Experiment No : I **Date :-**

Divide and Conquer Strategy

Aim :-

a) To implement Binary search and Merge sort
b) To implement Quick sort and MinMax
c) To implement Kth Smallest Element
d) To implement Strassen’s Matrix Multiplication

Theory :-

Divide the problem into a number of subproblems that are smaller instances of the same problem.

Conquer the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.

Combine the solutions to the subproblems into the solution for the original problem.

- Given a function to compute on n inputs the divide and conquer strategy suggest splitting the inputs into k distinct subproblem, 1 < k <= n, yielding k subproblems.
- Subproblems are solved.
- Combined into a solution to the large problem.
- If subproblem are relatively large then divide and conquer strategy is applied again. This is achieved using recursion.

a) Merge Sort

Set 1: Ascending Order

I, T, B, M, Z, F, S, U, G, H, Q

Set 2: Descending Order

81, 43, 61, 21, -8, 96, 55, 77, -18, 52, 17

Binary Search on output of set 1

Search for Q, E

```
Algorithm MergeSort(low, high)
// a[low : high] is a global array to be sorted.
// Small(P) is true if there is only one element
// to sort. In this case the list is already sorted.
{
    if (low < high) then  // If there are more than one element
    {
        // Divide P into subproblems.
            // Find where to split the set.
                mid := ⌊(low + high)/2⌋;
        // Solve the subproblems.
            MergeSort(low, mid);
            MergeSort(mid + 1, high);
        // Combine the solutions.
            Merge(low, mid, high);
    }
}
```

```
Algorithm Merge(low, mid, high)
// a[low : high] is a global array containing two sorted
// subsets in a[low : mid] and in a[mid + 1 : high]. The goal
// is to merge these two sets into a single set residing
// in a[low : high]. b[ ] is an auxiliary global array.
{
    h := low; i := low; j := mid + 1;
    while ((h ≤ mid) and (j ≤ high)) do
    {
        if (a[h] ≤ a[j]) then
        {
            b[i] := a[h];h := h + 1;
        }
        else
        {
            b[i] := a[j]; j := j + 1;
        }
        i := i + 1;
    }
```

```
        if (h > mid) then
            for k := j to high do
            {
                b[i] := a[k]; i := i + 1;
            }
        else
            for k := h to mid do
            {
                b[i] := a[k]; i := i + 1;
            }
        for k := low to high do a[k] := b[k];
}
```

```
Algorithm BinSrch(a, i, l, x)
// Given an array a[i : l] of elements in nondecreasing
// order, 1 ≤ i ≤ l, determine whether x is present, and
// if so, return j such that x = a[j]; else return 0.
{
    if (l = i) then  // If Small(P)
    {
        if (x = a[i]) then return i;
        else return 0;
    }
    else
    { // Reduce P into a smaller subproblem.
        mid := ⌊(i + l)/2⌋;
        if (x = a[mid]) then return mid;
        else if (x < a[mid]) then
                 return BinSrch(a, i, mid − 1, x);
             else return BinSrch(a, mid + 1, l, x);
    }
}
```

b) Perform **Quick Sort** :

43, -12, 11, 58, -5, 29, 65, -17, 37

Find the **maximum and minimum** :

