SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE NAME : 23ITT101-PROBLEMM SOLVING AND C PROGRAMMING

I YEAR /I SEMESTER

Unit 3- Arrays and Strings

Topic 8:Bubble sort and Selection sort.

Brain Storming

1. How 2D array is created and accessed?

Bubble sort

- Bubble sort is a simple sorting algorithm.
- This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order.
- This algorithm is not suitable for large data sets.

BUBBLE SORT

Bubble sort starts with very first two elements, comparing them to check which one is greater.

In this case, value 33 is greater than 14, so it is already in sorted locations. Next, we compare 33 with 27.

We find that 27 is smaller than 33 and these two values must be swapped.

The new array should look like this –

Conti...

Next we compare 33 and 35. We find that both are in already sorted positions.

Then we move to the next two values, 35 and 10.

We know then that 10 is smaller 35. Hence they are not sorted.

We swap these values. We find that we have reached the end of the array. After one iteration, the array should look like this −

To be precise, we are now showing how an array should look like after each iteration. After the second iteration, it should look like this −

Notice that after each iteration, at least one value moves at the end.

Conti...

And when there's no swap required, bubble sorts learns that an array is completely sorted.

Now we should look into some practical aspects of bubble sort.

Example

```c
/* Bubble sort code */

#include <stdio.h>

int main()
{
  int array[100], n, c, d, swap;

  printf("Enter number of elements\n");
  scanf("%d", &n);

  printf("Enter %d integers\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);

  for (c = 0 ; c < n - 1; c++)
  {
    for (d = 0 ; d < n - c - 1; d++)
    {
      if (array[d] > array[d+1]) /* For decreasing order use '<' instead of '>' */
      {
        swap       = array[d];
        array[d]   = array[d+1];
        array[d+1] = swap;
      }
    }
  }
```

Conti..

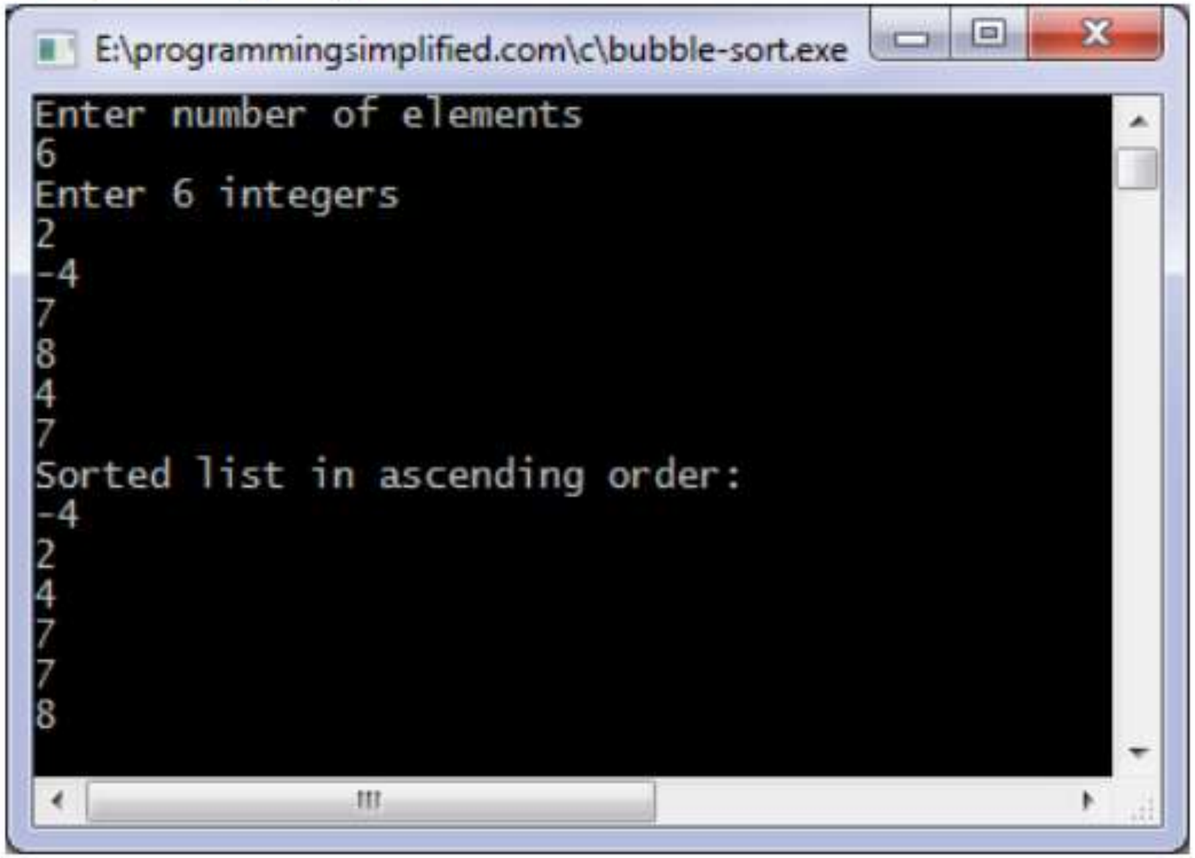
```
   printf("Sorted list in ascending order:\n");

   for (c = 0; c < n; c++)
      printf("%d\n", array[c]);

   return 0;
}
```

