**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 A

(PART A : TO BE REFFERED BY STUDENTS)

**Experiment No.03**

**A.1 Aim:**

Implementation of Divide & Conquer Algorithm Design paradigm.

Write a program to implement Quick Sort.

**A.2 Prerequisite:**

1. Concepts of Divide & Conquer algorithm design technique.

2. Knowledge of Array Handling.

3. Knowledge of recursive programming.

**A.3 Outcome:**

**After successful completion of this experiment students will be able to**

1. Design & implement a sorting algorithm.
2. Design & implement a recursive programs.
3. Design a solution to a given problem using divide & conquer technique.

**A.4 Theory:**

**A.4.1.**

**General Concept of Divide & Conquer**

* **Given a function to compute on *n* inputs, the divide-and-conquer strategy consists of:** 
  1. *splitting the inputs into k distinct subsets, 1*<k≤n*, yielding* k *subproblems.*
  2. *solving these subproblems*
  3. *combining the subsolutions into solution of the whole.*
  4. *if the subproblems are relatively large, then divide\_Conquer is applied again.*
  5. *if the subproblems are small, they are solved without splitting.*

**The Divide and Conquer Algorithm**

Divide\_Conquer(problem P)

{

if Small(P) return S(P);

else {

divide P into smaller instances *P*1, *P*2, …, *Pk*, *k*≥1;

Apply Divide\_Conquer to each of these subproblems;

return Combine(Divide\_Conque(*P*1), Divide\_Conque(*P*2),…, Divide\_Conque(*Pk*));

}

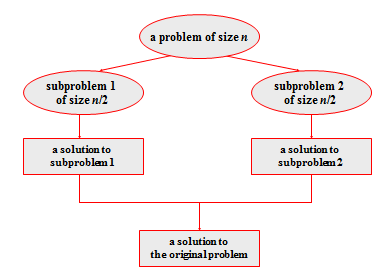
}

**Three Steps of The Divide and Conquer Approach**

**The most well known algorithm design strategy:**

1. Divide the problem into two or more smaller subproblems.
2. Conquer the subproblems by solving them recursively.
3. Combine the solutions to the subproblems into the solutions for the original problem.

**A Typical Divide and Conquer Case**

****

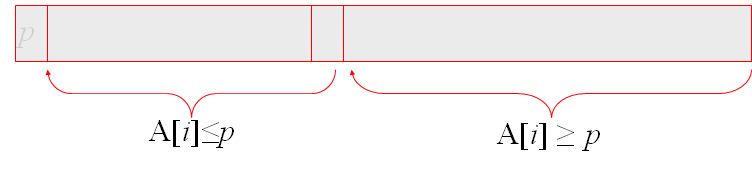
**Figure 1: Divide & Conquer Technique**

**The Divide, Conquer and Combine Steps in Quicksort**

* + Divide: Partition array A[l..r] into 2 subarrays, A[l..s-1] and A[s+1..r] such that each element of the first array is ≤A[s] and each element of the second array is ≥ A[s]. (computing the index of s is part of partition.)
    - *Implication: A[s] will be in its final position in the sorted array.*
  + Conquer: Sort the two subarrays A[l..s-1] and A[s+1..r] by recursive calls to quicksort.
  + Combine: No work is needed, because A[s] is already in its correct place after the partition is done, and the two subarrays have been sorted.

**Quicksort**

* Select a *pivot* whose value we are going to divide the sublist. (e.g., p = A[l])
* Rearrange the list so that it starts with the pivot followed by a ≤ sublist (a sublist whose elements are all smaller than or equal to the pivot) and a ≥ sublist (a sublist whose elements are all greater than or equal to the pivot ) (See algorithm *Partition*  in section A.5)
* Exchange the pivot with the last element in the first sublist(i.e., ≤ sublist) – the pivot is now in its final position.
* Sort the two sublists recursively using quicksort.

****

**Figure 2: Partition of an array**

**A.5 Procedure/Algorithm:**

**A.5.1:**

**The Quicksort Algorithm**

ALGORITHM Quicksort(A[l..r])

//Sorts a subarray by quicksort

//Input: A subarray A[l..r] of A[0..n-1],defined by its left and right indices l and r

//Output: The subarray A[l..r] sorted in nondecreasing order

if l < r

s 🡨 Partition (A[l..r]) // s is a split position

Quicksort(A[l..s-1])

Quicksort(A[s+1..r]

The Partition Algorithm

ALGORITHM Partition (A[l ..r])

//Partitions a subarray by using its first element as a pivot

//Input: A subarray A[l..r] of A[0..n-1], defined by its left and right indices l and r (l < r)

//Output: A partition of A[l..r], with the split position returned as this function’s value

P 🡨A[l]

i 🡨l; j 🡨 r + 1;

Repeat

repeat i 🡨 i + 1 until A[i]>=p //left-right scan

repeat j 🡨j – 1 until A[j] <= p//right-left scan

if (i < j) //need to continue with the scan

swap(A[i], a[j])

until i >= j //no need to scan

swap(A[l], A[j])

return j

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

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)***

|  |  |
| --- | --- |
| Roll No. | Name: |
| Class : | Batch : |
| Date of Experiment: | Date of Submission |
| Grade : | Time of Submission: |
| Date of Grading: |  |

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

***(Paste your c/c++ code completed during the 2 hours of practical in the lab here)***

**B.2 Input and Output:**

***(Paste your program input and output in following format, If there is error then paste the specific error in the output part. In case of error with due permission of the faculty extension can be given to submit the error free code with output in due course of time. Students will be graded accordingly.)***

**Input Data:**

**Output Data:**

**B.3 Observations and learning:**

***(Students are expected to comment on the output obtained with clear observations and learning for each task/ sub part assigned)***

**B.4 Conclusion:**

*(****Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)***

**B.5 Question of Curiosity**

***(To be answered by student based on the practical performed and learning/observations)***

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

|  |  |  |
| --- | --- | --- |
| **Array Size(n)** | **Data** | **No. of Comparisons** |
| 4 |  |  |
| 8 |  |  |
| 16 |  |  |
| 32 |  |  |
| 64 |  |  |
| 128 |  |  |

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

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