Name:- Yash Sanjay Kale
Reg NO:- 2020BIT047

Practical No. 05 Write a C/C++ Code to implement (With Practical example Implementation)

1) Binary Search

```
#include <iostream>
using namespace std;

// Binary search function to search for element in a sorted array
int binarySearch(int arr[], int left, int right, int x) {
    while (left <= right) {
        int mid = left + (right - left) / 2;

        // Check if x is present at mid
        if (arr[mid] == x)
            return mid;

        // If x is greater, ignore left half
        if (arr[mid] < x)
            left = mid + 1;

        // If x is smaller, ignore right half
        else
            right = mid - 1;
    }

    // If we reach here, then element was not present
    return -1;
}

int main() {
    int arr[] = {2, 4, 6, 8, 10};
    int n = sizeof(arr) / sizeof(arr[0]);
    int x = 8;
    int result = binarySearch(arr, 0, n - 1, x);
    if (result == -1)
        cout << "Element is not present in array";
    else
        cout << "Element is present at index " << result;
    return 0;
}
```

2. Merge Sort

```
#include <iostream>
using namespace std;

// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;

    // Create temporary arrays
    int L[n1], R[n2];

    // Copy data to temporary arrays L[] and R[]
    for (int i = 0; i < n1; i++)
```

```
        L[i] = arr[l + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    // Merge the temporary arrays back into arr[l..r]
    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    // Copy the remaining elements of L[], if there are any
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    // Copy the remaining elements of R[], if there are any
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

// Merge sort function to sort an array
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        // Merge the sorted halves
        merge(arr, l, m, r);
    }
}

int main() {
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Given array is:" << endl;
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";

    mergeSort(arr, 0, n - 1);

    cout << "\nSorted array is:" << endl;
    for (int i = 0; i < n; i++)
```

```
        cout << arr[i] << " ";
    return 0;
}
```

3.Quick Sort

```
#include <iostream>
using namespace std;

// Partition function to partition the array using a pivot element
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }

    swap(arr[i + 1], arr[high]);
    return i + 1;
}

// Quick sort function to sort an array
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);

        // Sort the subarrays on either side of the pivot
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

int main() {
    int arr[] = { 10, 7, 8, 9, 1, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Given array is:" << endl;
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";

    quickSort(arr, 0, n - 1);

    cout << "\nSorted array is:" << endl;
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

4. Strassen's Matrix multiplication

```
#include <iostream>
#include <vector>
using namespace std;
```

```
// Strassen's matrix multiplication function for two matrices of size n x n
vector<vector<int>> strassenMatrixMultiply(vector<vector<int>>& A,
vector<vector<int>>& B) {
    int n = A.size();

    // Base case
    if (n == 1) {
        vector<vector<int>> C(1, vector<int>(1));
        C[0][0] = A[0][0] * B[0][0];
        return C;
    }

    // Split matrices into submatricesvector<vector<int>> A11(n/2,
vector<int>(n/2)), A12(n/2, vector<int>(n/2)),
 A21(n/2, vector<int>(n/2)), A22(n/2, vector<int>(n/2));
    vector<vector<int>> B11(n/2, vector<int>(n/2)), B12(n/2, vector<int>(n/2)),
   B21(n/2, vector<int>(n/2)), B22(n/2, vector<int>(n/2));
    for (int i = 0; i < n/2; i++) {
        for (int j = 0; j < n/2; j++) {
            A11[i][j] = A[i][j];
            A12[i][j] = A[i][j+n/2];
            A21[i][j] = A[i+n/2][j];
            A22[i][j] = A[i+n/2][j+n/2];

            B11[i][j] = B[i][j];
            B12[i][j] = B[i][j+n/2];
            B21[i][j] = B[i+n/2][j];
            B22[i][j] = B[i+n/2][j+n/2];
        }
    }

    // Compute the seven Strassen's submatrices
    vector<vector<int>> M1 = strassenMatrixMultiply(A11, B12 - B22);
    vector<vector<int>> M2 = strassenMatrixMultiply(A11 + A12, B22);
    vector<vector<int>> M3 = strassenMatrixMultiply(A21 + A22, B11);
    vector<vector<int>> M4 = strassenMatrixMultiply(A22, B21 - B11);
    vector<vector<int>> M5 = strassenMatrixMultiply(A11 + A22, B11 + B22);
    vector<vector<int>> M6 = strassenMatrixMultiply(A12 - A22, B21 + B22);
    vector<vector<int>> M7 = strassenMatrixMultiply(A11 - A21, B11 + B12);

    // Compute the four quadrants of the resulting matrix C
    vector<vector<int>> C11(n/2, vector<int>(n/2)), C12(n/2, vector<int>(n/2)),
C21(n/2,
     vector<int>(n/2)), C22(n/2, vector<int>(n/2));
    for (int i = 0; i < n/2; i++) {
        for (int j = 0; j < n/2; j++) {
            C11[i][j] = M5[i][j] + M4[i][j] - M2[i][j] + M6[i][j];
            C12[i][j] = M1[i][j] + M2[i][j];
            C21[i][j] = M
```