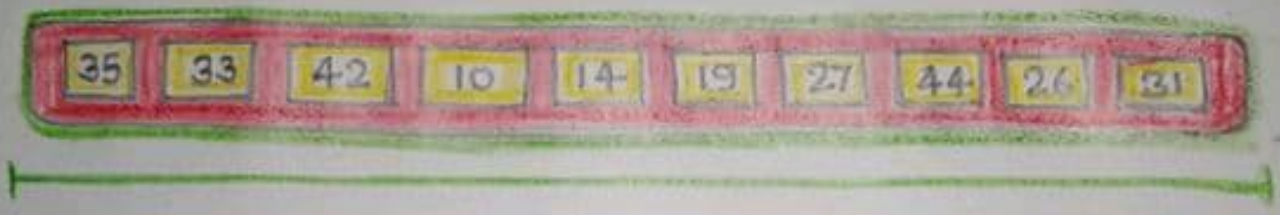
Page No. : .....
Date : ...../...../.....

ASSIGNMENT - 1

NAME : AMRITANSHU KESHARI
BRANCH : COMP. SCIENCE & ENGG.
ROLL NUMBER : 19402060006
SUBJECT : DATASTRUCTURE & ALGO.
SESSION : 2019 - 22
SEMESTER : 4th
COLLEGE : GOVT. POLYTECHNIC
COLLEGE, ADITYAPUR.

Array Declaration



Size : 10

Definition of Array

A container which holds the fix number of items of the same type is known as Array. To implement the algorithms the arrays are used by the data structure. The terminology used in the concept of Array is :

(1) Element
Each item stored in an array is called an element.

(2) Index
Each location of an element in an array has a numerical index, which is used to identify the element.

Representation of Array

Array can be declared in different languages in various ways. For instance, C array declaration is taken.

Name → arr
Elements → {35, 33, 42, 10, 14, 19, 27, 44, 26, 31}

int arr[10] = {35, 33, 42, 10, 14, 19, 27, 44, 26, 31};

↑ Type ↑ Size

As per the above illustration, the important points for consideration are :

- Index starts with 0.
- Array length is 10, which implies 10 elements can be stored.
- Each element can be accessed via its index. For example, we can fetch an element at index 6 as 9.

Operations on Array

The basic operations supported by an array are :

- Traverse
  print all the array elements one by one.

- Insertion
  Adds an element at the given index.

- Deletion
  Deletes an element at the given index.

- Search
  Searches an element using the given index or by the value.

- Update
  Updates an element at the given index.

Traversal

Traversal means accessing each array element for a specific purpose, either to perform an operation on them, counting the total number of elements or else using those values to calculate some other result.

Since array elements is a

linear data structure meaning that all elements are placed in consecutive blocks of memory it is easy to traverse them.

Algorithm of Traversal

Step1 : Initialise counter c = lower_bound index.
Step2 : Repeat step 3 to 4 while c < upper bound.
Step3 : Apply the specified operation on A[c].
step4 : Increment counter : c = c+1
[Loop Ends]
Step5 : Exit

In the first step we initialise the index for our array elements and it will be incremented until last index (step 4) to traverse all array elements. Step 2 specifies the condition that traversal will continue till upper bound of array is reached. Step 3 we perform the operation using each array element in every iteration.

Following program traverses and prints the elements of an array :

```
#include <stdio.h>
int main()
{
    int LA[] = {1,3,5,7,8};
    int item = 10, k = 3, n = 5;
    int i=0, j=n;
    printf("The original array elements are
    :\n");
    for(i=0; i<n; i++)
    {
        printf("LA[%d] = %d\n", i, LA[i]);
    }
    return 0;
}
```

