**Searching and Sorting Algorithms**

**3A. Binary Search:** Implement a program to search and find the student details in

an efficient manner. Reduce the number of comparisons as much as possible. Use the student’s registration number as the key. Store the student’s name, registration number, phone number and CGPA in the list of student details.

**Algorithm:**

Step 1: Create class Student to store the details of the students with attributes reg no, first name, last name, phone number, and CGPA.

Step 2: Input the number of students ‘n’.

Step 3: Initialize an array of type Student of length n.

Step 4: Input the details of n students and store it in the array initialized above.

Step 5: Sort the given array using merge sort.

Step 6: For implementing merge sort, recursively divide the array into two sub arrays, and when the base case is reached, merge the sorted subarrays, leading up the sorted main array.

Step 7: Input the search query.

Step 8: Binary search the sorted main array for the query.

Step 9: Binary search is implemented by the following method:



**Program:** [In next page]

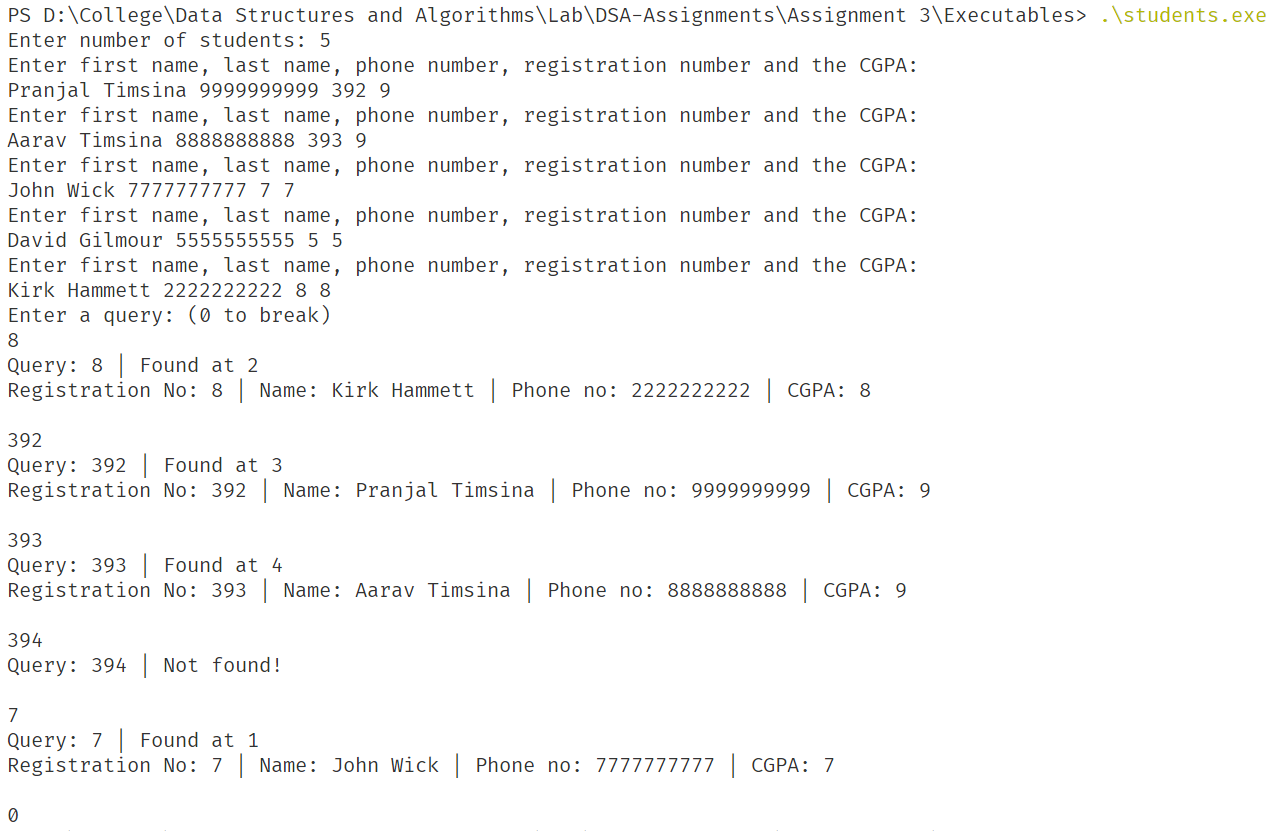








**Output:**

****

**Results:**

Thus, the program to search and find student details in an efficient manner is implemented.

**3B. Application of Binary Search:** Implement a program to find the square root of a number. User can give the number randomly. Floor the result in case of floating point.

**Algorithm:**

Step 1: Input the number from the user.

