|  |  |
| --- | --- |
| **Assignment No:2** | |
| **Aim:** | Write a program in C++ to sort the numbers in an array using separate functions for reading, displaying, sorting, and swapping. |
| **Objective:** | Understand and apply modular programming principles by developing separate functions for reading input, displaying output, sorting the array, and swapping elements, thereby promoting code reusability and clarity.  To implement and analyze various sorting algorithms (such as Bubble Sort, Selection Sort, and Insertion Sort) to understand their time complexity, efficiency, and applicability in different scenarios, thereby enhancing problem-solving skills in algorithm design and data organization. |
| **Theory:** | Sorting is a fundamental operation in computer science, essential for organizing data efficiently. In this program, we aim to sort numbers in an array using C++ by breaking down the task into modular functions. This approach enhances code readability, maintainability, and reusability.  **Advantages of sorting in the context of data structures:**   1. **Improved Search Efficiency**:    * Sorted data allows for faster search operations, such as binary search, which significantly reduces the time complexity compared to linear search in unsorted data. 2. **Simplified Data Organization**:    * Sorting helps in organizing data in a structured manner, making it easier to manage, access, and analyze, particularly in large datasets. 3. **Optimized Algorithms**:    * Many algorithms, especially those for searching and merging, perform better on sorted data. For instance, algorithms like merge sort and quicksort rely on sorting to optimize their operations. 4. **Facilitates Efficient Data Management**:    * Sorting is essential in data structures like priority queues, where elements need to be accessed in a particular order (e.g., smallest or largest first). 5. **Enhances Data Integrity**:    * Sorted data ensures that duplicates or anomalies are more easily identified and corrected, maintaining data integrity across operations. 6. **Better Data Visualization**:    * Sorting data structures allows for more meaningful and intuitive data visualizations, making patterns and trends easier to detect. 7. **Preparation for Further Operations**:    * Many data processing tasks, such as merging datasets, identifying medians, and performing range queries, require sorted data for accurate and efficient execution. 8. **Improved Performance in Real-Time Systems**:    * In real-time systems, where quick decision-making is critical, sorted data structures enable faster access and manipulation of data, ensuring timely responses.   **Applications where sorting is used in data structures:**   * Real-time scheduling in operating systems * Prioritization in task management systems * Load balancing in network traffic management * Event handling in real-time systems * Data streaming and processing in multimedia applications * Dynamic pricing algorithms in e-commerce * Stock market analysis and trading algorithms * Sensor data processing in embedded systems * Route optimization in real-time navigation systems * Real-time leaderboard updates in gaming applications |
| **Algorithm:** | **Bubble Sort Algorithm**   1. **Repeat the following steps until no swaps are needed**:  * Start from the first element and compare it with the next one. * If the current element is larger than the next, swap them.  1. **Continue this process for each element** in the list, moving from the start to the end. 2. **With each pass through the list**, the largest unsorted element moves to its correct position at the end, gradually sorting the entire list. 3. **Start** 4. **Input the size of the array** n. 5. **Input the elements of the array**:    * For i = 0 to n-1, input arr[i]. 6. **Begin Bubble Sort**:    * For i = n-1 down to 1:      + For j = 0 to i-1:        - If arr[j] > arr[j+1], then:          * **Swap** arr[j] and arr[j+1]:   Temporarily store arr[j] in temp.  Set arr[j] to arr[j+1].  Set arr[j+1] to temp.   1. **End Bubble Sort** 2. **Display the sorted array**:    * For i = 0 to n-1, print arr[i]. 3. **End**  Selection Sort:  1. **Start from the first element of the list** and assume it is the minimum. 2. **Compare this minimum with each subsequent element** in the list to find the actual minimum element. 3. **Swap the found minimum element** with the first element, then repeat the process for the remaining unsorted portion of the list until the entire list is sorted. 