SVKM's NMIMS
Mukesh Patel School of Technology Management & Engineering
Computer Engineering Department
Program: B.Tech Integrated- Sem V

Course: Basics of Data Structures

LAB Manual

PART A

(PART A : TO BE REFFERED BY STUDENTS)

Experiment No.10

A.1 Aim:

Task1: Implementation of Sequential Search

Task2: Implementation of Binary Search

Task3: Implementation of Bubble Sort

Task4: Implementation of Selection Sort

Task5: Implementation of Insertion sort

A.2 Prerequisite:

2. Knowledge of array, pointers

A.3 Outcome:

After successful completion of this experiment students will be able to

Design & develop a searching program.

A.4 Theory:
A.4.1.

Linear Search/Sequential Search

Linear search is also called as sequential search algorithm. It is the simplest searching algorithm. In Linear search, we simply traverse the list completely and match each element of the list with the item whose location is to be found. If the match is found, then the location of the item is returned; otherwise, the algorithm returns NULL.

It is widely used to search an element from the unordered list, i.e., the list in which items are not sorted.

The steps used in the implementation of Linear Search are listed as follows -

1. First, we have to traverse the array elements using a for loop.
2. In each iteration of for loop, compare the search element with the current array element, and -
    a. If the element matches, then return the index of the corresponding array element.
    b. If the element does not match, then move to the next element.
3. If there is no match or the search element is not present in the given array, return -1.

Binary Search

Binary Search Algorithm can be implemented in two ways which are discussed below.

- Iterative Method
- Recursive Method

Iteration Method

```
do until the pointers low and high meet each other.
   mid = (low + high)/2
   if (x == arr[mid])
      return mid
   else if (x > arr[mid]) // x is on the right side
      low = mid + 1
   else                   // x is on the left side
      high = mid - 1
```

Recursive Method

```
binarySearch(arr, x, low, high)
    if low > high
        return False
    else
        mid = (low + high) / 2
        if x == arr[mid]
            return mid
        else if x > arr[mid]        // x is on the right side
            return binarySearch(arr, x, mid + 1, high)
        else                        // x is on the right side
            return binarySearch(arr, x, low, mid - 1)
```

Bubble Sort

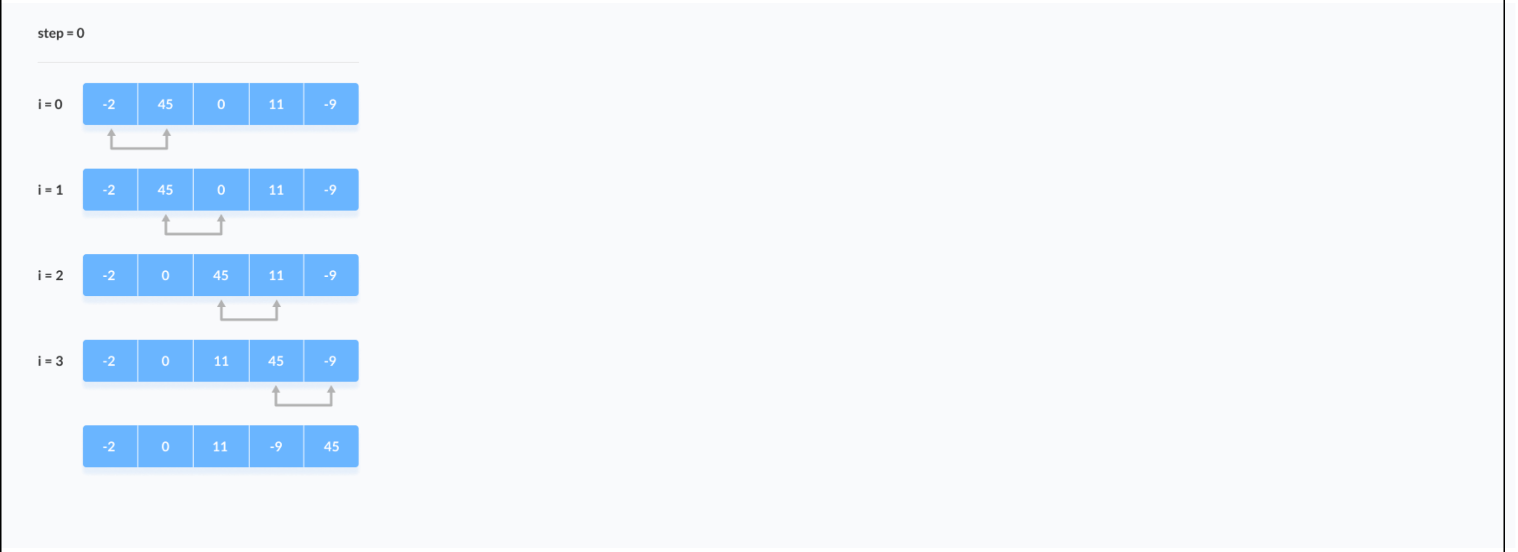
Bubble sort is a sorting algorithm that compares two adjacent elements and swaps them until they are in the intended order.