Step 2: Call the square root function which is implemented in the following way:



Step 3: Print the value returned by the function.

**Program:** [In next page]

****

**Output:**



**Results:**

Thus, the program to find the square root of a number using binary search is implemented.

**2C. Bubble Sort:** Sort the given array of elements in ascending order and print out the number of comparisons performed.

**Algorithm:**

Step 1: Input the number of elements in an array.

Step 2: Initialize an array of appropriate data type.

Step 3: Input the elements of the array

Step 4: Call bubble sort on the array which is implement as follows:



Step 5: Print the sorted array, the number of comparisons and the number of array accesses.

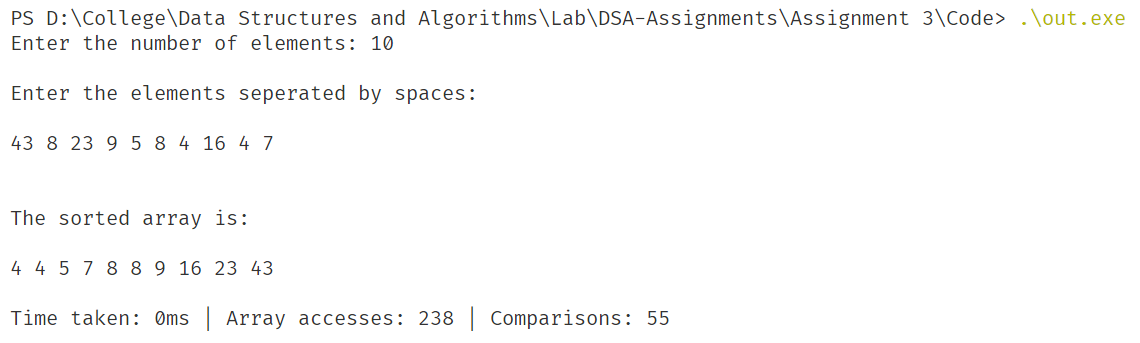
**Program:** [In next page]



[Continued in next page]



**Output:**

****

**Results:**

Thus, the program to sort the given array using bubble sort is implemented.

**2D.** **Insertion Sort:** Sort the given array of elements in ascending order and print out the number of comparisons performed.

**Algorithm:**

Step 1: Input the number of elements in an array.

Step 2: Initialize an array of appropriate data type.

Step 3: Input the elements of the array

Step 4: Call insertion sort on the array which is implement as follows:



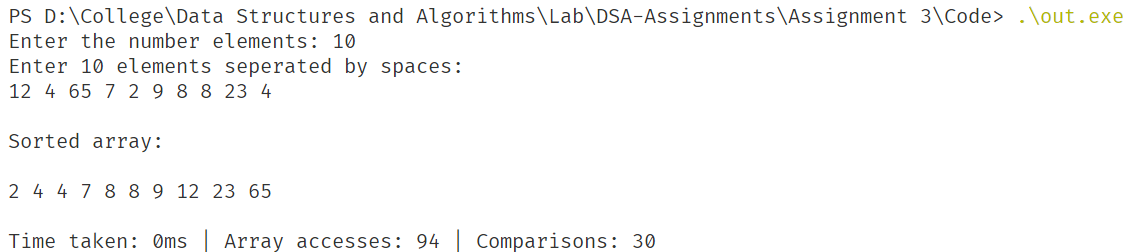
Step 5: Print the sorted array, the number of comparisons and the number of array accesses.

