Name: Amruta Pede

Reg No :2020BIT070

Assignment 5

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

1) Binary Search

2) Merge Sort

3) Quick Sort

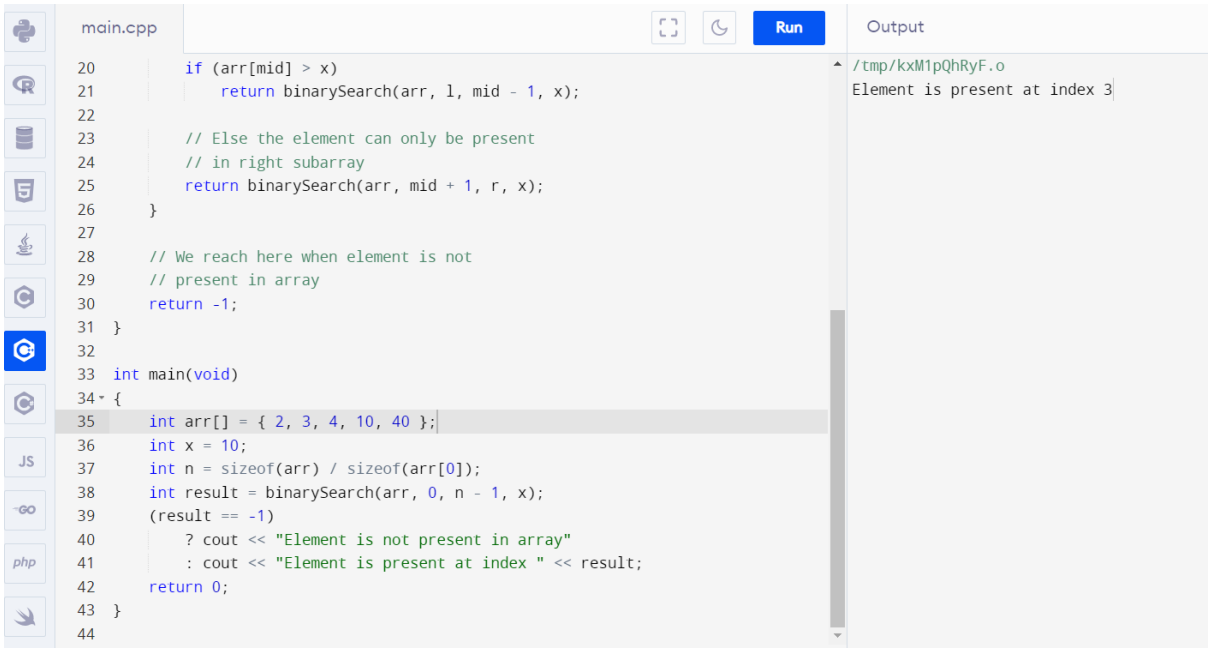
4) Strassen's Matrix multiplication

1) Binary Search:

```
// C++ program to implement recursive Binary Search
#include <bits/stdc++.h>
using namespace std;

// A recursive binary search function. It returns
// location of x in given array arr[l..r] is present,
// otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
	if (r >= l) {
		int mid = l + (r - l) / 2;

		// If the element is present at the middle
		// itself
		if (arr[mid] == x)
			return mid;
```

```
		// If element is smaller than mid, then
		// it can only be present in left subarray
		if (arr[mid] > x)
			return binarySearch(arr, l, mid - 1, x);

		// Else the element can only be present
		// in right subarray
		return binarySearch(arr, mid + 1, r, x);
	}

	// We reach here when element is not
	// present in array
	return -1;
}

int main(void)
{
	int arr[] = { 2, 3, 4, 10, 40 };
	int x = 10;
	int n = sizeof(arr) / sizeof(arr[0]);
	int result = binarySearch(arr, 0, n - 1, x);
	(result == -1)
		? cout << "Element is not present in array"
		: cout << "Element is present at index " << result;
	return 0;
}
```

Output :




