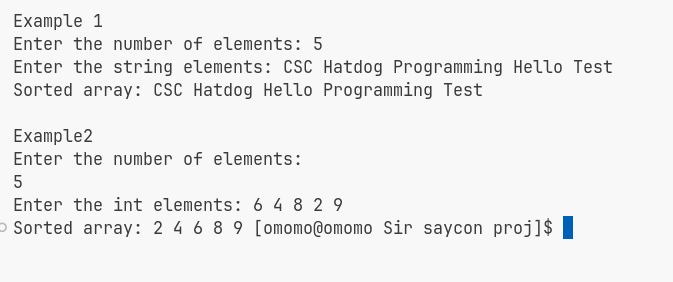
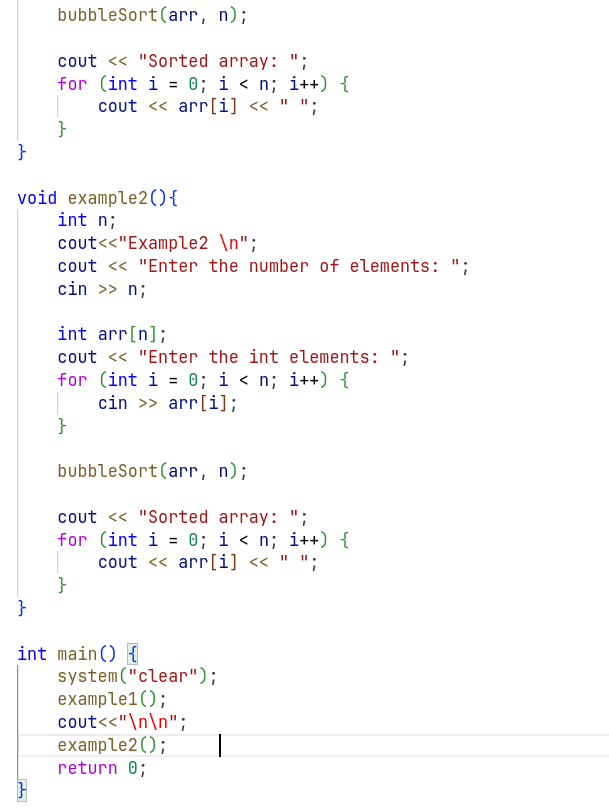
**Bubble Sort:**



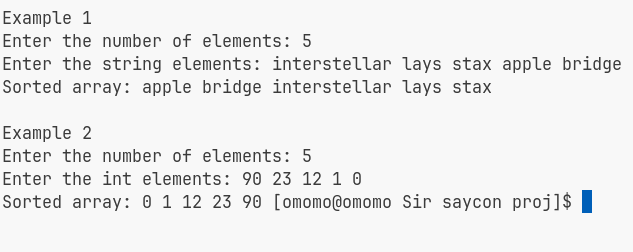
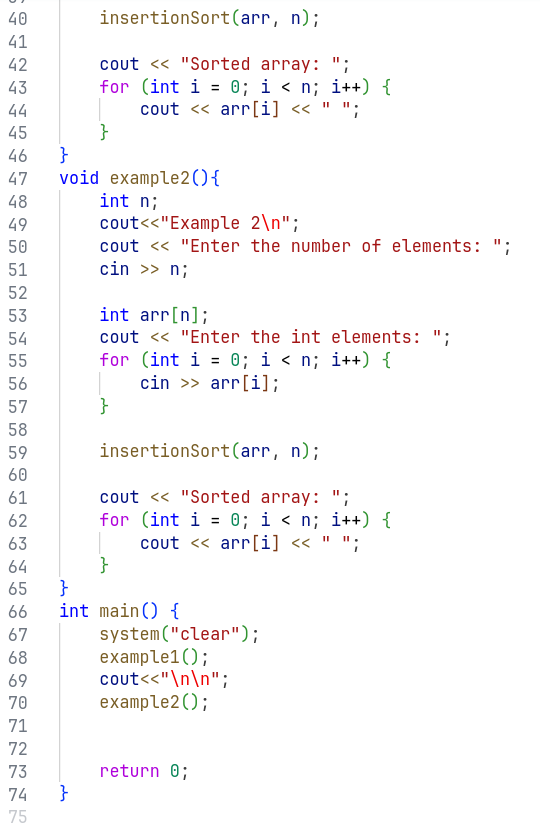
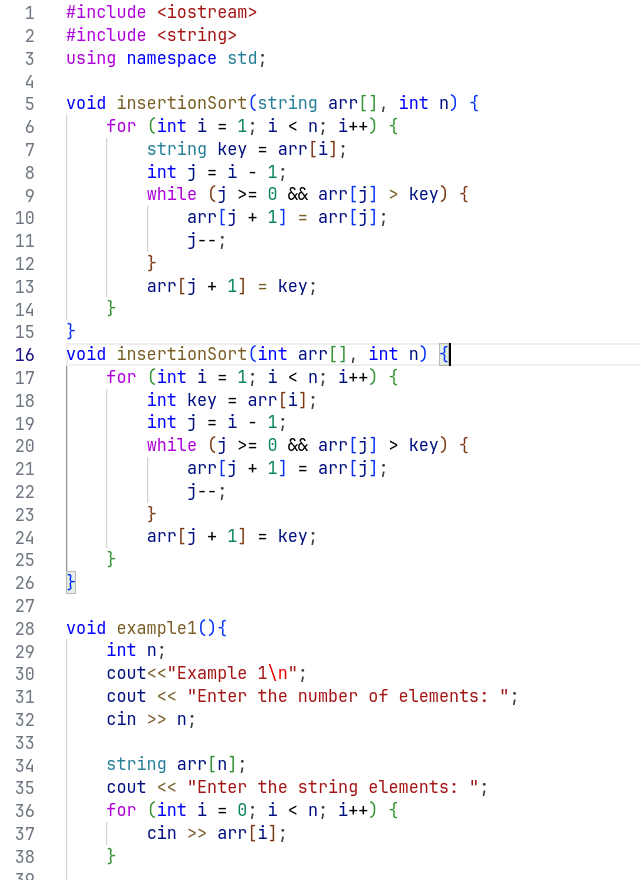
* Bubble Sort is a simple sorting algorithm that repeatedly compares adjacent elements and swaps them if they are in the wrong order.
* It is easy to understand and implement, but it is not efficient for large input sizes.