Working of Bubble Sort

Suppose we are trying to sort the elements in ascending order.

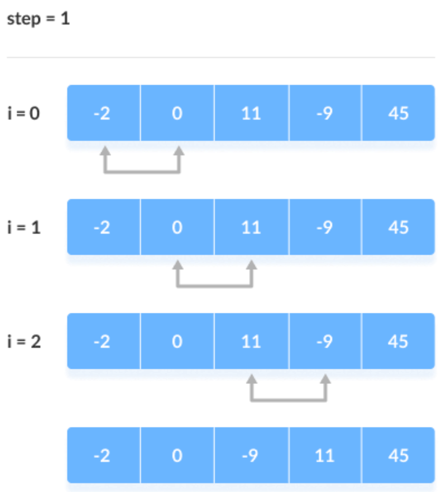
First Iteration (Compare and Swap)

1. starting from the first index, compare the first and the second elements.
2. If the first element is greater than the second element, they are swapped.
3. Now, compare the second and the third elements. Swap them if they are not in order.



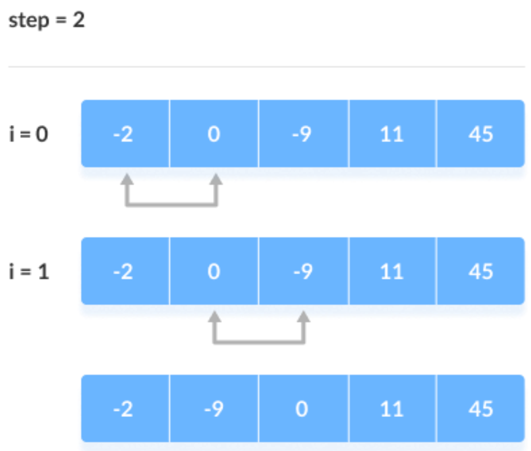
The above process goes on until the last element. Compare the Adjacent Elements

Remaining Iteration: The same process goes on for the remaining iterations. After each iteration, the largest element among the unsorted elements is placed at the end.

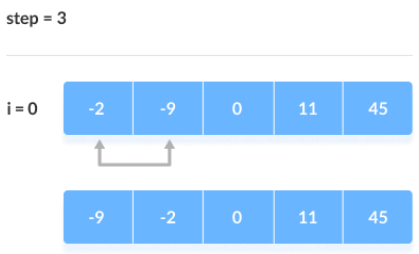


Put the largest element at the end

In each iteration, the comparison takes place up to the last unsorted element.



Compare the adjacent elements. The array is sorted when all the unsorted elements are placed at their correct positions.



The array is sorted if all elements are kept in the right order

Bubble Sort Algorithm

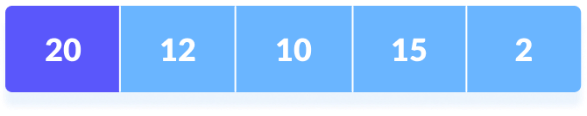
```
bubbleSort(array)
  for i <- 1 to indexOfLastUnsortedElement-1
    if leftElement > rightElement
      swap leftElement and rightElement
end bubbleSort
```

Selection sort

Selection sort is a sorting algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.

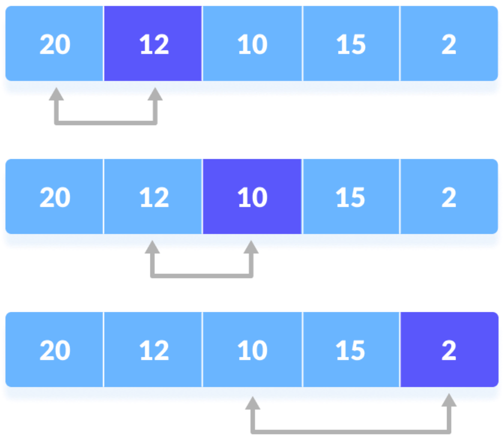
Working of Selection Sort

1. Set the first element as minimum.

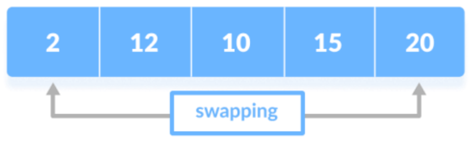


2. Compare minimum with the second element. If the second element is smaller than minimum, assign the second element as minimum.