## 2)Merge sort:

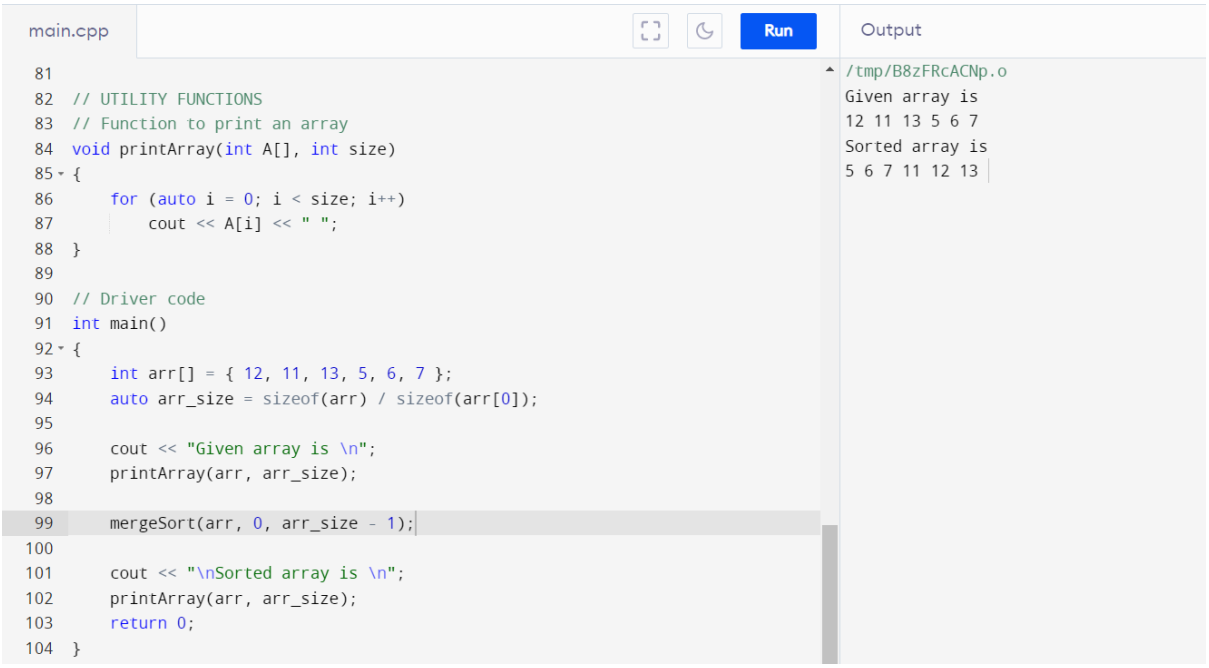
```
// Online C++ compiler to run C++ program online
// C++ program for Merge Sort
#include <iostream>
using namespace std;

// Merges two subarrays of array[].
// First subarray is arr[begin..mid]
// Second subarray is arr[mid+1..end]
void merge(int array[], int const left, int const mid,
                        int const right)
{
        auto const subArrayOne = mid - left + 1;
        auto const subArrayTwo = right - mid;

        // Create temp arrays
        auto *leftArray = new int[subArrayOne],
                *rightArray = new int[subArrayTwo];
```

```
// Copy data to temp arrays leftArray[] and rightArray[]
for (auto i = 0; i < subArrayOne; i++)
        leftArray[i] = array[left + i];
for (auto j = 0; j < subArrayTwo; j++)
        rightArray[j] = array[mid + 1 + j];

auto indexOfSubArrayOne
        = 0, // Initial index of first sub-array
        indexOfSubArrayTwo
        = 0; // Initial index of second sub-array
int indexOfMergedArray
        = left; // Initial index of merged array

// Merge the temp arrays back into array[left..right]
while (indexOfSubArrayOne < subArrayOne
        && indexOfSubArrayTwo < subArrayTwo) {
        if (leftArray[indexOfSubArrayOne]
                <= rightArray[indexOfSubArrayTwo]) {
                array[indexOfMergedArray]
                        = leftArray[indexOfSubArrayOne];
                indexOfSubArrayOne++;
        }
        else {
                array[indexOfMergedArray]
                        = rightArray[indexOfSubArrayTwo];
                indexOfSubArrayTwo++;
        }
        indexOfMergedArray++;
}
// Copy the remaining elements of
```

```
        // left[], if there are any
        while (indexOfSubArrayOne < subArrayOne) {
                array[indexOfMergedArray]
                        = leftArray[indexOfSubArrayOne];
                indexOfSubArrayOne++;
                indexOfMergedArray++;
        }
        // Copy the remaining elements of
        // right[], if there are any
        while (indexOfSubArrayTwo < subArrayTwo) {
                array[indexOfMergedArray]
                        = rightArray[indexOfSubArrayTwo];
                indexOfSubArrayTwo++;
                indexOfMergedArray++;
        }
        delete[] leftArray;
        delete[] rightArray;
}

// begin is for left index and end is
// right index of the sub-array
// of arr to be sorted */
void mergeSort(int array[], int const begin, int const end)
{
        if (begin >= end)
                return; // Returns recursively

        auto mid = begin + (end - begin) / 2;
        mergeSort(array, begin, mid);
        mergeSort(array, mid + 1, end);
        merge(array, begin, mid, end);
```

```
}

// UTILITY FUNCTIONS
// Function to print an array
void printArray(int A[], int size)
{
	for (auto i = 0; i < size; i++)
		cout << A[i] << " ";
}

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

	cout << "Given array is \n";
	printArray(arr, arr_size);

	mergeSort(arr, 0, arr_size - 1);

	cout << "\nSorted array is \n";
	printArray(arr, arr_size);
	return 0;
}
```