Output

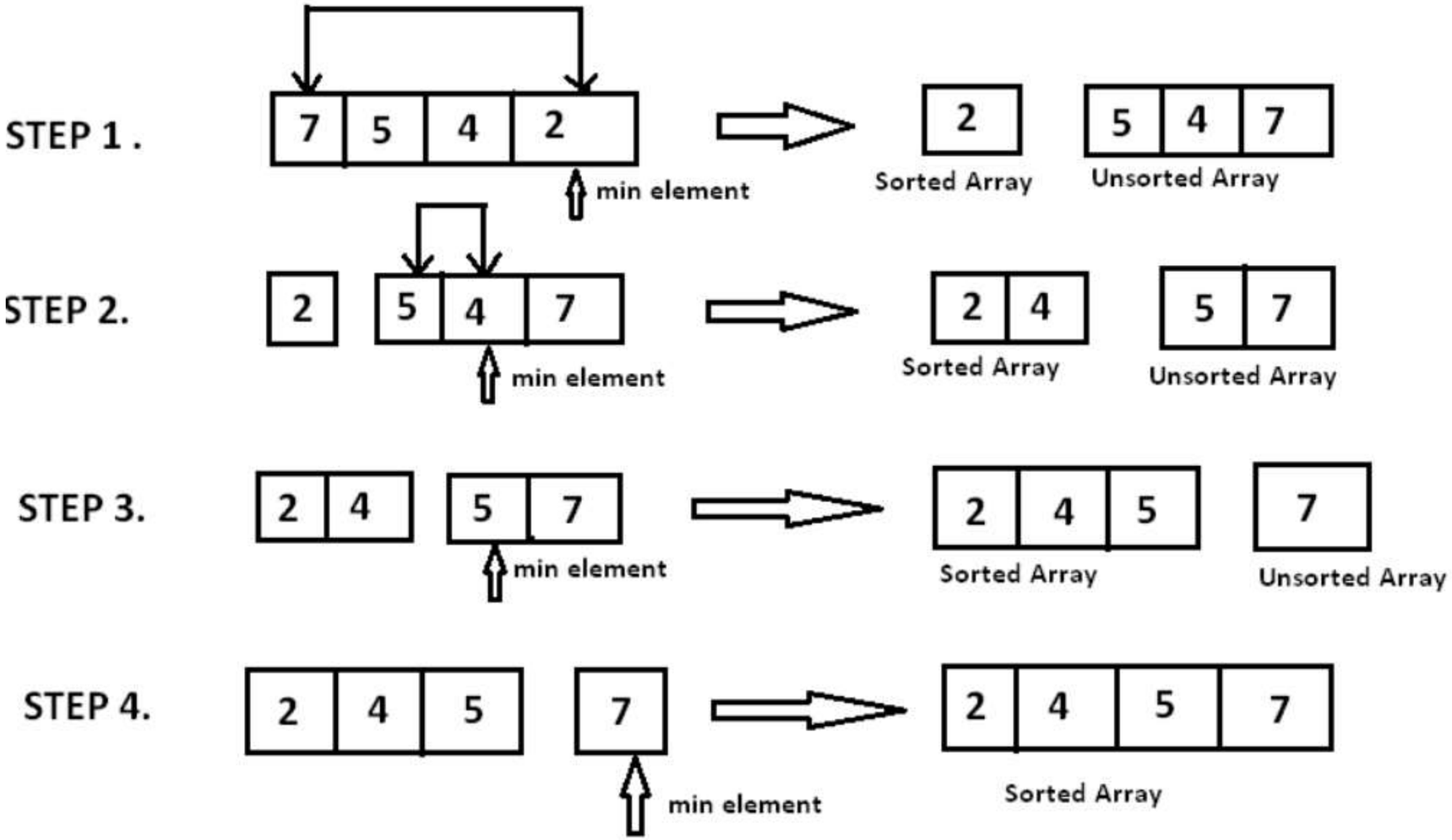
Output of program:



