Experiment No.2:- Quick Sort and Merge Sort

Reg No.:-24141045

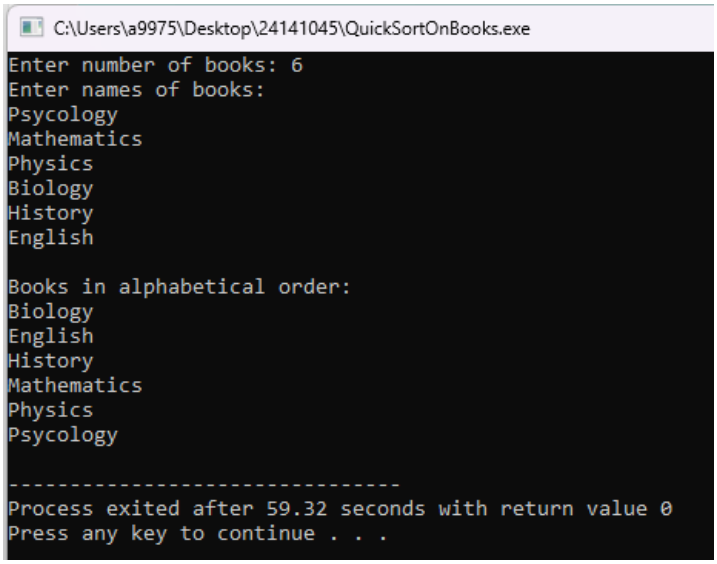
Program1:-

```
#include <iostream>
#include <string>
using namespace std;
void swap(string &a, string &b) {
   string temp = a;
   a = b;
   b = temp;
}
int partition(string arr[], int low, int high) {
   string 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;
}
void quickSort(string arr[], int low, int high) {
   if (low < high) {
      int pi = partition(arr, low, high);
      quickSort(arr, low, pi - 1);
      quickSort(arr, pi + 1, high);
```

```
    }
}

int main() {
    int n;
    cout << "Enter number of books: ";
    cin >> n;
    cin.ignore();
    string books[n];
    cout << "Enter names of books:\n";
    for (int i = 0; i < n; i++) {
        getline(cin, books[i]);
    }
    quickSort(books, 0, n - 1);
    cout << "\nBooks in alphabetical order:\n";
    for (int i = 0; i < n; i++) {
        cout << books[i] << endl;
    }
    return 0;
}
```

Output:-



List Of Applications of Quick Sort:-

1. Sorting arrays or vectors
2. Implementing custom sorting logic using std::sort()
3. Sorting strings or names alphabetically
4. Sorting structures or classes (like students, books, employees) by a field
5. Database-like record sorting in memory
6. Sorting objects in gaming (by position, score, etc.)
7. Efficient searching (preparing data for binary search)
8. File data sorting (after reading data from files)
9. Sorting data for graph algorithms (like edges in Kruskal's algorithm)
10. Used internally in STL algorithms for performance

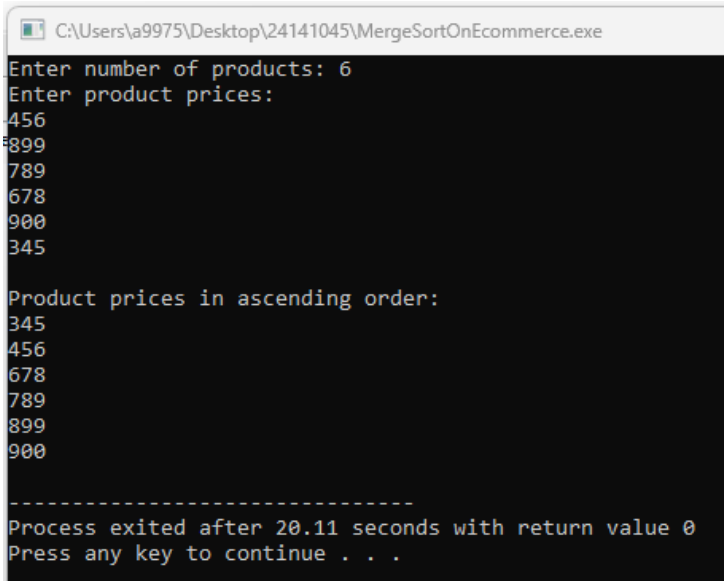
Program2:-

```
#include <iostream>
#include <string>
using namespace std;
void merge(double arr[], int left, int mid, int right) {
   int n1 = mid - left + 1;
   int n2 = right – mid;
   double L[n1], R[n2];
   for (int i = 0; i < n1; i++)
```

```
        L[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}
void mergeSort(double arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
```

```
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}
int main() {
    int n;
    cout << "Enter number of products: ";
    cin >> n;
    double price[n];
    cout << "Enter product prices:\n";
    for (int i = 0; i < n; i++) {
        cin >> price[i];
    }
    mergeSort(price, 0, n - 1);
    cout << "\nProduct prices in ascending order:\n";
    for (int i = 0; i < n; i++) {
        cout<< price[i] << endl;
    }
    return 0;
}
```

Output:-



List Of Applications of Merge Sort:-

1. Sorting linked lists efficiently
2. External sorting (for large files that don't fit in memory)
3. Sorting arrays or vectors in stable order
4. Implementing std::stable_sort() in C++ STL
5. Merging two sorted arrays or lists
6. Database sorting and merging operations
7. Inversion count problems (counting pairs where a[i] > a[j])
8. Used in divide and conquer algorithms (like counting inversions or sorting for median)
9. Sorting objects or structures where stability is required
10. Sorting data for parallel processing or multithreading