Experiment #5		Student ID	
Date		Student Name	

Experiment Title: Implementation of Programs on Divide and Conquer Problems.

Aim/Objective: To understand and implement Divide and Conquer algorithms, and to analyze their performance in solving computational problems.

Description: Divide and Conquer is a powerful algorithmic paradigm used to solve complex problems by breaking them down into simpler sub-problems, solving each sub-problem recursively, and then combining their solutions to solve the original problem. This approach is used in many classical algorithms, such as Strassen's Multiplication, and convex hull algorithms for finding the closest pair of points.

Pre-Requisites:

Understanding of Recursion: Familiarity with the concept of recursion and how recursive functions work. Ability to trace and debug recursive functions.

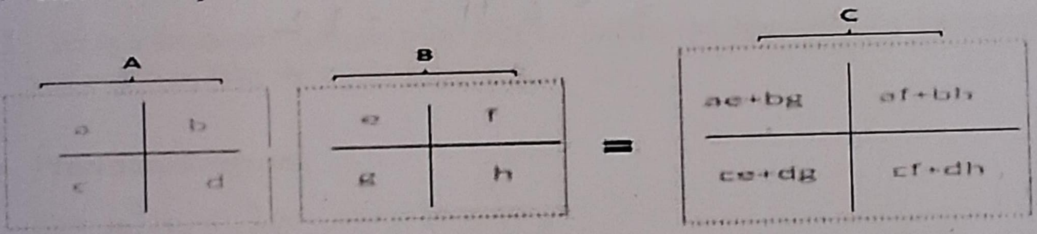
Basic Algorithm Analysis: Knowledge of Big-O notation and how to analyze the time complexity of algorithms. Understanding of recurrence relations and how to solve them.

Basic Data Structures: Proficiency in using arrays, lists, and other fundamental data structures. Understanding of data structures that can be used to implement Divide and Conquer algorithms.

Programming Skills: Competence in a programming language like Python, Java, C++, etc. Familiarity with writing and testing code in an integrated development environment (IDE).

Pre-Lab:

1. Trace the output of the following matrix multiplication using Strassen's Multiplication Method



A, B and C are the Matrices of Size NxN

a, b, c and d are the sub-Matrices of A of size N/2xN/2

e, f, g and h are the sub-Matrices of B of size N/2xN/2

- **Procedure/Program:**

Matrix A: 2 2

$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

Matrix B:

$B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$

submatrices

2   2

$a=1, \quad b=2 \qquad e=5. \quad f=6$

$c=3, \quad d=4 \qquad g=7, \quad h=8$

1) $P_1 = a\cdot(f-h) = 1 \cdot (6-8) = 1 \cdot (-2) = -2$

2) $P_2 = (a+b)\cdot h = (1+2)\cdot 8 = 3 \cdot 8 = 24$

3) $P_3 = P_3 = (c+d)\cdot e = (3+4)\cdot 5 = 7 \cdot 5 = 35$

4) $P_4 = P_4 = d\cdot(g-e) = 4\cdot(7-5) = 4\cdot 2 = 8$

5) $P_5 = (a+d)\cdot(e+h) = (1+4)\cdot(5+8) = 5\cdot 13 = 65$

6) $P_6 = (b-d)\cdot(g+h) = (2-4)\cdot(7+8)$

$= -2\cdot 15 = -30$

7) $P_7 = (a-c)\cdot(e+f) = (1-3)\cdot(5+6)$

$= -2\cdot 11 = -22$

calculate C_{11}:

$C_{11} = P_5 + P_4 - P_2 + P_6 = 65 + 8 - 24 - 30 = 19$

$C_{12} = P_1 + P_2 = -2 + 24 = 22$

$C_{21} = P_3 + P_4 = 35 + 8 = 43$

$C_{22} = P_1 + P_5 - P_3 - P_7 = -2 + 65 - 35 - (-22)$

$= 50$

Final Result.

2

$$C = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

- **Data and Results:**

Data: strassen's algorithm efficiently multiplies matrices using recursive partitioning technique.

