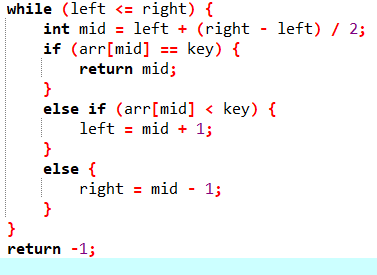
**Lab 8 Exercise**

**Task-1:**

Suppose you are given a list of student records, where each record contains a student's name and their score on a recent test. You need to sort this list by score in ascending order using **Radix Sort**, and represent the list using a linked list in C++.

Design and implement a **Radix Sort algorithm** using a linked list to sort the student records. Assume that each student's score is an integer between 0 and 100, and that there are no more than 10,000 students in the list. After that asks the user to enter the name and score if that record is available in the list, then delete that record and display the updated list. Otherwise do nothing. Note use binary search to search the record in the linked list.

**Input :**

addNode(&head, “Ayan” , 90);

addNode(&head, “Zameer” , 60);

addNode(&head, “Sara” , 70);

addNode(&head, “Sohail” , 30);

addNode(&head, “Ahmed” , 20);

**Output:**

Ahmed 20

Sohail 30

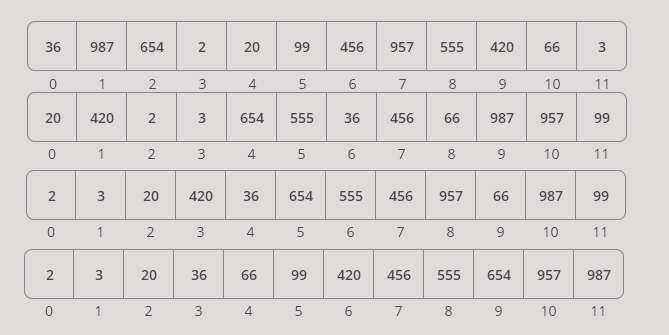
Zameer 60

Sara 70

Ayan 90

**Task-2:**

Given an array in the figure of integers, sort the array in ascending as well as descending order and return it using **radix sort**



**Steps of Radix Sort:**

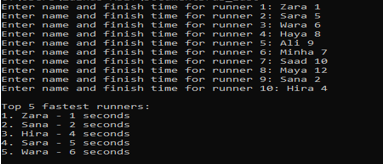
* Find the maximum number in the array to know the number of digits.
* Sort based on each digit, starting from the least significant digit (LSD) to the most significant digit (MSD), using a stable sorting algorithm (such as counting sort).
* Repeat the sorting process for each digit position until all digits are processed.

**Task-3:**

Suppose you are working for a company that organizes a marathon every year. There are 10 participants in the marathon, and their finish times are recorded in seconds.

Write a program in C++ that uses **Merge sort** to find the top 5 fastest runners. Create an array of runner objects that will store the participant's name and finish time in seconds.

* Prompt the user to input the participant's names and finish times into the array.
* Implement a merge function that will merge two subarrays of runners based on their finish time in ascending order.
* Use the Merge sort algorithm to sort the runners based on their finish time by calling the merge function in step 3.
* Display the top 5 fastest runners to the console, along with their name and finish time in seconds.
* Use the appropriate data types and size for the arrays and variables to ensure efficient memory usage.



**Steps of Merge Sort:**

* Divide: Split the array into two halves until each sub-array contains a single element.
* Conquer (Sort): Recursively sort each half.
* Merge: Merge the two sorted halves into a single sorted array.

**Task-4:**

Imagine you are working for a company that provides an online shopping platform. There are only 3 products listed on the platform, and customers often search for products based on their price. Write a program in C++ that uses **Quick sort** to sort the products based on their price in ascending order. Define a class for each product that will store its name, price, description, and availability?

A screenshot of a computer

Description automatically generated

**Steps of Quick Sort:**

* Pivot Selection: Choose a pivot element.
* Partitioning: Rearrange the array such that all elements smaller than the pivot are on the left, and all greater elements are on the right.
* Recursion: Recursively apply the quick sort on the left and right halves of the array.

The efficiency of Quick Sort depends on the pivot selection strategy.

**Task 5:**

Implement **Quick Sort** using linked list in C++. Define a node for a linked list with integer data and a pointer to the next node. Write a function to swap two nodes in the linked list. Write a function to partition the linked list using the last node as the pivot. Implement the Quick Sort algorithm using the partition function and the swap function. Finally, write a function to print the sorted linked list. Your implementation should be able to handle empty linked lists and linked lists with only one node. Test your implementation with a sample linked list containing the following integers: 10, 7, 8, 9, 1, 5, 3. The expected output after sorting should be: 1 3 5 7 8 9 10.

**Task 6:**

You have been given two unsorted arrays, arr1 and arr2, each containing 10 integers. You need to combine the two arrays into a single sorted array using either Radix Sort or Merge Sort.

* Write a C++ program to combine arr1 and arr2 using Radix Sort and print the sorted array.
* Write a C++ program to combine arr1 and arr2 using Merge Sort and print the sorted array.

**Task 7:**

In the **quick sort** algorithm, the selection of a pivot element is of utmost importance. There are different ways to select the pivot element. Your task is to implement quick sort using the following pivot selection techniques and count the number of comparisons of each approach:

Note: The implementation should be in-place.

* Select the first element as pivot.
* Select any random element as pivot
* Select middle element as pivot
* Use median approach for pivot selection
* Apply quick sort on list (Use any pivot selection approach)

After applying all the above approaches, evaluate the efficiency of each pivot selection technique, we count the number of comparisons made during partitioning.

**Note:** Apply the **quick sort** on different input arrays (e.g., sorted, reverse-sorted, random) to see how each pivot strategy performs.