**Insertion Sort:**



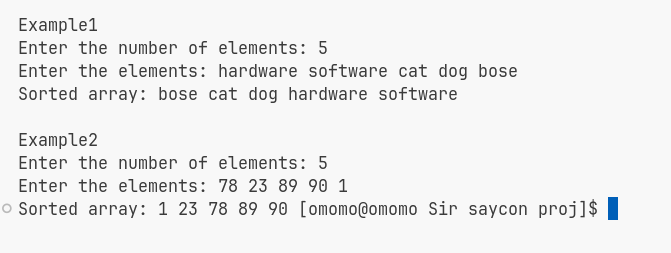
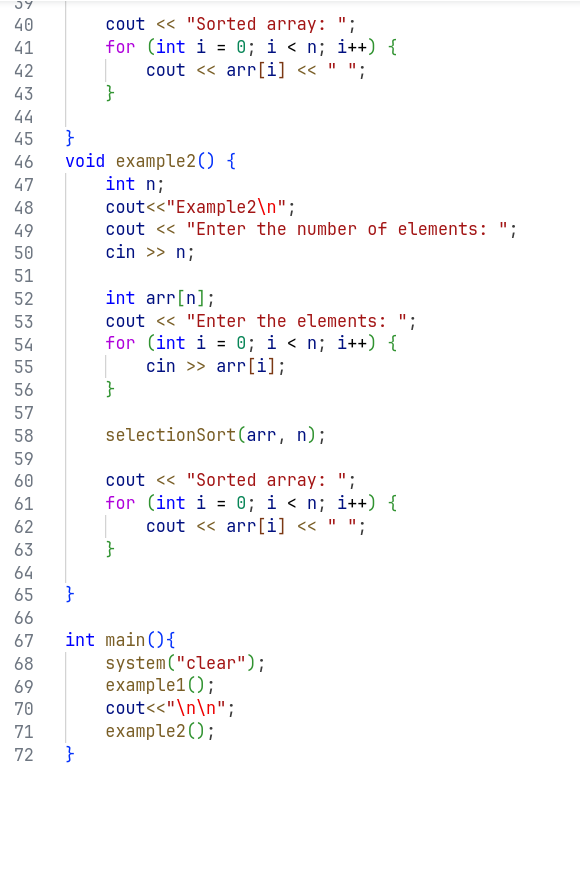
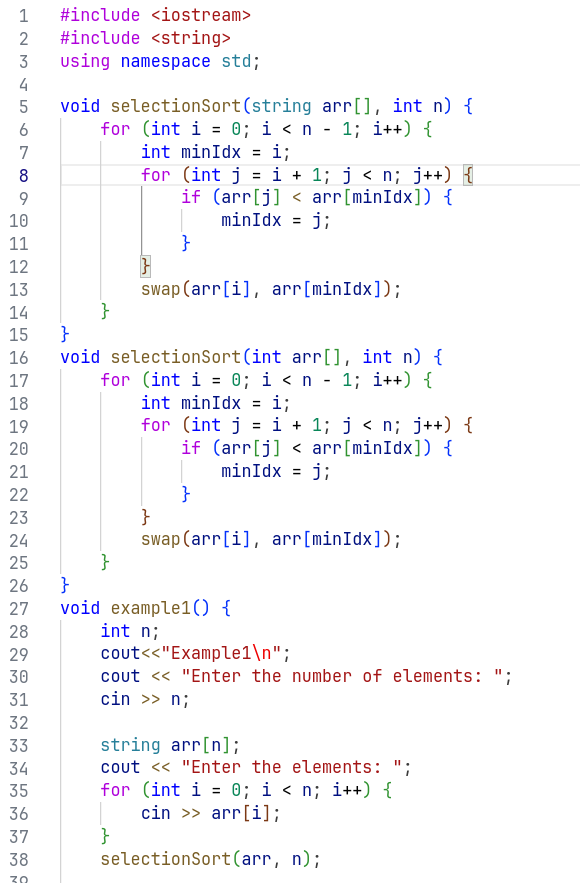
* Insertion Sort is a comparison-based sorting algorithm that builds the final sorted array one element at a time.
* It iterates over the array, comparing each element with the elements before it and inserting it in the correct position.
* Insertion Sort performs well on small input sizes and partially sorted arrays.