Output:




## 3)Quick Sort:

/* C++ implementation of QuickSort */

#include <bits/stdc++.h>

using namespace std;

// A utility function to swap two elements

void swap(int* a, int* b)

{

int t = *a;

*a = *b;

*b = t;

}

/* This function takes last element as pivot, places

the pivot element at its correct position in sorted

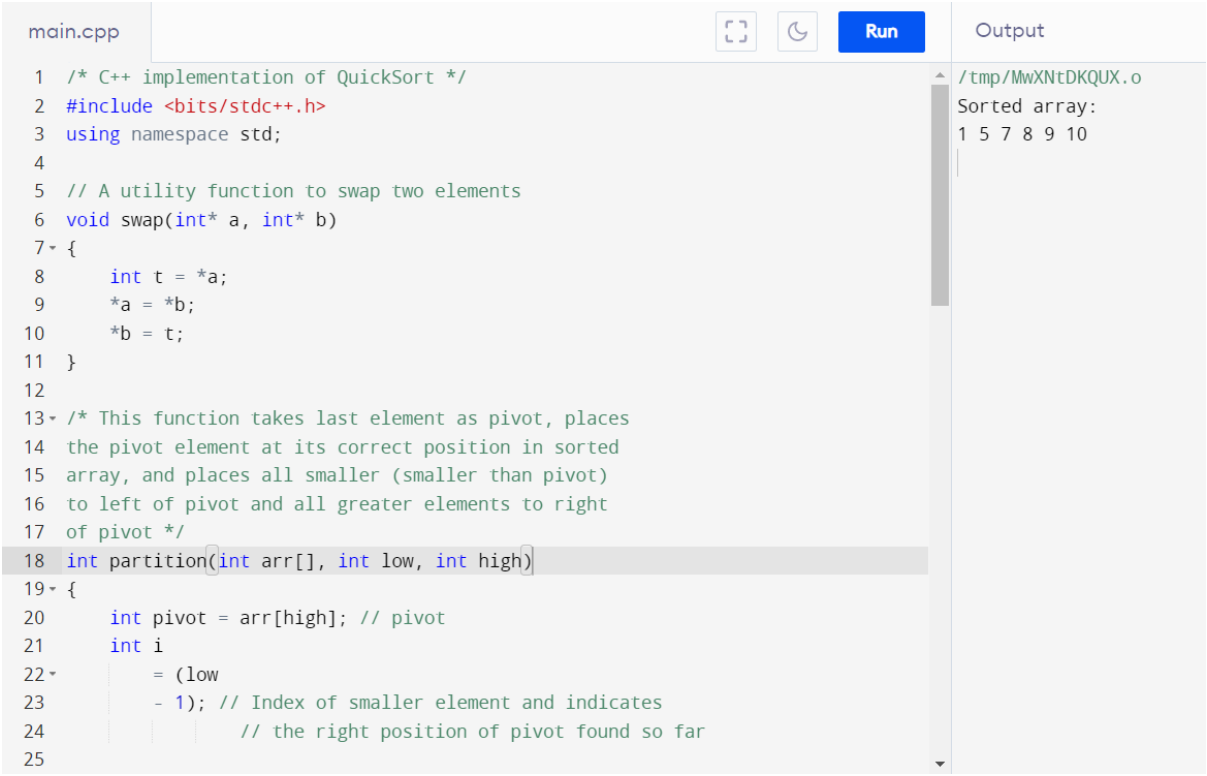
array, and places all smaller (smaller than pivot)