Selection sort

- Selection sort is a simple sorting algorithm.
- This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts.
- The sorted part at the left end and the unsorted part at the right end.
- Initially, the sorted part is empty and the unsorted part is the entire list.

Example-2 Selection sort



Selection Sort algorithm

Let's take an array {20, 12 , 23, 55 ,21}

Set the first element of the array as minimum.

Minimum = 20

Compare the minimum with the next element, if it is smaller than minimum assign this element as minimum. Do this till the end of the array.

Comparing with 12 : 20 > 12 , minimum = 12

Comparing with 23 : 12 < 23 , minimum = 12

Comparing with 55 : 12 < 55 , minimum = 12

Comparing with 21 : 12 < 21 , minimum = 12

Place the minimum at the first position(index 0) of the array.

Array = {12, 20 ,23, 55, 21}

for the next iteration, start sorting from the first unsorted element i.e. the element next to where the minimum is placed.

Array = {12, 20 ,23, 55, 21}

Searching starts from 20, next element where minimum is placed.

Iteration 2 :

Minimum = 20

Comparing with 23 : 20 < 23 , minimum = 20

Comparing with 55 : 20 < 55 , minimum = 20

Comparing with 21 : 20 < 21 , minimum = 20

Minimum in place no change,

Array = {12, 20 ,23, 55, 21}

Iteration 3 :

Minimum = 23.

Comparing with 55 : 23 < 55 , minimum = 23

Comparing with 21 : 23 > 21 , minimum = 21

Minimum is moved to index = 2

Array = {12, 20, 21, 55, 23}

Iteration 4 :