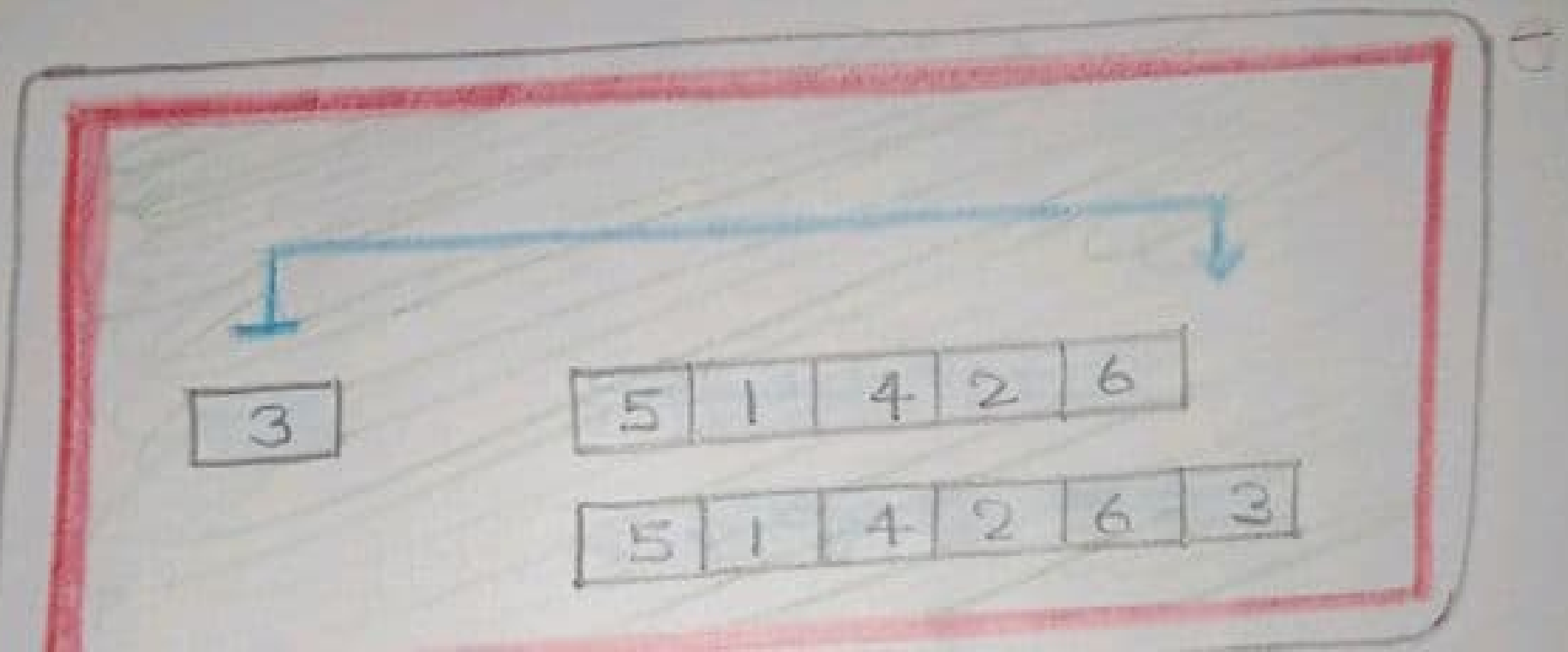
Traverse Operation

The traverse operation is mainly used to traverse through the elements of an array loop by loop using for loop concept.

OUTPUT

```
The original array elements are :
LA [0] = 1
LA [1] = 3
LA [2] = 5
LA [3] = 7
LA [4] = 8
```

Insert Operation in Unsorted Array



Insertion Operation of Array

Inserting one or more element or data element into an array is known as insert operation. A new element is either added at beginning, end or at any given index of array, based on the requirement.

Algorithm of Insertion Operation

Let an array be a linear unordered array of MAX elements.

Let LA be a linear array (unordered) with N elements and K is a positive integer such that $K <= N$. Following is the algorithm where ITEM is inserted into the Kth position of LA –

1. Set J = N
2. Set N = N + 1
3. Repeat steps 5 & 6 while J >= K
4. Set LA[J+1] = LA[J]
5. Set J = J - 1
6. Set LA[K] = ITEM

Following program insert one or more data elements into an array :

```
#include <stdio.h>
int main()
{
  int LA[] = {1,3,5,7,8};
  int item = 10, k = 3, n = 5;
  int i = 0, j = n;
  printf ("The original array elements
  are :\n");
  for (i=0; i<n; i++)
  {
    printf ("LA[%d] = %d \n", i, LA[i]);
  }
  n = n + 1;
  while (j>=k)
  {
    LA[j+1] = LA[j];
    j = j-1;
  }
  LA[k] = item;
  printf ("The array elements after
  insertion :\n");
  for (i= 0; i<n; i++) {
    printf ("LA[%d] = %d\n", i, LA[i]);
  }
}
```

OUTPUT

The original array elements are:

LA[0] = 1

LA[1] = 3

LA[2] = 5

LA[3] = 7

LA[4] = 8

The array elements after insertion:

LA[0] = 1

LA[1] = 3

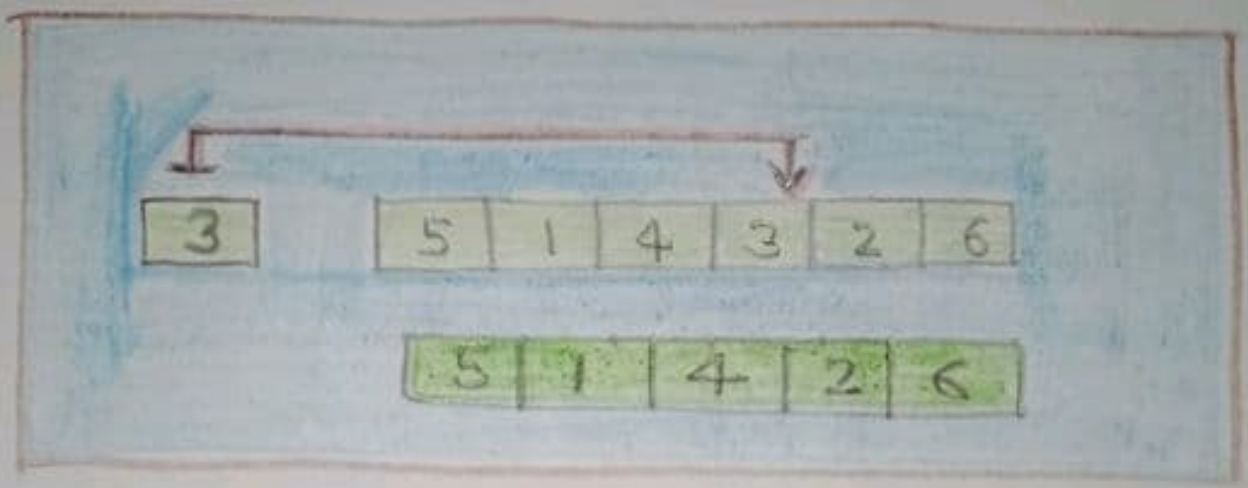
LA[2] = 5

LA[3] = 10

LA[4] = 7

LA[5] = 8

Delete Operation in Unsorted Array



Deletion Operation of Array

To remove an existing element from the array and to re-organize the other elements of an array, is known as Deletion operation.

Algorithm of Deletion Array or Deletion Operation

Consider LA is a linear array with N elements and K is a positive integer such that K <= N.

Following is the algorithm to delete an array element available at the Kth position of LA.

1. Start
2. Set J = K
3. Repeat step 4 & 5 while J < N
4. Set LA[J-1] = LA[J]
5. Set J = J+1
6. Set N = N-1
7. Stop

Following program delete one or more data elements into an array :

```
#include <stdio.h>

int main()
{
    int LA[] = {1,3,5,7,8};
    int k=3, n=5;
    int i, j;
    printf("The original array elements are
    :\n");
    for(i=0; i<n; i++)
    {
        printf("LA[%d] = %d \n", i, LA[i]);
    }
    j=k;
    while (j<n) {
        LA[j-1] = LA[j]
        j = j+1;
    }
    n = n-1;
    printf("After deletion : \n");
    for (i=0; i<n; i++)
    {
        printf("LA[%d] = %d \n", i, LA[i]);
    }
    return 0;
}
```

OUTPUT

The original array elements are :

LA [0] = 1

LA [1] = 3

LA [2] = 5

LA [3] = 7

LA [4] = 8

After deletion :

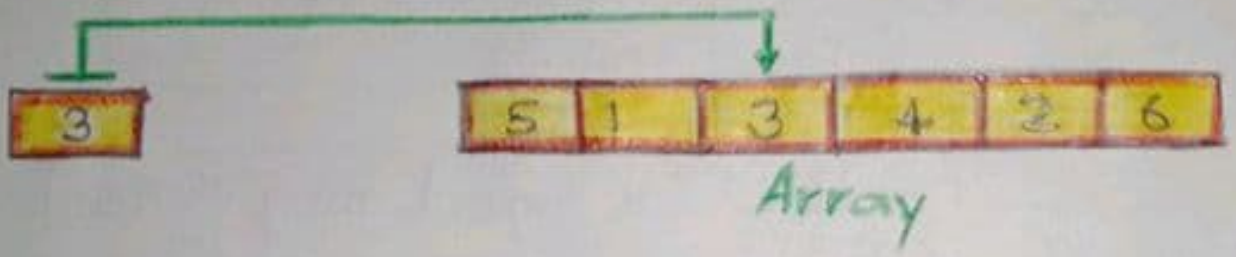
LA [0] = 1

LA [1] = 3

LA [2] = 7

LA [3] = 8

Search Operation in Unsorted Array



Search Operation of Array

In an unsorted array, the search operation can be performed by linear traversal from the first element to the last element. A search operation is performed for an array element based on its value or its index.

