 32, 36, 20, 80, 86, 50, 99, 1, 17, 54, 64, 9, 41, 71, 75, 53 | 47 |
| 32 | 69, 58, 68, 5, 20, 97, 25, 12, 8, 55, 74, 78, 72, 49, 43, 2, 7, 39, 90, 59, 9, 98, 33, 36, 80, 60, 19, 32, 51, 50, 23, 94 | 141 |
| 64 | 84, 46, 23, 26, 8, 96, 81, 68, 17, 6, 66, 41, 86, 77, 30, 70, 31, 51, 13, 2, 94, 88, 24, 55, 22, 47, 92, 53, 71, 42, 74, 61, 28, 5, 50, 79, 60, 89, 14, 67, 9, 76, 49, 34, 44, 33, 75, 69, 7, 25, 43, 59, 62, 78, 48, 35, 56, 99, 3, 64, 80, 45, 57, 10 | 333 |
| 128 | 175, 152, 41, 24, 20, 93, 97, 162, 85, 138, 56, 5, 193, 13, 70, 94, 112, 108, 34, 80, 92, 47, 18, 52, 4, 22, 98, 179, 132, 115, 131, 26, 158, 101, 63, 50, 176, 178, 23, 53, 55, 40, 7, 88, 71, 150, 95, 33, 124, 69, 37, 64, 182, 184, 117, 195, 14, 159, 121, 190, 163, 157, 61, 137, 145, 185, 36, 83, 199, 62, 146, 194, 74, 181, 113, 123, 76, 109, 96, 133, 167, 25, 126, 59, 198, 35, 39, 165, 58, 66, 134, 189, 67, 127, 1, 129, 188, 104, 168, 160, 84, 21, 49, 46, 105, 107, 28, 100, 125, 187, 77, 82, 136, 148, 86, 135, 17, 11, 30, 139, 48, 19, 89, 116, 29, 81, 166, 142 | 742 |

theoretically we know that quicksort is of order N Log N Using the above table we can Justify that claim as we can observe as the number of elements goes up the number of comparisons also go up however not as drastically as we have seen before this is not an exponential relationship it's a logarithmic relationship .

Q.2 Compare the performance of quick sort, with that of insertion sort.

in the previous previous experiment we performed insertion sort and for 128 numbers The number of comparisons was near 4000 however with quick short it is only 742 therefore we require less amount of comparisons thus it equals 2 less amount of time as comparisons are units of computation and take up time

|  |  |  |
| --- | --- | --- |
| Array Size | Quick Sort | Insertion Sort |
| 4 | 4 | 4 |
| 8 | 28 | 12 |
| 16 | 47 | 65 |
| 32 | 141 | 270 |
| 64 | 333 | 1067 |
| 128 | 742 | 4000 |

However as we can see for smaller numbers the number of comparisons was higher for Quicksort rather than insertion sort therefore for small array sizes we should prefer insertion sort

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*