Minimum = 55

Comparing with 23 : 23 < 55 , minimum = 23

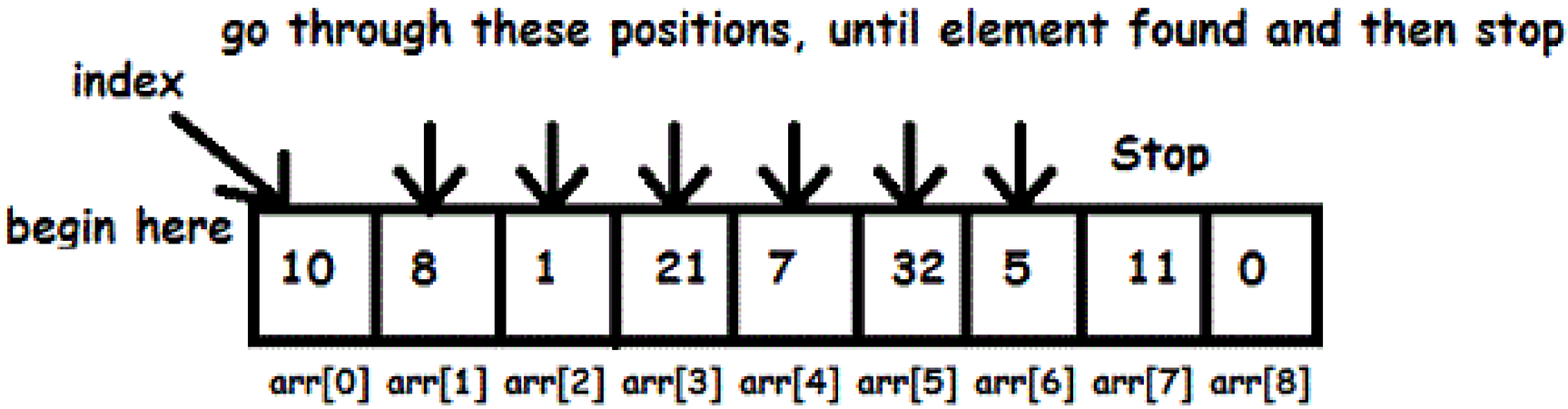
Minimum in moved to index 3 Array = {12, 20, 21, 23, 55}

Selection sort

- **Step 1** – Set MIN to location 0
- **Step 2** – Search the minimum element in the list
- **Step 3** – Swap with value at location MIN
- **Step 4** – Increment MIN to point to next element
- **Step 5** – Repeat until list is sorted

```
#include <stdio.h>
int main() {
   int arr[4]={7,5,4,2};
   int n=4;
   int i, j, position, swap;
   for (i = 0; i < (n - 1); i++) {
      position = i;
      for (j = i + 1; j < n; j++) {
         if (arr[position] > arr[j])
            position = j;
      }
      if (position != i) {
         swap = arr[i];
         arr[i] = arr[position];
         arr[position] = swap;
      }
   }
   for (i = 0; i < n; i++)
      printf("%d\t", arr[i]);
   return 0;
}
```

Linear search or Sequential search



Element to search : 5

Typical Algorithm

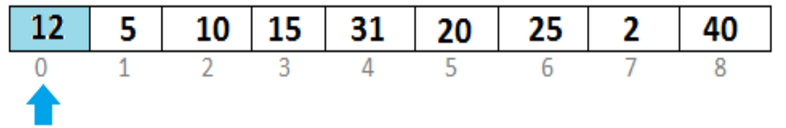
Step 1 − Start from the 0th index of the input array, compare the key value with the value present in the 0th index.
Step 2 − If the value matches with the key, return the position at which the value was found.
Step 3 − If the value does not match with the key, compare the next element in the array.
Step 4 − Repeat Step 3 until there is a match found. Return the position at which the match was found.
Step 5 − If it is an unsuccessful search, print that the element is not present in the array and exit the program.

Conti...

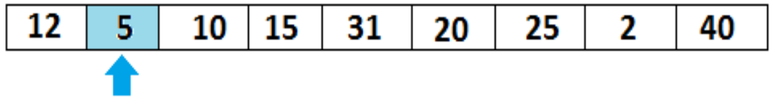
Search 20

12	5	10	15	31	20	25	2	40
0	1	2	3	4	5	6	7	8

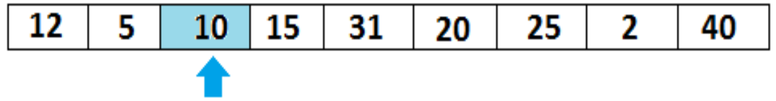
Step 1: *number to search == arr[0]* *20 == 12?* *No, check next element.*



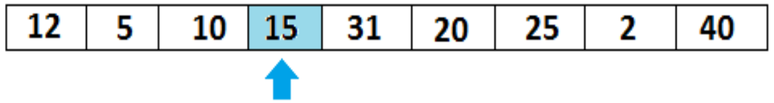
Step 2: *number to search == arr[1]* *20 == 5?* *No, check next element.*