Algorithm of Search Operation :

Consider LA is a linear array with N elements and K is a positive integer such that K<=N; . Following is the algorithm to find an element with a value of ITEM using sequential search.

1. Start
2. Set J = 0
3. Repeat step 4 & 5 while J < N
4. IF LA[J] = ITEM Then GOTO S-6
5. Set J = J+1
6. PRINT J, ITEM
7. Stop

Following program search data elements into an array :

```
#include <stdio.h>

void main()
  {
    int LA[]= {1,3,5,7,8};
    int item = 5, n = 5;
    int i= 0; j = 0;

    printf ("The original array element
     are : \n");
    for (i=0; i<n; i++)
      {
        printf ("LA[%d] = %d \n", i, LA[i]);
      }
    while (j<n)
      {
        if (LA[j] == item) {
            break;
          }
        j = j+1;
      }
    printf ("Found element %d at position
     %d\n", item, j+1);
  }
```

OUTPUT

```
The original array elements are:
LA[0] = 1
LA[1] = 3
LA[2] = 5
LA[3] = 7
LA[4] = 8
Found element 5 at position 3
```

Update Operation of Array

Update operation refers to updating an existing element from the array at a given index. Update or updating of an existing element from the array at a given index is known as Update operation.

Algorithm of Update Operation

Consider LA is a linear array with N elements and K is a positive integer such that K <= N. Following is the algorithm to update an element available at the K[th] position of LA.

1. Start
2. Set LA[K-1] = ITEM
3. Stop

It is better understand with the practical implementation of update operation, where the data is updated at the end of the array.

Following program update the data elements into an array :

```
#include <stdio.h>

int main()
{
    int LA[] = {1,3,5,7,8};
    int k=3, n=5, item = 10;
    int i, j;

    printf ("The original array elements
    are : \n");

    for (i=0; i<n; i++) {
      printf ("LA[%d] = %d \n", i,
      LA[i]);
    }
    LA [k-1] = item;

    printf ("The array elements
    after updation : \n");

    for (i=0; i<n; i++) {
        printf ("LA[%d] = %d \n", i, LA[i]
    }
    return 0;
}
```

OUTPUT

```
The original array elements are :
LA[0] = 1
LA[1] = 3
LA[2] = 5
LA[3] = 7
LA[4] = 8
The array elements after updation :
LA[0] = 1
LA[1] = 3
LA[2] = 10
LA[3] = 7
LA[4] = 8
```

Sorting in Array

Sorting an array means to arrange the elements in the array in a certain order. Various algorithms have been designed that sort the array using different methods. Some of these sorts are more useful than the others in certain situations.

Internal / External Sorting

Internal sorting means that all the data that is to be sorted is stored in memory while sorting is in progress.

External sorting means that the data is stored outside memory (like on disk) and only loaded into memory in small chunks.

External sorting is usually applied in cases when data cannot fit into memory entirely, effectively allowing to sort data that does not fit in the memory.

Stability of Sort

A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in the sorted output as they appear in the unsorted input.

A sorting algorithm is said to be unstable if there are two or more objects with equal keys which do not appear in same order before and after sorting.

Bubble Sort

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order. The pass through the list is repeated until the list is sorted.

This is an inefficient sort as it has to loop through all the elements multiple times. It takes O(n^2) time to completely

Sort the array.

Here is the following program for the bubble sort algorithm.

```
# include <stdio.h>

int main()
  {
    int arr[100], n, i, j, temp;
    printf(" Enter number of elements
          \n");
    scanf ("%d", &n);

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

    for (i=0 ; i<n-1; i++)
     {
       for (j=0; j<n-i-1 ; j++)
        {
          if (arr[j] > arr[j+1])
            {
              temp = arr[j];
              arr[j] = arr[j+1];
              arr[j+1] = temp;
            }
        }
     }
```

```
    printf ("%d \n", arr[i]);

    return 0;
}
```

Properties

- Average Time Complexity : O(n^2)
- Stability : Stable

Best Use Case

This is a very elementary sort which is easy to understand and implement. It is not recommended in actual production environments. No external memory is required to sort as it is an in-place sort.