**Selection Sort:**



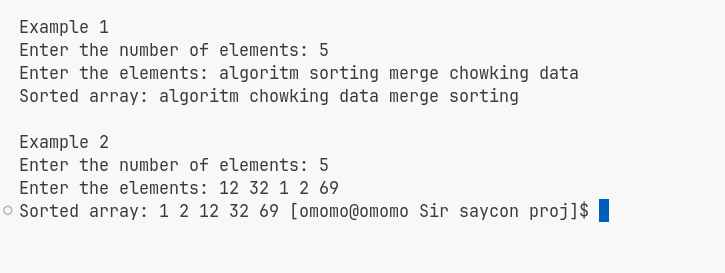
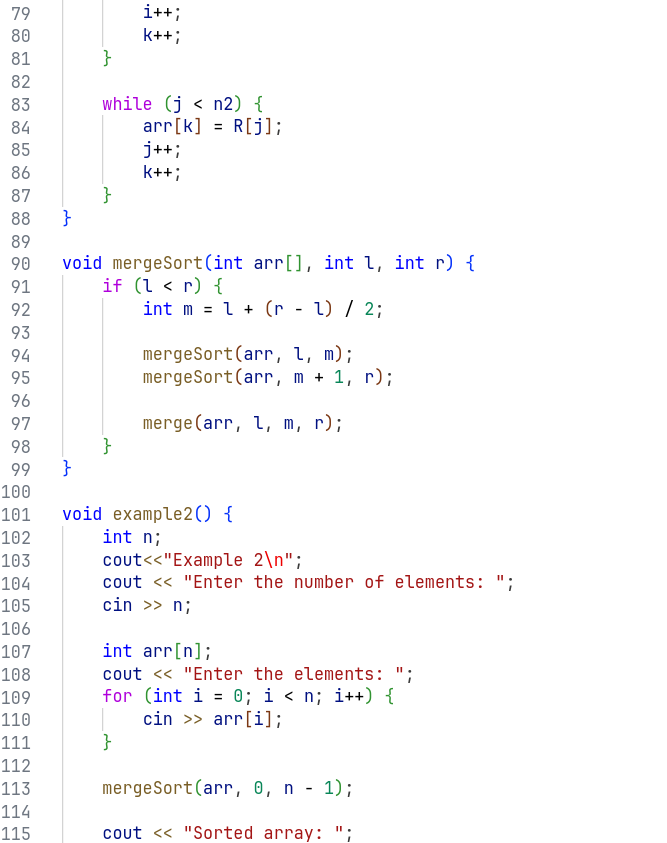
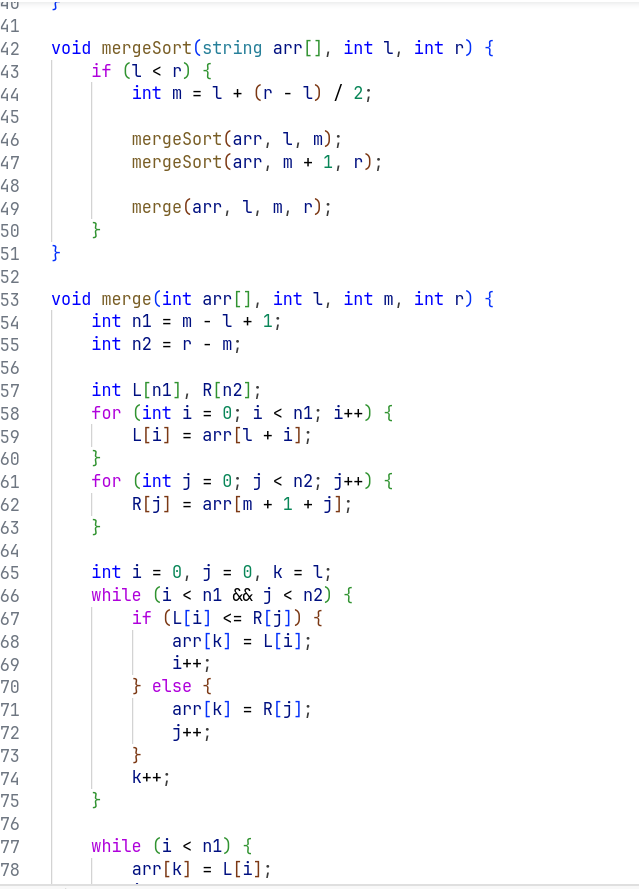
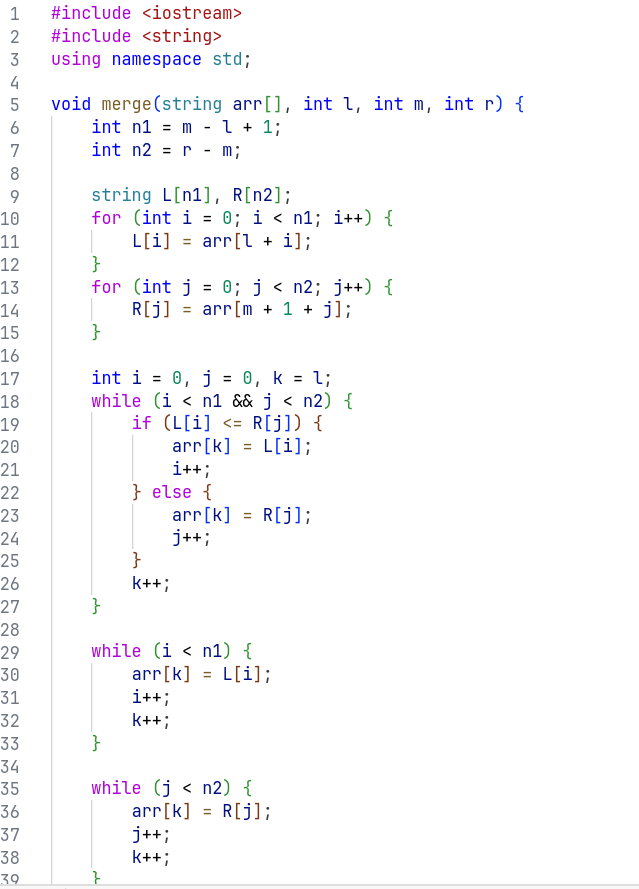
* Selection Sort is an in-place comparison-based sorting algorithm.
* It divides the input array into a sorted and an unsorted region.
* In each iteration, it finds the minimum (or maximum) element from the unsorted region and swaps it with the first element of the unsorted region.