**Program:** [In next page]





**Output:**

****

**Results:**

Thus, the program to sort the given array using insertion sort is implemented.

**2E. Selection Sort**: Sort the given array of elements in ascending order and print out the number of comparisons performed.

**Algorithm:**

Step 1: Input the number of elements in an array.

Step 2: Initialize an array of appropriate data type.

Step 3: Input the elements of the array

Step 4: Call selection sort on the array which is implement as follows:



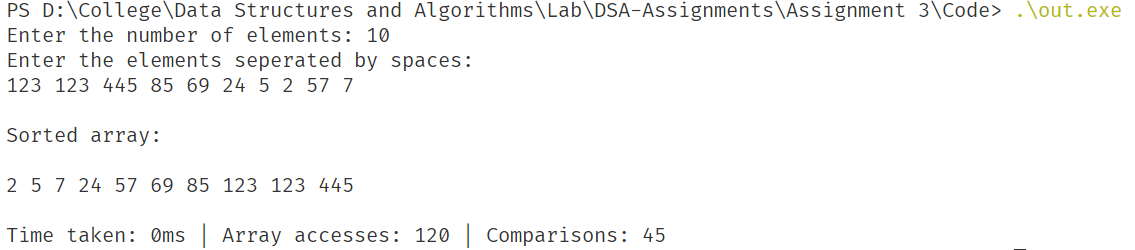
Step 5: Print the sorted array, the number of comparisons and the number of array accesses.

**Program:** [In next page]





**Output:**



**Results:**

Thus, the program to sort the given array using selection sort is implemented.

**2F. Quick Sort**: Sort the given array of elements in ascending order and print out the number of comparisons performed.

**Algorithm:**

Step 1: Input the number of elements in an array.

Step 2: Initialize an array of appropriate data type.

Step 3: Input the elements of the array

Step 4: Call quick sort on the array which is implement as follows:



Step 5: Print the sorted array, the number of comparisons and the number of array accesses.

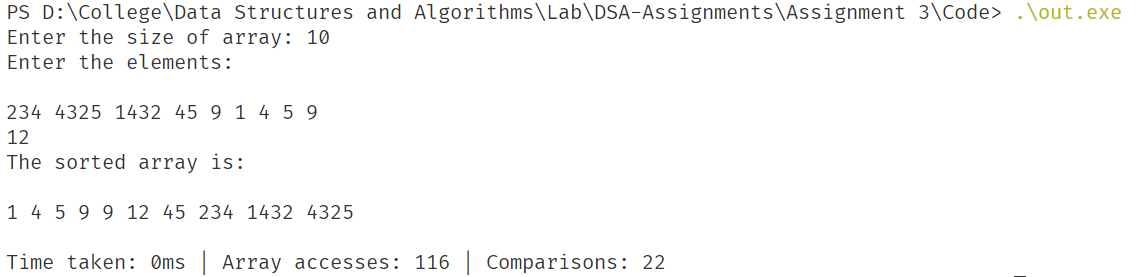
**Program:** [In next page]







**Output:**



**Results:**

Thus, the program to sort the given array using quick sort is implemented.

**2G. Merge Sort**: Sort the given array of elements in ascending order and print out the number of comparisons performed.

**Algorithm:**

Step 1: Input the number of elements in an array.

Step 2: Initialize an array of appropriate data type.

Step 3: Input the elements of the array

Step 4: Call merge sort on the array which is implement as follows:



Step 5: Print the sorted array, the number of comparisons and the number of array accesses.