to left of pivot and all greater elements to right

```
of pivot */
int partition(int arr[], int low, int high)
{
	int pivot = arr[high]; // pivot
	int i
		= (low
		- 1); // Index of smaller element and indicates
				// the right position of pivot found so far

	for (int j = low; j <= high - 1; j++) {
		// If current element is smaller than the pivot
		if (arr[j] < pivot) {
			i++; // increment index of smaller element
			swap(&arr[i], &arr[j]);
		}
	}
	swap(&arr[i + 1], &arr[high]);
	return (i + 1);
}

/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
void quickSort(int arr[], int low, int high)
{
	if (low < high) {
		/* pi is partitioning index, arr[p] is now
		at right place */
		int pi = partition(arr, low, high);
```

```
                // Separately sort elements before
                // partition and after partition
                quickSort(arr, low, pi - 1);
                quickSort(arr, pi + 1, high);
        }
}

/* Function to print an array */
void printArray(int arr[], int size)
{
        int i;
        for (i = 0; i < size; i++)
                cout << arr[i] << " ";
        cout << endl;
}

// Driver Code
int main()
{
        int arr[] = { 10, 7, 8, 9, 1, 5 };
        int n = sizeof(arr) / sizeof(arr[0]);
        quickSort(arr, 0, n - 1);
        cout << "Sorted array: \n";
        printArray(arr, n);
        return 0;
}
```