**Merge Sort:**



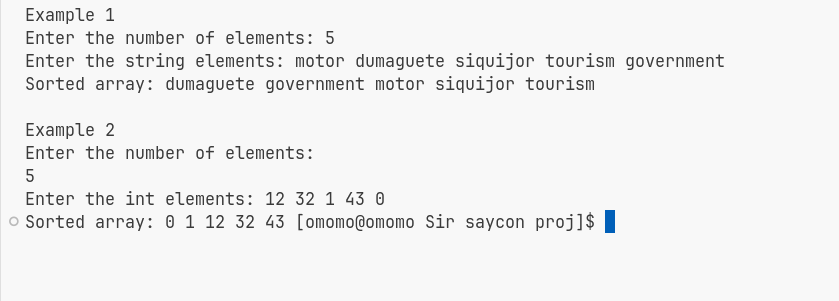
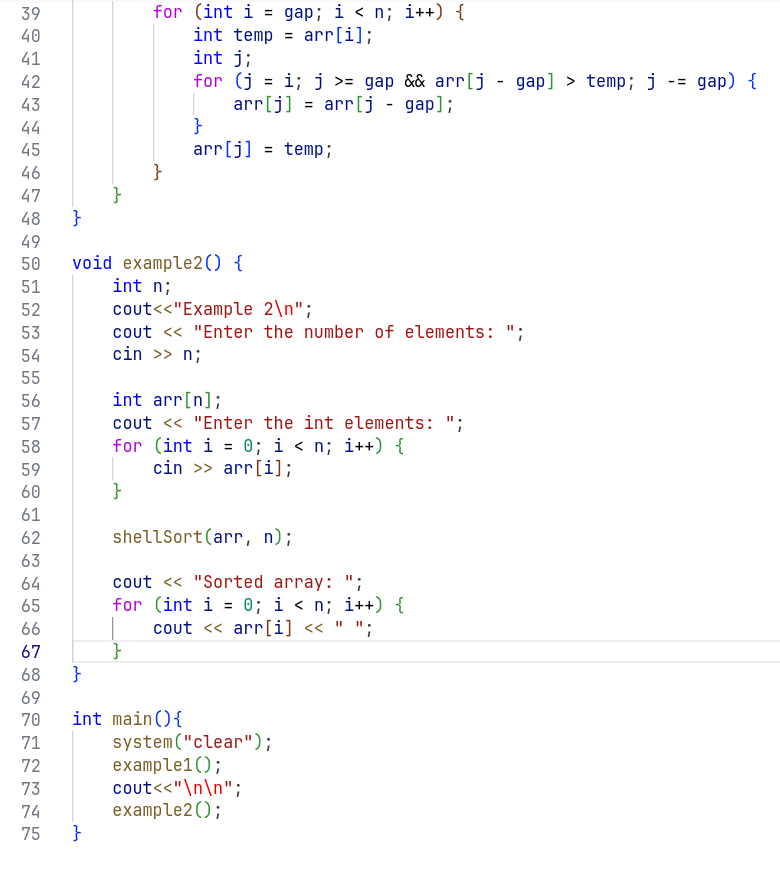
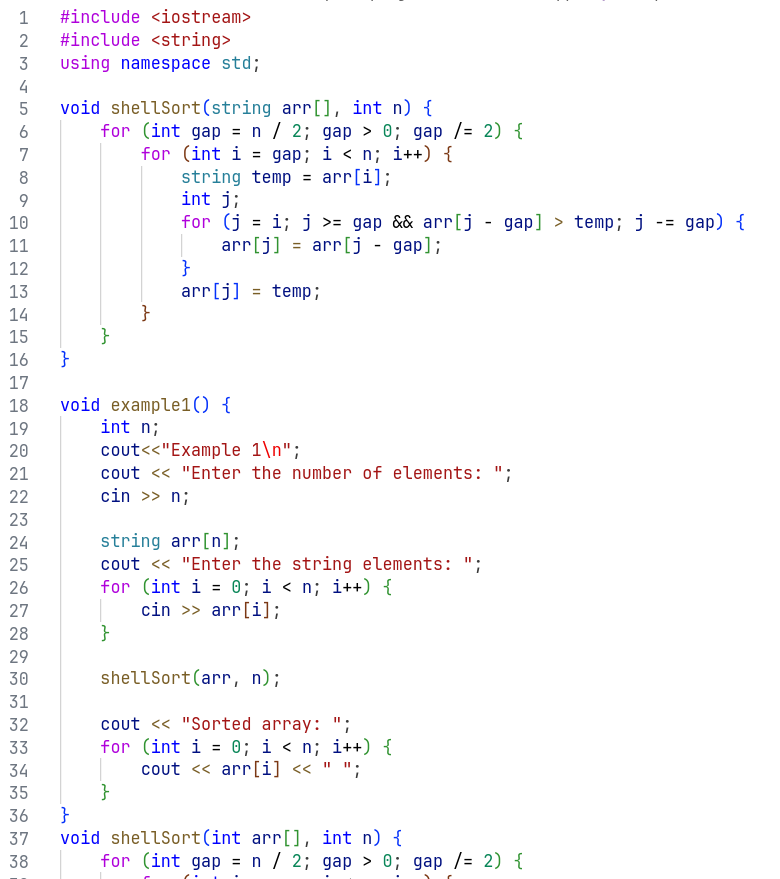
* Merge Sort is a divide-and-conquer algorithm that divides the input array into two halves,
* recursively sorts them, and then merges them to produce a sorted output.
* Merge Sort requires additional space for merging the subarrays, so it may not be suitable for memory-constrained environments.