**Program:**



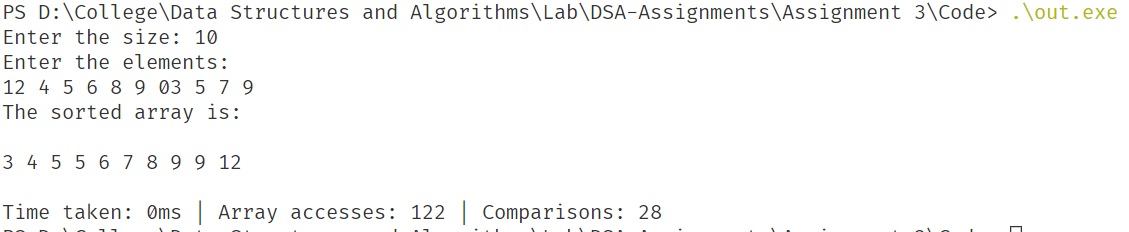
[Continued in next page]



[Continued in next page]



**Output:**



**Results:**

Thus, the program to sort the given array using merge sort is implemented.

**2H.** Compare the number of comparisons of various sorting algorithms mentioned in above questions 3 to 6. Print a table which shows the input array, and number of comparisons performed by various algorithms. Reuse above sorting programs as functions in this new program.

**[Note: Since there are up to 1,00,000 elements in the examples below, I have used a .txt file as an input and since it is not feasible to print out all 1,00,000 elements, I have not printed the sorted array.]**

**Algorithm:**

Step 1: Implement the algorithms as in the above questions.

Step 2: Read inputs from a text file.

Step 3: Run each of the sorting algorithms on the array.

Step 4: Print the statistics of the time and operations done by the sorting algorithms

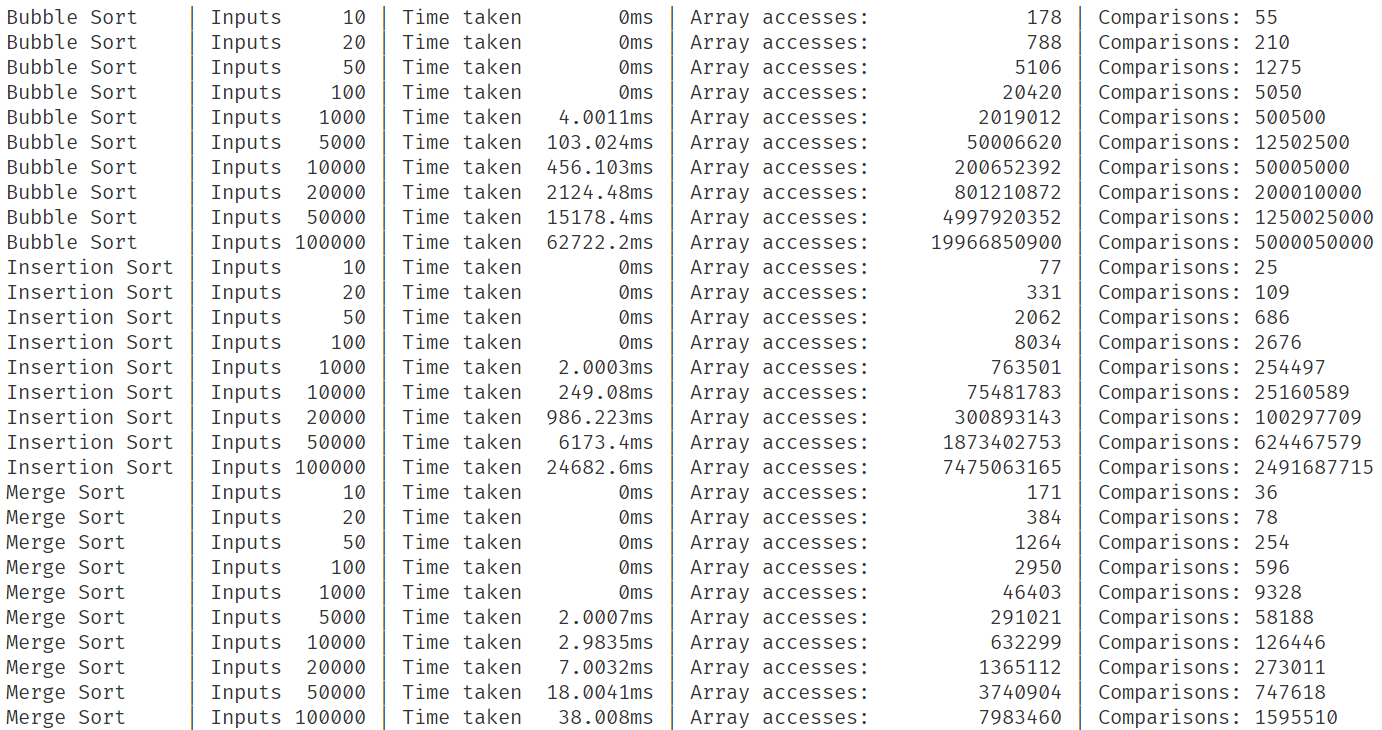
**Program:** [In next page]