Example of Bubble Sort

Bubble sort is the simplest sorting algorithm that works by repeatedly swapping the algorithm the adjacent elements if they are in wrong order.

First Pass :

(|51|428) → (|15|428)
Here, the algorithm compares the first two elements and swaps since 5 > 1.

(1|54|28) → (1|45|28)
Swap since 5 > 4

(14|52|8) → (14|25|8)
Swap since 5 > 2

(142|58|) → (142|58|)
Now, since these elements are already in order (8 > 5), algorithm does not swap them.

Second Pass :

(|14|258) → (|14|258)
(1|42|58) → (1|24|58)
Swap since 4 > 2

(12|45|8) → (12|45|8)

(124|58|) → (124|58|)

Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one whole pass without any swap to know it is sorted.

Third Pass:

([1 2] 4 5 8) ——> ([1 2] 4 5 8)
(1 [2 4] 5 8) ——> (1 [2 4] 5 8)
(1 2 [4 5] 8) ——> (1 2 [4 5] 8)
(1 2 4 [5 8]) ——> (1 2 4 [5 8])

Insertion Sort

In insertion sort, every iteration moves an element from unsorted portion to sorted portion until all the elements are sorted in the list. An analogy of insertion sort is the sorting of a deck of cards with our hands. We select one card from the unsorted deck and put it in th right order in our hands, effectively sorting the whole deck.

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

Steps for Insertion Sort

1. Assume that first element in the list is in its sorted portion of the list and remaining all elements are in unsorted portion.

2. Take the first element from the unsorted list and insert that element into the sorted list in order specified (ascending or descending).

3. Repeat the above process until all the elements from the unsorted list are moved into the sorted list.

Here is the following program for the bubble sort algorithm.

```
#include <stdio.h>

int main()
{
  int data[100], n, temp, i, j;
  printf("Enter number of elements to be
  sorted : ");
  scanf("%d", &n);
  printf("Enter elements: ");

  for (i=0; i<n; i++)
    {
      scanf("%d", &data[i]);
    }
      for (i=1; i<n; i++)
        {
          temp = data[i];
          j = i - 1;
          while (temp < data[j]
            && j>=0)
            {
              data[j+1] = data[j];
                  j = j-1;
            }
          data[j+1] = temp;
```

```
}
printf ("Sorted array : ");
for(i=0; i<n; i++)
    {
        printf ("%d ", data[i]);
        return 0;
    }
```

Properties of Insertion Sort

- Average Time Complexity : O(n^2)
- Stability : Stable

Best Use Case

Although this is a elementary sort with the worst case of O(n^2), it performs much better when the array is nearly sorted, as lesser elements would have to be moved. It is also preferred when the number of elements are less as it has significantly less overhead than the other sorts. It consumes less memory and is simpler to the implement.

In some quick sort implementations, insertion sort is internally to sort the smaller lists faster.

Selection Sort

Selection sort is generally used for sorting files with very large records and small keys. It selects the smallest (or largest) element in the array and then removes it to place in a new list. Doing this multiple times would yield the sorted array.

Steps for Selection Sort

1. Select the first element of the list.
2. Compare the selected element with all other elements in the list.
3. For every comparison, if any element is smaller (or larger) than selected element, swap these two elements.

4. Repeat the same procedure with next position in the list till the entire list is sorted.

Here is the following program for the selection sort algorithm.

```
#include <stdio.h>

int main()
{
    int array[100], n, pos, temp, i, j;

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

    printf("Enter the %d values \n", n);

    for(i=0; i<n; i++)
    {
        scanf("%d", &array[i]);
    }
    for(i=0; i<(n-1); i++)
    {
        pos = i;
```

```
for (j = i+1 ; j < n ; j++)
  {
    if ( array [pos] > array [j])
        pos = j ;
    }
    if ( pos != i)
    {
        temp = array [i] ;
        array [i] = array [pos] ;
        array [pos] = temp ;
    }
}
printf (" Sorted list in ascending
order : \n ") ;

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

  return 0 ;
}
```

Properties of Selection Sort

- Average Time Complexity : O (n^2)
- Stability : Non - Stable