**Shell Sort:**



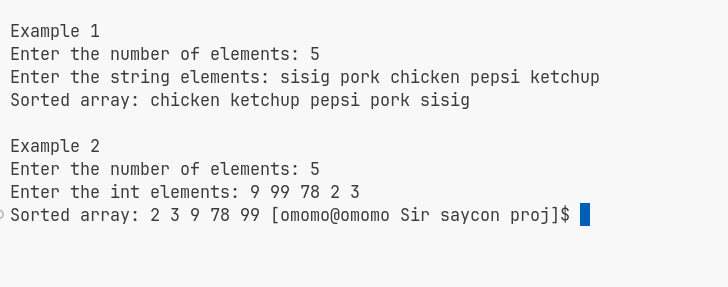
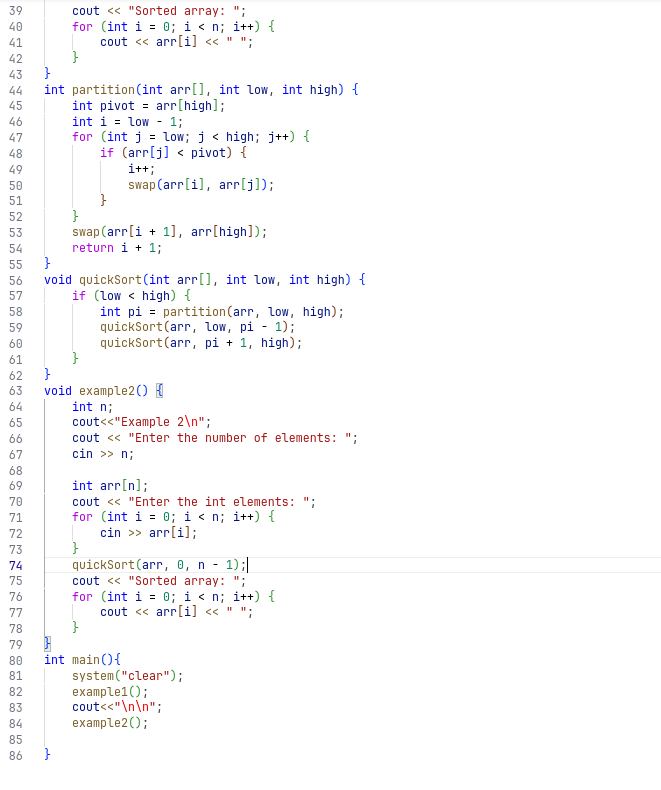
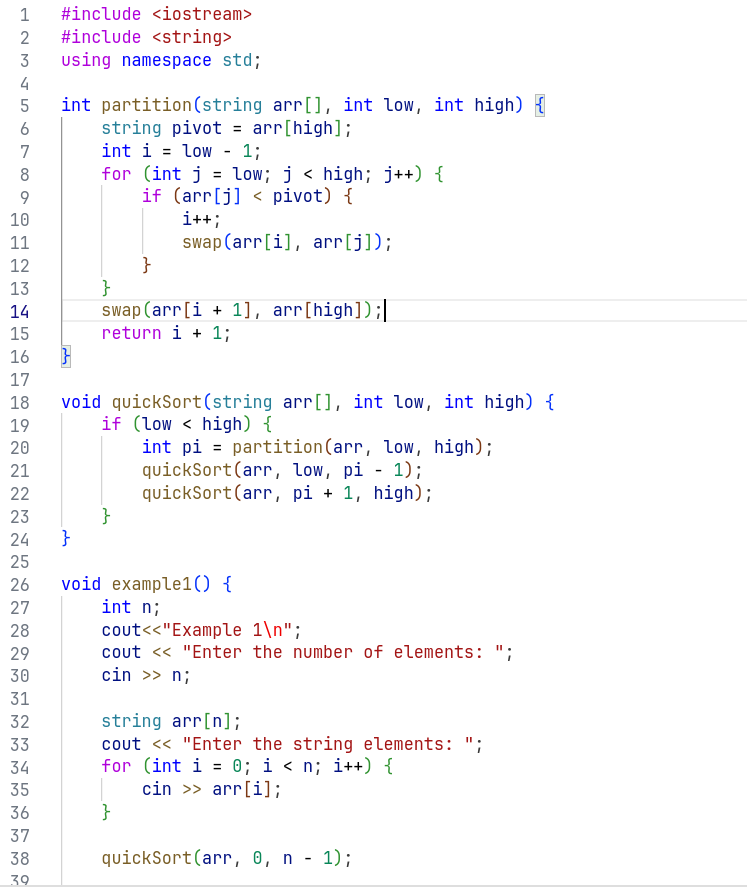
* Shell Sort is an optimization of Insertion Sort that allows elements to move in larger steps.
* It starts by sorting pairs of elements that are far apart and gradually reduces the gap between elements to be compared.
* It performs better than Insertion Sort but is not as efficient as more advanced algorithms like Quick Sort or Merge Sort.