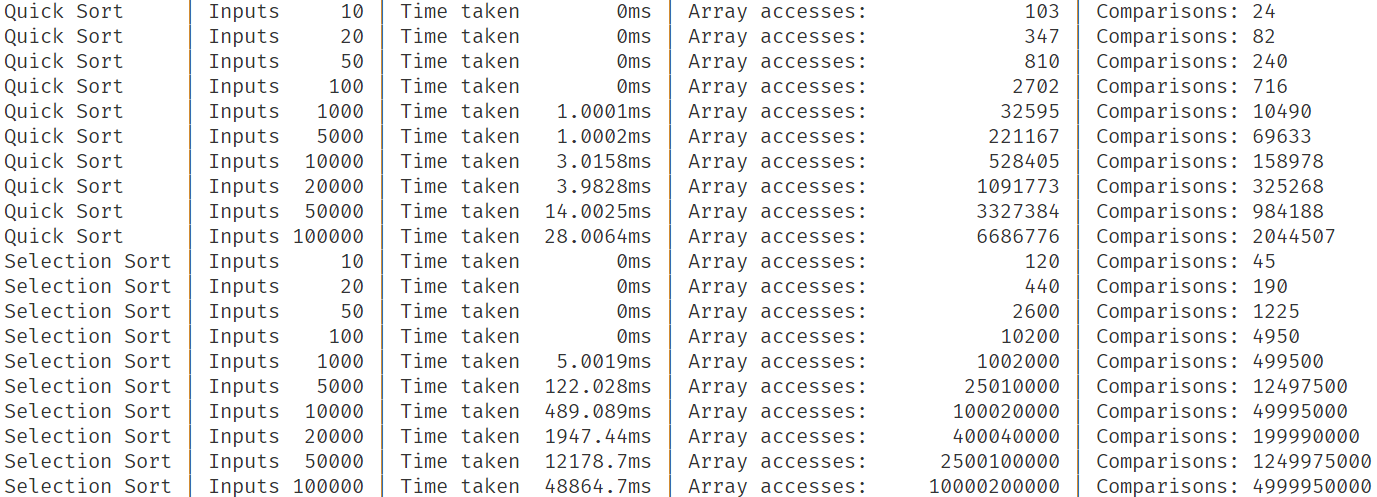
********

****

****

**Output:**

****

****

**Results:**

From the results above, we can verify the order of growth of different sorting algorithms. It is evident that for a very small number of inputs (< 500) the sorting algorithms do not make much difference; however, as the number of inputs grows, merge sort and quick sort prove to be much faster. Also, despite having a worst-case time complexity of n2, Quick sort, always performs the best.