Result: The final product of the matrices is $C = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$

- **Analysis and Inferences:**

2. Write a divide and conquer algorithm for finding the maximum and minimum in the sequence of numbers. Find the time complexity.

- **Procedure/Program:**

```
#include <stdio.h>
 int main() {
int arr[] = {3, 1, 5, 2, 4, 6};
  int   n = size of (arr) / size of (arr [0]);
      int max, min;
      if(n<=0) return 0;
 max  = min  = arr[0];

for(int i= 1;   i<n  ; i++){
 if(arr[i] > max)  max = arr[i];
 if(arr[i] < min)  min = arr[i];
  }
 Printf(" Maximum:   %d\n", max);
 Printf(" minimum:   %d\n", min);
  return 0;
}
```

Time complexity:

Time complexity is O(n), where n is the number of elements.

- **Data and Results:**

Data: input consists of a sequence of numbers to analyze

Result: maximum and minimum values are identified from the sequence

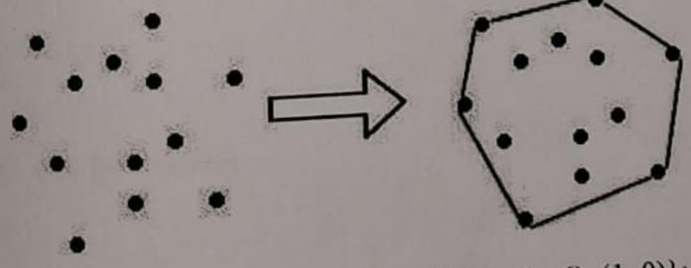
- **Analysis and Inferences:**

Analysis: The algorithm efficiently finds max and min in linear time

inferences: Divide and conquer efficietively optimizes maximum and minimum searches.

In-Lab:

1. Given an input is an array of points specified by their x and y co-ordinates. The output is the convex hull of this set of points by using Divide and Conquer algorithm.



Input : points[] = {(0, 0), (0, 4), (-4, 0), (5, 0), (0, -6), (1, 0)};

Output : (-4, 0), (5, 0), (0, -6), (0, 4)

- **Procedure/Program:**

```
        #include <stdio.h>
      int main(){
        int Points[][2]={{0,0}, {0,4}, {-4,0},
   {5,0}, {0,-6}, {1,0}};
 int n= sizeof(Points) / sizeof(Points[0]), l=0;
 for(int i=1; i<n; i++)
  if(Points[i][0] <Points[l][0]) l = i;
      int P= l, q;
    do{
  Printf("(%d , %d)", Points[P][0], Points[P][1]);
     q=(P+1) % n;
  for(int i = 0; i<n; i++){
 if((Points[q][0] - Points[P][0] )*(Points[i][1] -
                                 points[P][1]) -
```

- **Data and Results:**

```
   (Points[q][1] - Points[P][1]) * (Points[i][0]-
                 Points[P][0] <0)

                  q=i;
```

- **Analysis and Inferences:**

```
                }
               P= q;

             }
```

```
while (P!=l);

    return 0;

}
```

data:

Input consists of Points defined by their x and y coordinates

Result:

convex Hull Points are identified from the given set

Analysis:

convex Hull computed efficiently using the quick hull algorithm

inferences:

quick hull algorithm efficiently determines convex hull for Point sets.

2. Harry's Aunt and family treat him badly and make him work all the time. Dudley, his cousin got homework from school and he as usual handed it over to Harry but Harry has a lot of work and his own homework to do.

The homework is to solve the problems which are numbered in numerical he tries to solve random question after solving random questions he did not put those questions in order Dudley will return in a time of n*logn Harry has to arrange them as soon as possible. Help Harry to solve this problem so that he can go on and do his own homework.