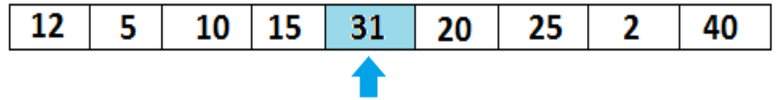
Step 3: *number to search == arr[2]* *20 == 10?* *No, check next element.*



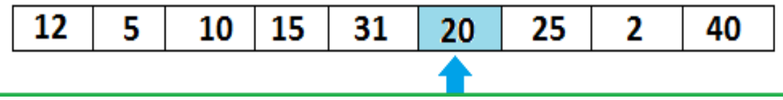
Step 4: *number to search == arr[3]* *20 == 15?* *No, check next element.*



Step 5: *number to search == arr[4]* *20 == 31?* *No, check next element.*



Step 6: *number to search == arr[5]* *20 == 20?* *YES, return true*



Conti...

```
#include <stdio.h>

int linearSearch(int arr[], int n, int target) {
    int i;
    for (i = 0; i< n; i++) {
        if (arr[i] == target) {
            return i;  // Element found at index i
        }
    }
    return -1;  // Element not found
}

int main() {
    int arr[] = {10, 2, 8, 5, 17};
    int n = sizeof(arr) / sizeof(arr[0]);
    int target = 8;
    int result = linearSearch(arr, n, target);
    if (result == -1) {
printf("Element not found in the array.\n");
    } else {
printf("Element found at index: %d\n", result);
    }
    return 0;
}
```

Binary Search

Binary Search is a searching algorithm for finding an element's position in a sorted array.
In this approach, the element is always searched in the middle of a portion of an array.

- This search algorithm works on the principle of divide and conquer.

Algorithm

Binary Search Algorithm

Below is the step-by-step algorithm for Binary Search:

1. Divide the search space into two halves bfinding the middle index “mid”.
2. Compare the middle element of the search space with the **key**.
3. If the **key** is found at middle element, the process is terminated.
4. If the **key** is not found at middle element, choose which half will be used as the next search space.

1. If the **key** is smaller than the middle element, then the **left** side is used for next search.
2. If the **key** is larger than the middle element, then the **right** side is used for next search.

5. This process is continued until the **key** is found or the total search space is exhausted.

Conti...


Binary Search
Search 23
0 1 2 3 4 5 6 7 8 9
2 5 8 12 16 23 38 56 72 91
23 > 16
take 2nd half
L=0 1 2 3 M=4 5 6 7 8 H=9
2 5 8 12 16 23 38 56 72 91
23 > 56
take 1st half
0 1 2 3 4 L=5 6 M=7 8 H=9
2 5 8 12 16 23 38 56 72 91
Found 23,
Return 5
0 1 2 3 4 L=5,M=5 H=6 7 8 9
2 5 8 12 16 23 38 56 72 91

Example Program

```
#include<stdio.h>
void binary_search(int a[], int low, int high, int key)
{ int mid; mid = (low + high) / 2;
if (low <= high)
 { if (a[mid] == key)
printf("Element found at index: %d\n", mid);
 else if(key < a[mid])
binary_search(a, low, mid-1, key);
 else if (key>a[mid])
 binary_search(a, mid+1, high, key); }
else if (low > high)
printf("Unsuccessful Search\n"); }

 void main()
{ int i, n, low, high, key;
n = 5; low = 0; high = n-1;
int a[10] = {12, 14, 18, 22, 39};
key = 22;
binary_search(a, low, high, key);
 }
```

Output:
Element found at index: 3

Assessment 1

1. What is 2D array?

Ans : _____

2. Write C program for bubble sort ?

Ans : _____

References

1. Reema Thareja, “Programming in C”, Oxford University Press, Second Edition, 2016

Thank You