**Quick Sort:**



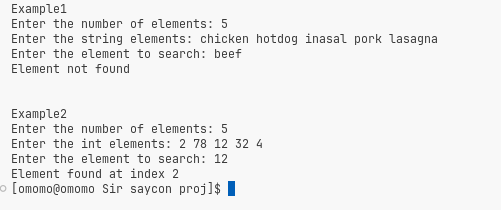
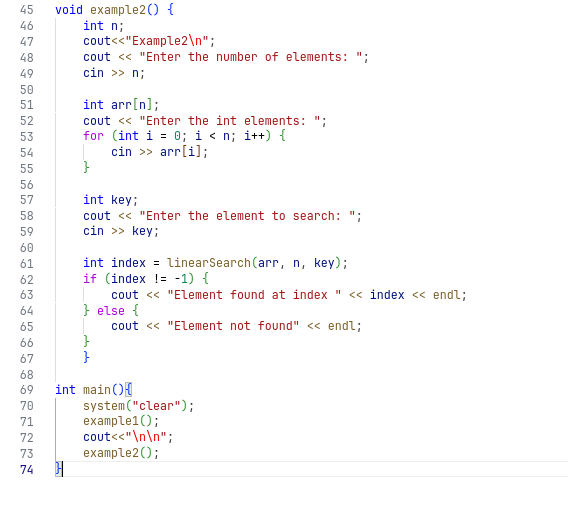
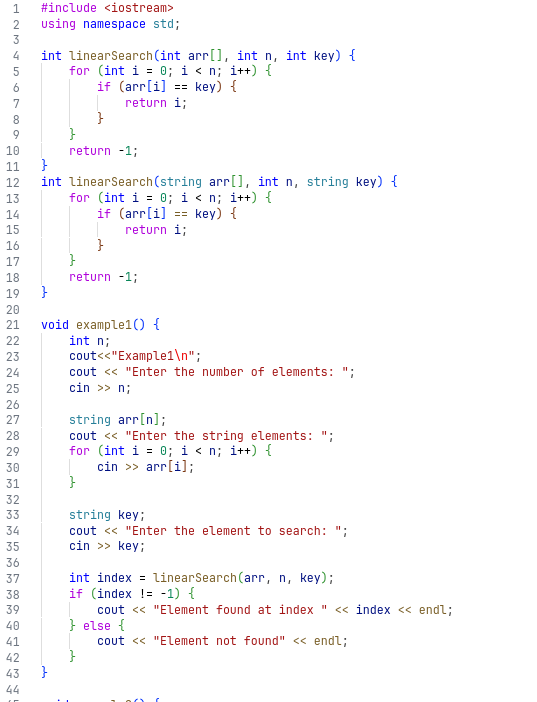
* Quick Sort is a widely used comparison-based sorting algorithm that follows the divide-and-conquer approach.
* It selects a pivot element and partitions the array around the pivot, placing smaller elements before it and larger elements after it.
* Quick Sort then recursively applies the same process to the subarrays.