## Output:




## 4) Strassen's Matrix multiplication :

```
#include <bits/stdc++.h>

using namespace std;


#define ROW_1 4

#define COL_1 4


#define ROW_2 4

#define COL_2 4


void print(string display, vector<vector<int> > matrix,

                int start_row, int start_column, int end_row,

                int end_column)

{

        cout << endl << display << " =>" << endl;

        for (int i = start_row; i <= end_row; i++) {
```

```
                for (int j = start_column; j <= end_column; j++) {
                        cout << setw(10);
                        cout << matrix[i][j];
                }
                cout << endl;
        }
        cout << endl;
        return;
}

void add_matrix(vector<vector<int> > matrix_A,
                                vector<vector<int> > matrix_B,
                                vector<vector<int> >& matrix_C,
                                int split_index)
{
        for (auto i = 0; i < split_index; i++)
                for (auto j = 0; j < split_index; j++)
                        matrix_C[i][j]
                                = matrix_A[i][j] + matrix_B[i][j];
}

vector<vector<int> >
multiply_matrix(vector<vector<int> > matrix_A,
                                vector<vector<int> > matrix_B)
{
        int col_1 = matrix_A[0].size();
        int row_1 = matrix_A.size();
        int col_2 = matrix_B[0].size();
        int row_2 = matrix_B.size();

        if (col_1 != row_2) {
```

```
        cout << "\nError: The number of columns in Matrix "
                        "A must be equal to the number of rows in "
                        "Matrix B\n";
        return {};
}

vector<int> result_matrix_row(col_2, 0);
vector<vector<int> > result_matrix(row_1,
                                                        result_matrix_row);

if (col_1 == 1)
        result_matrix[0][0]
                = matrix_A[0][0] * matrix_B[0][0];
else {
        int split_index = col_1 / 2;

        vector<int> row_vector(split_index, 0);
        vector<vector<int> > result_matrix_00(split_index,
row_vector);
        vector<vector<int> > result_matrix_01(split_index,
row_vector);
        vector<vector<int> > result_matrix_10(split_index,
row_vector);
        vector<vector<int> > result_matrix_11(split_index,
row_vector);

        vector<vector<int> > a00(split_index, row_vector);
        vector<vector<int> > a01(split_index, row_vector);
```

```
vector<vector<int> > a10(split_index, row_vector);
vector<vector<int> > a11(split_index, row_vector);
vector<vector<int> > b00(split_index, row_vector);
vector<vector<int> > b01(split_index, row_vector);
vector<vector<int> > b10(split_index, row_vector);
vector<vector<int> > b11(split_index, row_vector);

for (auto i = 0; i < split_index; i++)
        for (auto j = 0; j < split_index; j++) {
                a00[i][j] = matrix_A[i][j];
                a01[i][j] = matrix_A[i][j + split_index];
                a10[i][j] = matrix_A[split_index + i][j];
                a11[i][j] = matrix_A[i + split_index]
                                                        [j + split_index];
                b00[i][j] = matrix_B[i][j];
                b01[i][j] = matrix_B[i][j + split_index];
                b10[i][j] = matrix_B[split_index + i][j];
                b11[i][j] = matrix_B[i + split_index]
                                                        [j + split_index];
        }

add_matrix(multiply_matrix(a00, b00),
                multiply_matrix(a01, b10),
                result_matrix_00, split_index);
add_matrix(multiply_matrix(a00, b01),
                multiply_matrix(a01, b11),
                result_matrix_01, split_index);
add_matrix(multiply_matrix(a10, b00),
                multiply_matrix(a11, b10),
                result_matrix_10, split_index);
add_matrix(multiply_matrix(a10, b01),
```

```
                multiply_matrix(a11, b11),
                result_matrix_11, split_index);

        for (auto i = 0; i < split_index; i++)
            for (auto j = 0; j < split_index; j++) {
                result_matrix[i][j]
                    = result_matrix_00[i][j];
                result_matrix[i][j + split_index]
                    = result_matrix_01[i][j];
                result_matrix[split_index + i][j]
                    = result_matrix_10[i][j];
                result_matrix[i + split_index]
                                    [j + split_index]
                    = result_matrix_11[i][j];
            }

        result_matrix_00.clear();
        result_matrix_01.clear();
        result_matrix_10.clear();
        result_matrix_11.clear();
        a00.clear();
        a01.clear();
        a10.clear();
        a11.clear();
        b00.clear();
        b01.clear();
        b10.clear();
        b11.clear();
    }
    return result_matrix;
}
```

```
int main()
{
	vector<vector<int> > matrix_A = { { 1, 1, 1, 1 },
                                  { 2, 2, 2, 2 },
                                  { 3, 3, 3, 3 },
                                  { 2, 2, 2, 2 } };

	print("Array A", matrix_A, 0, 0, ROW_1 - 1, COL_1 - 1);

	vector<vector<int> > matrix_B = { { 1, 1, 1, 1 },
                                  { 2, 2, 2, 2 },
                                  { 3, 3, 3, 3 },
                                  { 2, 2, 2, 2 } };

	print("Array B", matrix_B, 0, 0, ROW_2 - 1, COL_2 - 1);

	vector<vector<int> > result_matrix(
		multiply_matrix(matrix_A, matrix_B));

	print("Result Array", result_matrix, 0, 0, ROW_1 - 1,
		COL_2 - 1);
}
```

## Output:

main.cpp

Run

Output

```
137
138 int main()
139 {
140     vector<vector<int> > matrix_A = { { 1, 1, 1, 1 },
141                                      { 2, 2, 2, 2 },
142                                      { 3, 3, 3, 3 },
143                                      { 2, 2, 2, 2 } };
144
145     print("Array A", matrix_A, 0, 0, ROW_1 - 1, COL_1 - 1);
146
147     vector<vector<int> > matrix_B = { { 1, 1, 1, 1 },
148                                      { 2, 2, 2, 2 },
149                                      { 3, 3, 3, 3 },
150                                      { 2, 2, 2, 2 } };
151
152     print("Array B", matrix_B, 0, 0, ROW_2 - 1, COL_2 - 1);
153
154     vector<vector<int> > result_matrix(
155         multiply_matrix(matrix_A, matrix_B));
156
157     print("Result Array", result_matrix, 0, 0, ROW_1 - 1,
158         COL_2 - 1);
159 }
160
```

```
/tmp/NhdMo7ghTP.o
Array A =>
        1       1       1       1
        2       2       2       2
        3       3       3       3
        2       2       2       2


Array B =>
        1       1       1       1
        2       2       2       2
        3       3       3       3
        2       2       2       2

Result Array =>
        8       8       8       8
        16      16      16      16
        24      24      24      24
        16      16      16      16
```