43, -12, 11, 58, -5, 29, 65, -17, 37

```
Algorithm MaxMin(i, j, max, min)
// a[1 : n] is a global array. Parameters i and j are integers,
// 1 ≤ i ≤ j ≤ n. The effect is to set max and min to the
// largest and smallest values in a[i : j], respectively.
{
    if (i = j) then max := min := a[i]; // Small(P)
    else if (i = j − 1) then  // Another case of Small(P)
        {
            if (a[i] < a[j]) then
            {
                max := a[j]; min := a[i];
            }
            else
            {
                max := a[i]; min := a[j];
            }
        }
        else
        {   // If P is not small, divide P into subproblems.
            // Find where to split the set.
                mid := ⌊(i + j)/2⌋;
            // Solve the subproblems.
                MaxMin(i, mid, max, min);
                MaxMin(mid + 1, j, max1, min1);
            // Combine the solutions.
                if (max < max1) then max := max1;
                if (min > min1) then min := min1;
        }
}
```

```
Algorithm QuickSort(p, q)
// Sorts the elements a[p], . . . , a[q] which reside in the global
// array a[1 : n] into ascending order; a[n + 1] is considered to
// be defined and must be ≥ all the elements in a[1 : n].
{
    if (p < q) then  // If there are more than one element
    {
        // divide P into two subproblems.
            j := Partition(a, p, q + 1);
                // j is the position of the partitioning element.
        // Solve the subproblems.
            QuickSort(p, j − 1);
            QuickSort(j + 1, q);
        // There is no need for combining solutions.
    }
}
```

```
Algorithm Partition(a, m, p)
// Within a[m], a[m+1], ..., a[p-1] the elements are
// rearranged in such a manner that if initially t = a[m],
// then after completion a[q] = t for some q between m
// and p-1, a[k] ≤ t for m ≤ k < q, and a[k] ≥ t
// for q < k < p. q is returned. Set a[p] = ∞.
{
    v := a[m]; i := m; j := p;
    repeat
    {
        repeat
            i := i + 1;
        until (a[i] ≥ v);

        repeat
            j := j - 1;
        until (a[j] ≤ v);

        if (i < j) then Interchange(a, i, j);

    } until (i ≥ j);

    a[m] := a[j]; a[j] := v; return j;
}
```

```
Algorithm Interchange(a, i, j)
// Exchange a[i] with a[j].
{
    p := a[i];
    a[i] := a[j]; a[j] := p;
}
```

c) Find the 8th smallest element and the 1st smallest element :

33, 92, 87, 43, 23, 11, 79, 54, 28, 69, 5

Find the 7th smallest element and the 2st smallest element :

K, Y, W, N, I, G, U, Q, J, R, E

```
Algorithm Select2(a, k, low, up)
// Find the k-th smallest in a[low : up].
{
    n := up - low + 1;
    if (n ≤ r) then sort a[low : up] and return the k-th element;
    Divide a[low : up] into n/r subsets of size r each;
    Ignore excess elements;
    Let m[i], 1 ≤ i ≤ (n/r) be the set of medians of
    the above n/r subsets.
    v := Select2(m, ⌈(n/r)/2⌉, 1, n/r);
    Partition a[low : up] using v as the partition element;
    Assume that v is at position j;
    if (k = (j - low + 1)) then return v;
    elseif (k < (j - low + 1)) then
                return Select2(a, k, low, j - 1);
        else return Select2(a, k - (j - low + 1), j + 1, up);
}
```

d) find the Strassen's matrix multiplication (**4x4**) of the following :

A =

-2	5	-8	9
6	-7	4	3
1	-8	6	7
-9	5	3	4

B =

5	-3	8	6
-4	2	9	1
-9	3	-4	7
8	-5	2	6

find the Strassen's matrix multiplication (**2x2**) of the following :

A =

14	-13
-9	8

B =

-17	11
-15	-20

$$
\begin{aligned}
P &= (A_{11} + A_{22})(B_{11} + B_{22}) \\
Q &= (A_{21} + A_{22})B_{11} \\
R &= A_{11}(B_{12} - B_{22}) \\
S &= A_{22}(B_{21} - B_{11}) \\
T &= (A_{11} + A_{12})B_{22} \\
U &= (A_{21} - A_{11})(B_{11} + B_{12}) \\
V &= (A_{12} - A_{22})(B_{21} + B_{22})
\end{aligned}
$$

$$
\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}
$$

$$
\begin{aligned}
C_{11} &= P + S - T + V \\
C_{12} &= R + T \\
C_{21} &= Q + S \\
C_{22} &= P + R - Q + U
\end{aligned}
$$

$$
\begin{aligned}
C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\
C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\
C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\
C_{22} &= A_{21}B_{12} + A_{22}B_{22}
\end{aligned}
$$