**Linear Search:**



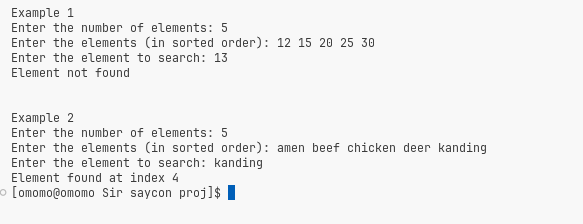
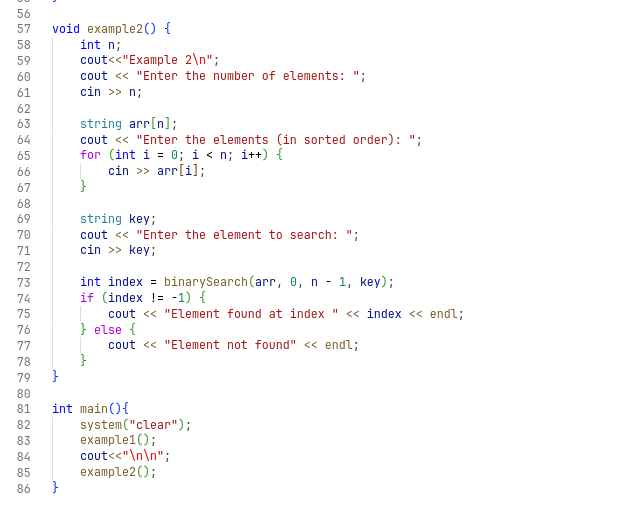
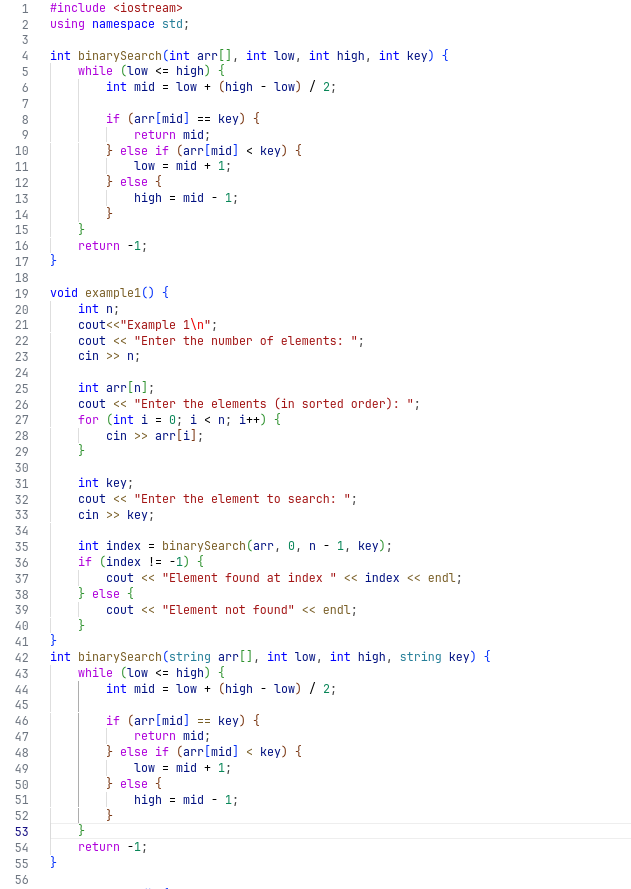
* Linear Search is a simple searching algorithm that sequentially checks each element in a list until a match is found or the end of the list is reached.
* It is suitable for small lists or unsorted data but can be inefficient for large lists.



**Binary Search:**



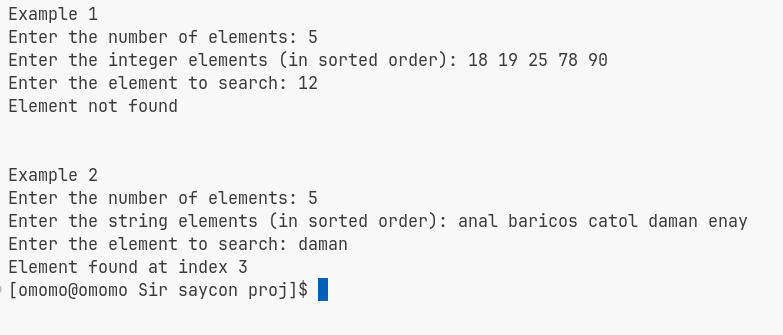
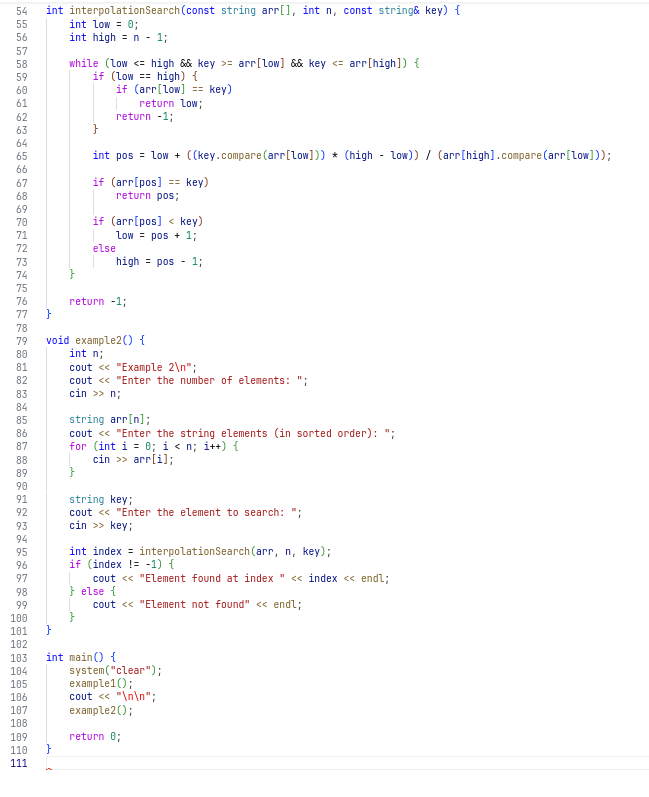
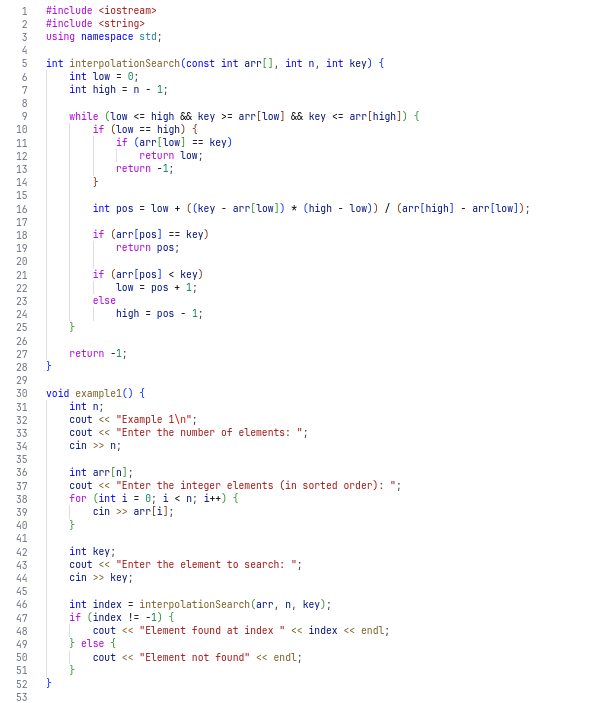
* Binary Search is an efficient searching algorithm for sorted lists.
* It compares the target value with the middle element of the sorted list and eliminates half of the remaining elements based on the comparison.
* Binary Search continues dividing the search space in half until the target value is found or the search space is empty.



