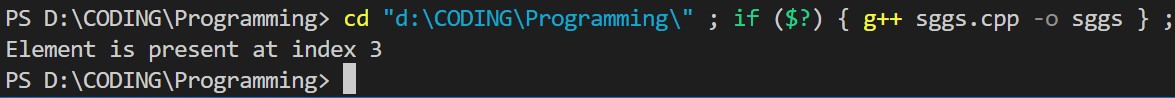
**Assignment – 6**

Name : Abhishek satapure

1. **Binary Search**

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  int binarySearch(int arr[], int l, int r, int x)  { if (r >= l) { int mid = l + (r - l) / 2;  if (arr[mid] == x) return mid;  if (arr[mid] > x) return binarySearch(arr, l, mid - 1, x);  return binarySearch(arr, mid + 1, r, x);  } return -1;  } int main(void){ int arr[] = { 2, 3, 4, 10, 40 }; int x = 10; int result = binarySearch(arr, 0, 5, x);  (result == -1)  ? cout << "Element is not present in array"  : cout << "Element is present at index " << result; return 0;  } |

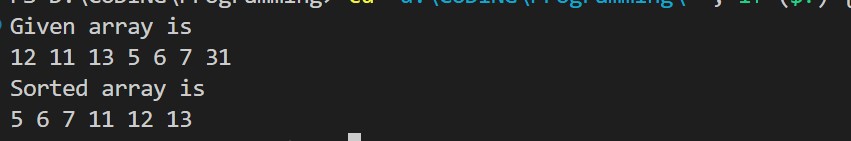
**Output:**



1. **Merge Sort**

|  |
| --- |
| #include <iostream>  using namespace std;  void merge(int array[], int const left, int const mid, int const right)  { auto const subArrayOne = mid - left + 1; auto const subArrayTwo = right - mid;  auto \*leftArray = new int[subArrayOne],  \*rightArray = new int[subArrayTwo];  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, indexOfSubArrayTwo= 0; int indexOfMergedArray = left;  while (indexOfSubArrayOne < subArrayOne && indexOfSubArrayTwo < subArrayTwo) { if (leftArray[indexOfSubArrayOne] <= rightArray[indexOfSubArrayTwo]) { array[indexOfMergedArray]  = leftArray[indexOfSubArrayOne]; indexOfSubArrayOne++;  } else { array[indexOfMergedArray]  = rightArray[indexOfSubArrayTwo]; indexOfSubArrayTwo++;  } indexOfMergedArray++;  } while (indexOfSubArrayOne < subArrayOne) { array[indexOfMergedArray]  = leftArray[indexOfSubArrayOne]; indexOfSubArrayOne++; indexOfMergedArray++;  } while (indexOfSubArrayTwo < subArrayTwo) { |
| array[indexOfMergedArray]  = rightArray[indexOfSubArrayTwo]; indexOfSubArrayTwo++; indexOfMergedArray++;  } delete[] leftArray; delete[] rightArray;  } void mergeSort(int array[], int const begin, int const end)  { if (begin >= end) return;  auto mid = begin + (end - begin) / 2; mergeSort(array, begin, mid); mergeSort(array, mid + 1, end); merge(array, begin, mid, end);  } void printArray(int A[], int size)  { for (auto i = 0; i < size; i++) cout << A[i] << " ";  }  int main()  { int arr[] = { 12, 11, 13, 5, 6, 7 };  cout << "Given array is \n"; printArray(arr, 7);  mergeSort(arr, 0, 6);  cout << "\nSorted array is \n"; printArray(arr, 6); return 0;  } |

**Output:**

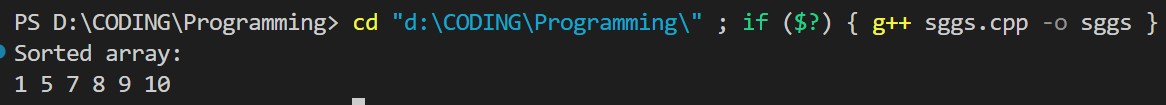


1. **Quick Sort**

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std; void swap(int\* a, int\* b)  { int t = \*a;  \*a = \*b;  \*b = t;  } int partition(int arr[], int low, int high)  { int pivot = arr[high]; int i = (low- 1); for (int j = low; j <= high - 1; j++) { if (arr[j] < pivot) { i++; swap(&arr[i], &arr[j]);  } } swap(&arr[i + 1], &arr[high]); return (i + 1);  } void quickSort(int arr[], int low, int high){ if (low < high) { int pi = partition(arr, low, high); quickSort(arr, low, pi - 1); quickSort(arr, pi + 1, high);  } } void printArray(int arr[], int size){ int i; for (i = 0; i < size; i++) cout << arr[i] << " "; cout << endl;  }  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 :



1. **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:**