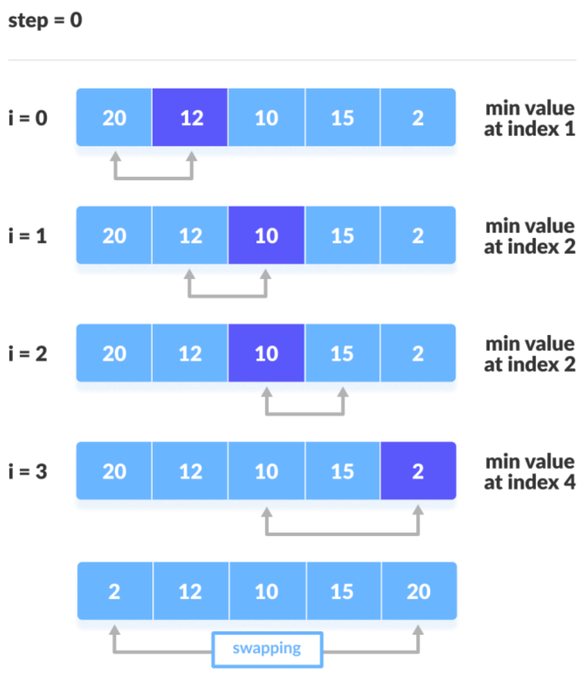
   Compare minimum with the third element. Again, if the third element is smaller, then assign minimum to the third element otherwise do nothing. The process goes on until the last element.

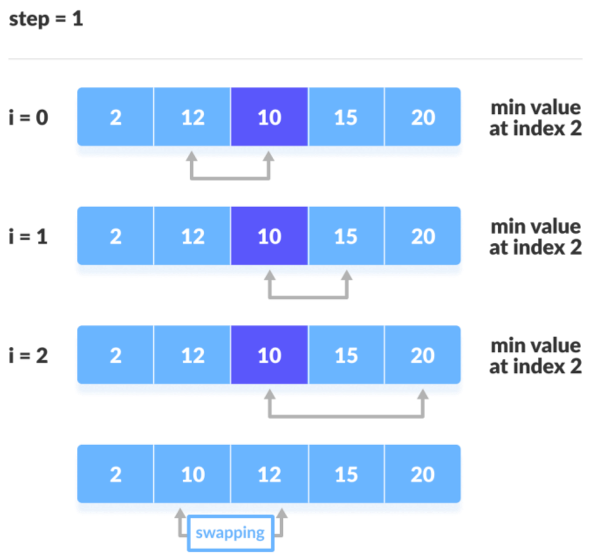


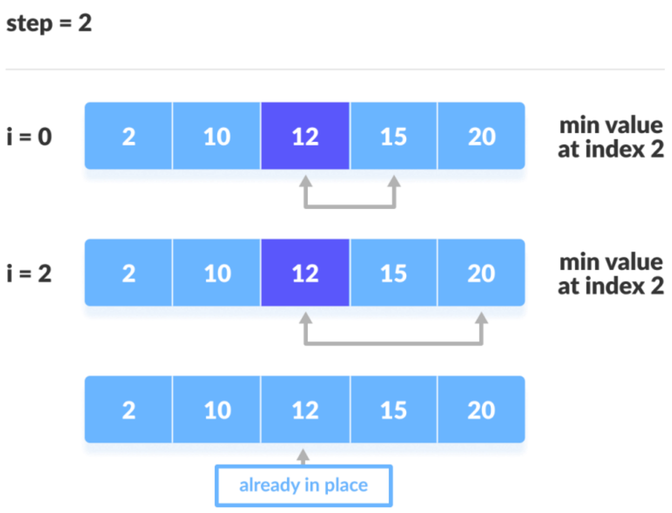
3. After each iteration, minimum is placed in the front of the unsorted list

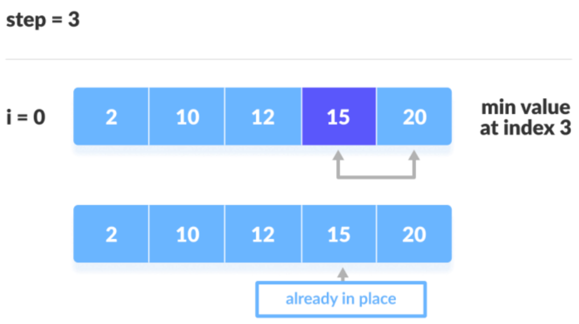


4. For each iteration, indexing starts from the first unsorted element. Step 1 to 3 are repeated until all the elements are placed at their correct positions.









Selection Sort Algorithm

```
selectionSort(array, size)
  repeat (size - 1) times
  set the first unsorted element as the minimum
  for each of the unsorted elements
    if element < currentMinimum
      set element as new minimum
  swap minimum with first unsorted position
end selectionSort
```

****

****

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Portal.)

Program:	Sem:
Roll No.	Name:
Division:	Batch :
Date of Experiment:	Date of Submission:
Grade :	

B.1 Software Code written by student:

(Paste your code completed during the 2 hours of practical in the lab here)

B.2 Input and Output:

(Paste your commented program input and output in following format, If there is error then paste the specific error in the output part. In case of error with due permission of the faculty extension can be given to submit the error free code with output in due course of time.)

B.3 Observations and learning:

(Students are expected to comment on the output obtained with clear observations and learning for each task/ sub part assigned)

B.4 Conclusion:

(Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)

B.5 Question of Curiosity

(To be answered by student based on the practical performed and learning/observations)

Identify the applications of searching Techniques.

****