**Interpolation Search:**



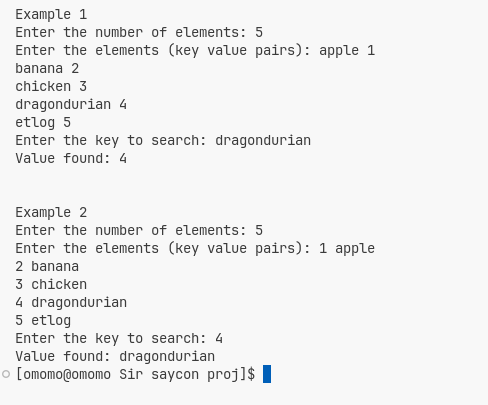
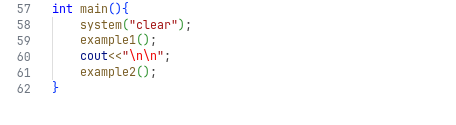
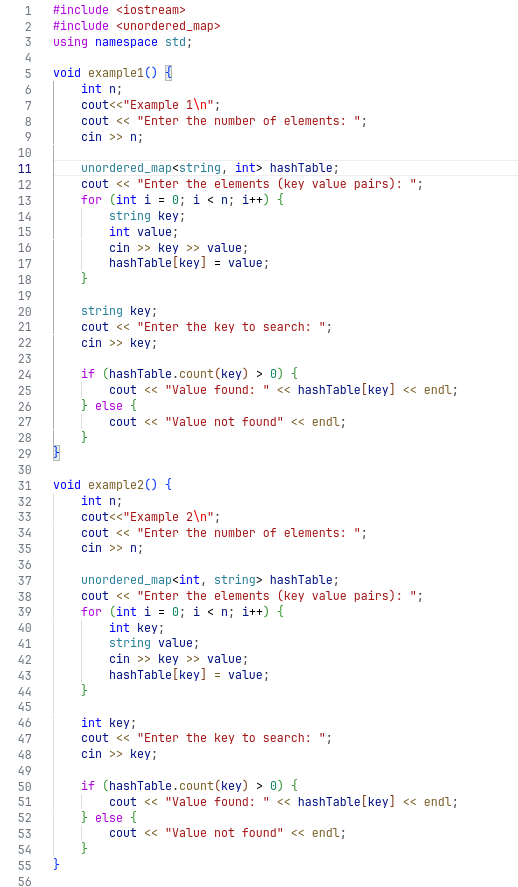
* Interpolation Search is an improved variant of Binary Search.
* It uses the distribution of values in a sorted list to make intelligent guesses about the location of the target value.
* Interpolation Search can perform better than Binary Search for uniformly distributed data



**Hash Table:**



* A Hash Table is a data structure that uses a hash function to map keys to values.
* It provides efficient insertion, deletion, and retrieval operations.
* Hash Tables have an average-case time complexity of O(1) for these operations.
* However, collisions (multiple keys mapping to the same index) can occur, leading to performance degradation and a worst-case time complexity of O(n).
* Hash Tables are widely used when fast access to data is required, and they are particularly effective for dictionary-like data structures.



**Data Structures  
and   
Algorithm**

**Submitted by:   
Tyler Christian D. Opo  
  
Submitted to:  
Celester L. Saycon**