Example

Input

9

15,5,24,8,1,3,16,10,20

Output

1, 3, 5, 8, 10, 15, 16, 20, 24

- **Procedure/Program:**

```
#include<stdio.h>
int main(){
int l=9, a[]={15, 5, 24, 8, 1, 3, 16, 10, 20},
b[9];
for(int s2=1;  s2=l;  s2*=2)
for(int lo=0;  ~~s2~~ lo<l-1;  lo+=2*s2){
    int mid = lo+s2-1;
    int hi = (lo+2*s2-1 < l)? lo+
                          2*s2-1 : l-1;

  int i=lo,  j=mid+1,  k=lo;
 while (i<=mid &&j<=hi) b[k++]=(a[i]<a[j])?
```

```
a[i++] : a[j++];
```

```
while (i <= mid) b[k++] = a[i++];
while (j <= hi) b[k++] = a[j++];

for (k = lo; k <= hi; k++) a[k] = b[k];

 }

for(int i = 0; i < l; i++)
Prinf(i == l-1 ? "%d\n" : "%d,", a[i]);

  return 0;

 }
```

- **Data and Results:**

Data: The initial array contains unsorted numbers for sorting operations

2 2

Result: The array is sorted successfully using merge sort algorithm

2 2

- **Analysis and Inferences:**

Analysis: merge sort efficiently sorts the array in O(n log n) time

2 2

inferences: merge sort Provides stability and efficiency for large datasets

2 2

Post-Lab:

1. Matrix Chain Multiplication

Problem Statement: Given a sequence of matrices, find the most efficient way to multiply these matrices together. The problem is not to perform the multiplications, but to determine the order in which to multiply the matrices such that the total number of scalar multiplications is minimized.

```
def matrix_chain_order_recursive(p, i, j):

if i == j:

return 0

min_cost = sys.maxsize

for k in range(i, j):

cost = (matrix_chain_order_recursive(p, i, k) + matrix_chain_order_recursive(p, k+1, j) + p[i-1] *
p[k] * p[j])

if cost < min_cost:
```

```
min_cost = cost
return min_cost
```

- **Procedure/Program:**

```
    #include <stdio.h>
    #include <limits.h>
    #define  MAX 100

int  matrixchainorder (int P[], int n){
          int   m[MAX][MAX] ={0};

for(int  len =2;   len <n;   len++)

for(int   i =1;   i<n- len +1;   i++){

  int  j= i + len-1;
   m[i][j]  = INT_MAX;

for(int   k= i;  k<j;  k++) {

    int  cost  =  m[i][k]  +m[ k+1][j]+P[i-1]
                                * P[k] * P[j];

     if (cost <m[i][j])

            m[i][j] =cost;

     }
  }
 return  m[1][n-1];

 }
```

```
int  main(){

int P[] = {10, 20, 30, 40, 30};

int n= size of (P) / sizeof (P[0]);

printf("minimum multiplications:  %d\n", matrix
                                chainorder (P, n));

return 0;
}
```

Experiment #5		Student ID	
Date		Student Name	

- **Data and Results:**

Data: The matrix dimensions are Provided for optimal multiplication order calculation.

Result: The optimal order mimizes scalar multiplications for matrix chain

- **Analysis and Inferences:**

analysis: Matrix chain multiplication Problem solved using dynamic Programming efficiently. inferences: Dynamic Programming reduces time complexity in multiplication Problems.

- **Sample VIVA-VOCE Questions (In-Lab):**

1. What is the divide and conquer approach?
2. Can you explain the three main steps involved in the divide and conquer strategy?
3. What are the seven submatrix multiplications used in Strassen's algorithm?
4. What is the convex hull of a set of points?
5. What are the key steps involved in the Graham scan algorithm?

Evaluator Remark (if Any):	**Marks Secured:** 50 **out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

1) Divide and conquer Approach: Breaks Problems into smaller, manageable subproblems recursively

2) Three main steps: Divide, conquer subproblems, and combine results efficiency.

3) Seven submatrix multiplications: P_1, P_2, P_3, P_4, P_5, P_6, P_7 compute efficiently

4) convex Hull: The smallest convex Polygon enclosing a set of Points.

5) Graham scan steps: sort Points, create hull using stack, remove inner points.