4. **Start** 5. **Input the array** with size n. 6. **Begin Selection Sort**:    * For i = 0 to n-1 (where n is the size of the array):      + Set min\_index to i.      + For j = i+1 to n-1:        - If ar[j] < ar[min\_index], then:          * Set min\_index to j.      + If min\_index is not equal to i, then:        - **Swap** ar[i] and ar[min\_index]:          * Temporarily store ar[i] in temp.          * Set ar[i] to ar[min\_index].          * Set ar[min\_index] to temp. 7. **End Selection Sort** 8. **Display the sorted array**. 9. **End**  Insertion Sort Algorithm  1. **Start with the second element** of the list, considering the first element as the sorted portion. 2. **Compare the current element** with the elements in the sorted portion, moving larger elements one position to the right until the correct position for the current element is found. 3. **Insert the current element** into its correct position in the sorted portion and repeat the process for each subsequent element until the entire list is sorted. 4. **Start** 5. **Input the array** with size n. 6. **Begin Insertion Sort**:    * For i = 1 to n-1 (where n is the size of the array):      + Set temp to ar[i] (the current element to be inserted).      + Initialize j = i - 1 (the index of the last element in the sorted portion).      + While j >= 0 and ar[j] > temp:        - Shift ar[j] to the right:          * Set ar[j + 1] to ar[j].          * Decrement j by 1.      + Insert temp at position j + 1:        - Set ar[j + 1] to temp. 7. **End Insertion Sort** 8. **Display the sorted array**. 9. **End** |
| **Program:** | **Bubble Sort:**  #include <iostream>  using namespace std;  void readArray(int arr[], int n) {  cout << "Enter " << n << " elements: ";  for (int i = 0; i < n; i++) {  cin >> arr[i];  }  }  void displayArray(int arr[], int n) {  cout << "Array elements: ";  for (int i = 0; i < n; i++) {  cout << arr[i] << " ";  }  cout << endl;  }  void swap(int &a, int &b) {  int temp = a;  a = b;  b = temp;  }  void bubbleSort(int arr[], int n) {  for (int i = n - 1; i > 0; i--) {  for (int j = 0; j < i; j++) {  if (arr[j] > arr[j + 1]) {  swap(arr[j], arr[j + 1]);  }  }  }  }  int main() {  int n;  cout << "Enter the number of elements: ";  cin >> n;  int arr[n];  readArray(arr, n);  bubbleSort(arr, n);  displayArray(arr, n);  return 0;  } Selection Sort: #include <iostream>  using namespace std;  void readArray(int arr[], int n) {  cout << "Enter " << n << " elements: ";  for (int i = 0; i < n; i++) {  cin >> arr[i];  }  }  void displayArray(int arr[], int n) {  cout << "Array elements: ";  for (int i = 0; i < n; i++) {  cout << arr[i] << " ";  }  cout << endl;  }  void swap(int &a, int &b) {  int temp = a;  a = b;  b = temp;  }  void selectionSort(int arr[], int n) {  for (int i = 0; i < n - 1; i++) {  int min\_index = i;    for (int j = i + 1; j < n; j++) {  if (arr[j] < arr[min\_index]) {  min\_index = j;  }  }    if (min\_index != i) {  swap(arr[i], arr[min\_index]);  }  }  }  int main() {  int n;  cout << "Enter the number of elements: ";  cin >> n;  int arr[n];    readArray(arr, n);  selectionSort(arr, n);  displayArray(arr, n);  return 0;  }  **Insertion Sort:**  #include <iostream>  using namespace std;  void readArray(int arr[], int n) {  cout << "Enter " << n << " elements: ";  for (int i = 0; i < n; i++) {  cin >> arr[i];  }  }  void displayArray(int arr[], int n) {  cout << "Array elements: ";  for (int i = 0; i < n; i++) {  cout << arr[i] << " ";  }  cout << endl;  }  void insertionSort(int arr[], int n) {  for (int i = 1; i < n; i++) {  int temp = arr[i];  int j = i - 1;  while (j >= 0 && arr[j] > temp) {  arr[j + 1] = arr[j];  j--;  }  arr[j + 1] = temp;  }  }  int main() {  int n;  cout << "Enter the number of elements: ";  cin >> n;  int arr[n];  readArray(arr, n);  insertionSort(arr, n);  displayArray(arr, n);  return 0;  } |
| **Output:** | **Bubble Sort Output:**  Enter the number of elements: 7  Enter 7 elements: 64 34 25 12 22 11 90  Array elements: 11 12 22 25 34 64 90 Selection Sort Output: Enter the number of elements: 5  Enter 5 elements: 88 56 34 12 90  Array elements: 12 34 56 88 90  **Insertion Sort Output:**  Enter the number of elements: 6  Enter 6 elements: 45 23 89 10 67 32  Array elements: 10 23 32 45 67 89 |
| **Conclusion:** | |
| **Date:** | |
| **Staff Sign:** | |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***END**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |