UNIT – V

SEARCHING

DEFINITION

It is a method of finding the given element in the given list of elements.

or

It is technique to find the location where the element is available or present.

or

It is an algorithm to check whether a particular element is present in the given list or not.

Types of Searching

- ✓ **Linear Search**
- ✓ **Binary Search**
- ✓ **Fibonacci Search**

LINEAR SEARCH

- ✓ It is a very **simple** search algorithm when compared with the other two search algorithms.
- ✓ It is also called as **sequential search** or **indexed search**.
- ✓ To perform linear search, the list of elements **need not be sorted**.
- ✓ An ordered or unordered list will be searched by comparing the search element with one by one element from the beginning of the list until the desired element is found or till the end of the list.
- ✓ If the desired element is found in the list then the search is **successful** otherwise **unsuccessful**.
- ✓ The **time complexity** for linear search is **O(n)** where n is the number of elements in the list.
- ✓ The time complexity increases with the increase of the input size **n**.

Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Algorithm for Linear Search

LINEAR_SEARCH(A, N, KEY)

Step 1: SET POS = -1

Step 2: SET I = 1

Step 3: Repeat Step 4 while I<=N

Step 4: IF A[I] = KEY

SET POS = I

PRINT POS

Go to Step 6

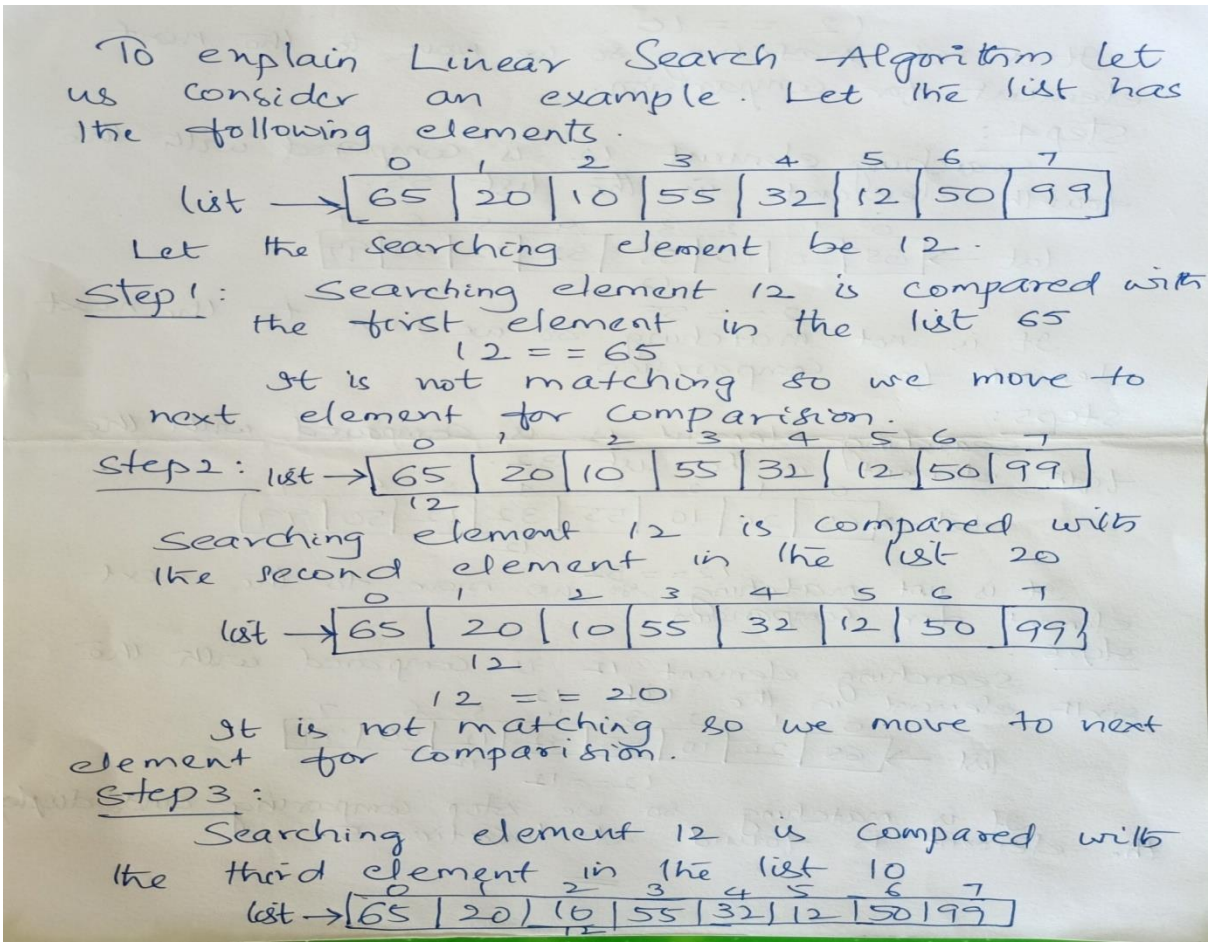
SET I = I + 1

Step 5: IF POS = –1

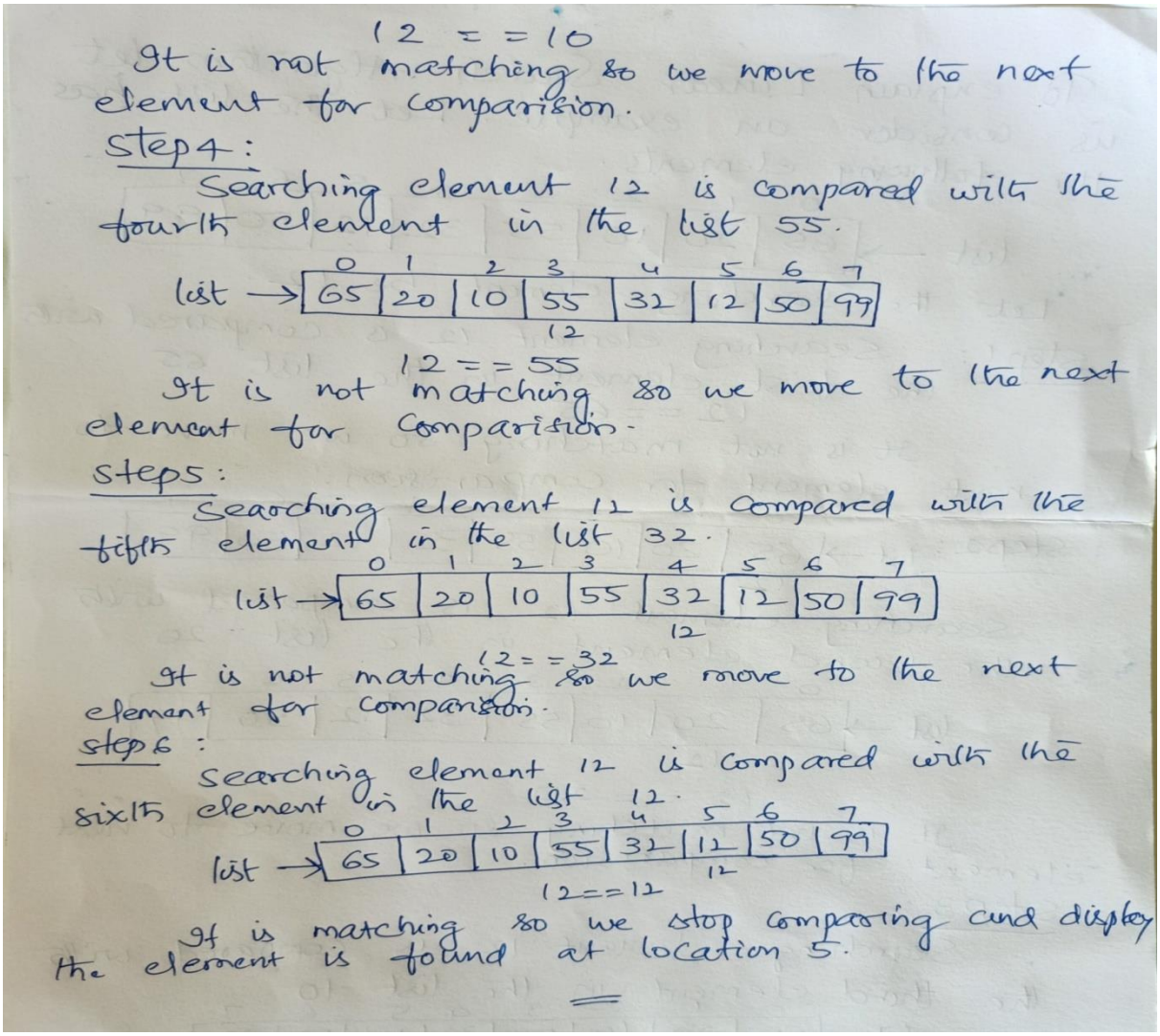
PRINT “VALUE IS NOT PRESENT IN THE ARRAY”

Step 6: EXIT

Example of Linear Search



Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



Program for Linear Search

```
#include<stdio.h>
int main(void)
{
        int a[20], n, i, key;
        printf("Enter size of the list: ");
        scanf("%d", &n);
        printf("Enter the elements”);
        for(i = 0; i < n; i++)
                scanf("%d", &a[i]);
        printf("Enter the element to be Search: ");
        scanf("%d", &key);
```


Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

```
for(i = 0; i < n; i++)
        {
                if(key == a[i])
                {
                        printf("Element is found at %d index", i);
                        break;
                }
        }
        if(i == n)
                printf("Given element is not found in the li st!!!");
        return 0;
}
```

BINARY SEARCH

- ✓ It is the **fastest** searching algorithm when compared with the other two algorithms.
- ✓ It works on the principle **divide – conquer** strategy.
- ✓ To apply binary search algorithm the list of elements should be in **sorted order**.
- ✓ The time complexity for binary search algorithm is **O(log n)**.
- ✓ It is applied to **very large set** of elements
- ✓ The process carried by binary search algorithm is find the middle element and compare it with search element it match return the index of the element and say success otherwise see if the search element is greater than or less than the middle element.
- ✓ If it is greater than the middle element then search the element in the upper part of the list otherwise in the lower part of the list.
- ✓ Again find middle element and do the same process till the element is found or not found.
- ✓ Using binary search algorithm we can reduce the number of comparisons hence it is best.

Algorithm forBinary Search

BINARY_SEARCH(A, lower_bound, upper_bound, KEY)

Step 1: SET BEG = lower_bound

END = upper_bound, POS = - 1

Step 2: Repeat Steps 3 and 4 while BEG <= END

Step 3: SET MID = (BEG + END)/2

Step 4: IF A[MID] = KEY

SET POS = MID

PRINT POS

Go to Step 6

ELSE IF A[MID] > VAL

SET END = MID - 1

ELSE

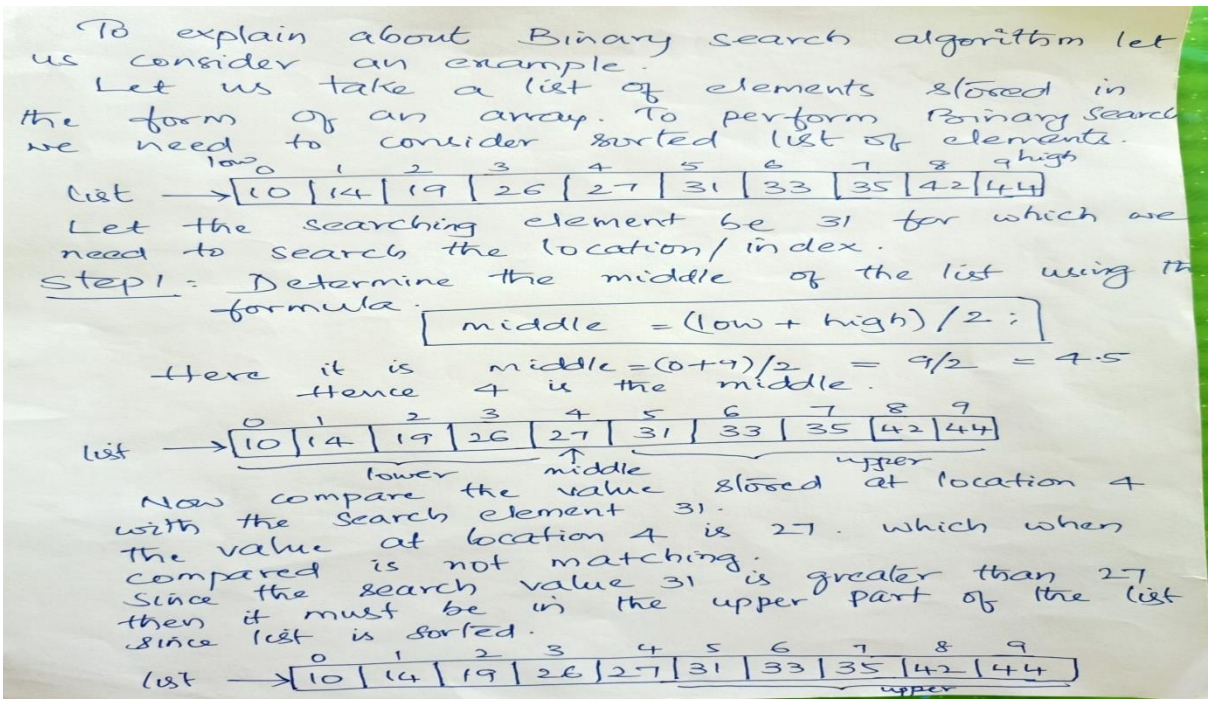
SET BEG = MID + 1

Step 5: IF POS = -1

PRINT "VALUE IS NOT PRESENT IN THE ARRAY"

Step 6: EXIT

Example of Binary Search

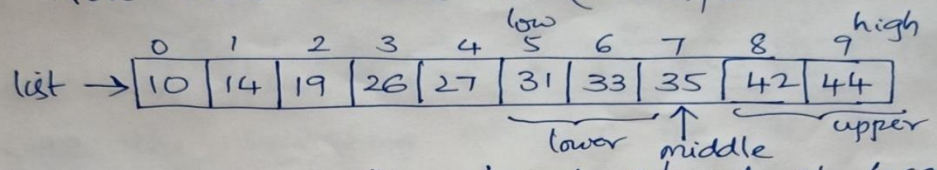


Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Step2: Now we change low to middle +1 and find new middle value again.

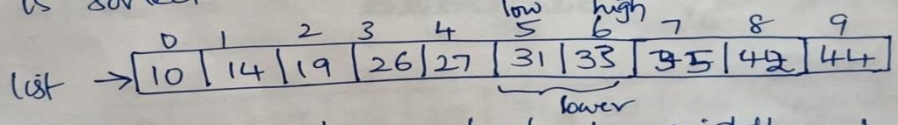
```
low = middle +1
middle =(low+high)/2
```

Now new middle is (5+9)/2 = 14/2 = 7



Now compare the element stored at location 7 which is 35 with the search element 31. The element at location 7 is not matching.

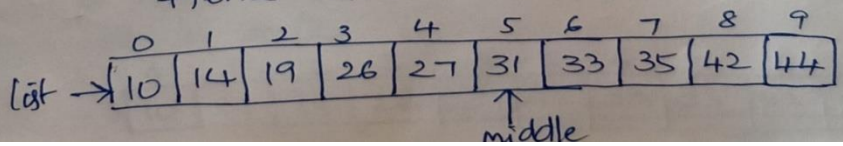
Since the search element 31 is less than 35 then it must be in the lower part of the list as it is sorted.



Step3: Now we change high to middle -1 and find new middle value again.

```
low = 5
high = middle -1
middle =(low + high)/2
```

Now new middle is (5+6)/2 = 11/2 = 5.5
Hence middle is 5.



Now compare the element stored at location 5 which is 31 with the search element 31. Now the element at location 5 is matching. Hence the desired location where the search element is stored is found.

=

Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Program for Binary Search

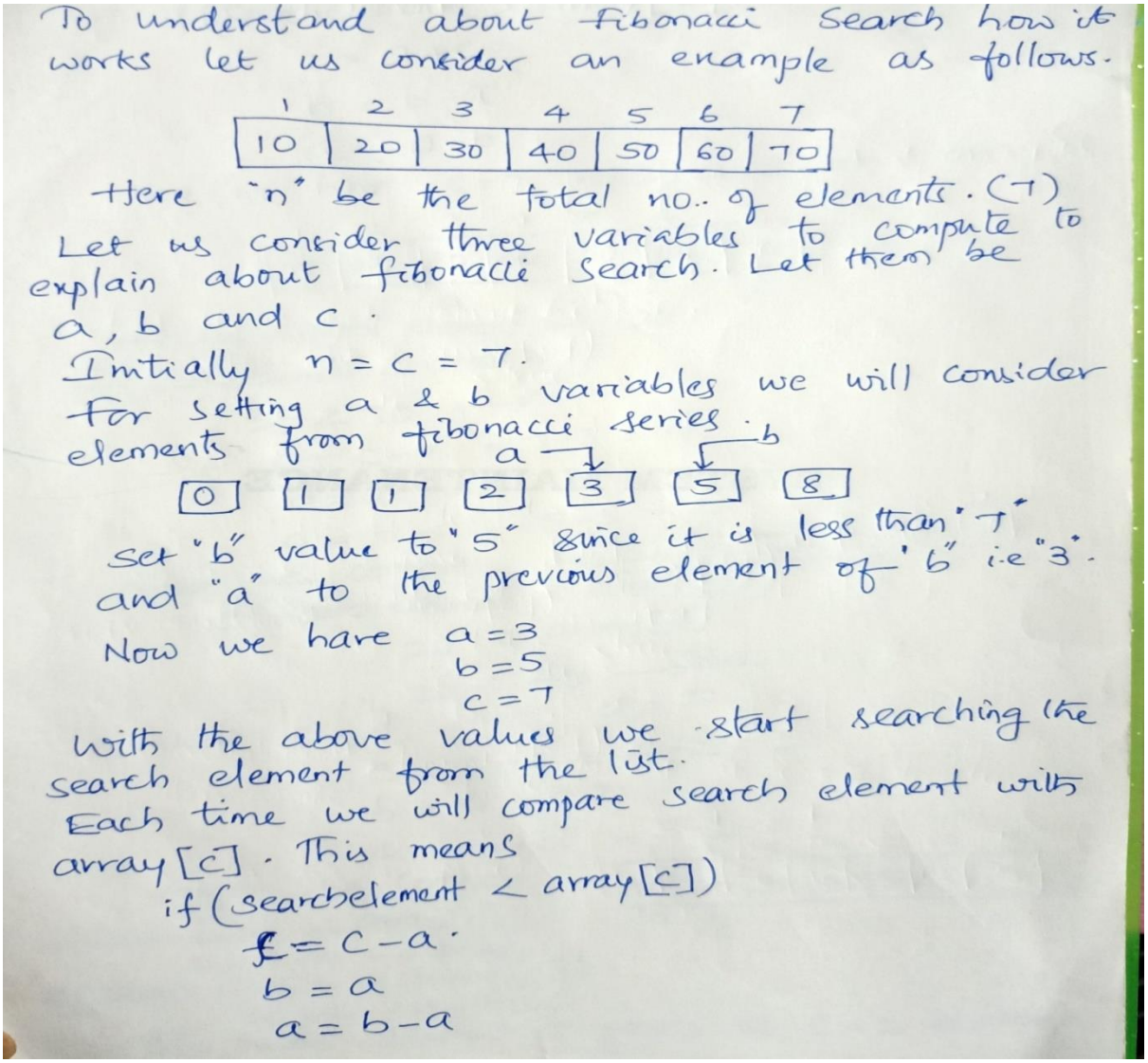
```
#include <stdio.h>
int main(void)
{
        int i, low, high, middle, n, key, a[10];
        printf("Enter number of elements");
        scanf("%d", &n);
        printf("Enter the elements”);
        for (i = 0; i < n; i++)
                scanf("%d", &a[i]);
        printf("Enter value to find");
        scanf("%d", &key);
        low = 0;
        high = n - 1;
        middle = (low + high)/2;
        while(low <= high)
        {
                if(a[middle] < key)
                        low = middle + 1;
                else if(a[middle] == key)
                {
                        printf(“Element is found”);
                        break;
                }
                else
                        high = middle - 1;
                middle = (low + high)/2;
        }
        if(low > high)
                printf(“Element is not found” );
        return 0;
    }
```


Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

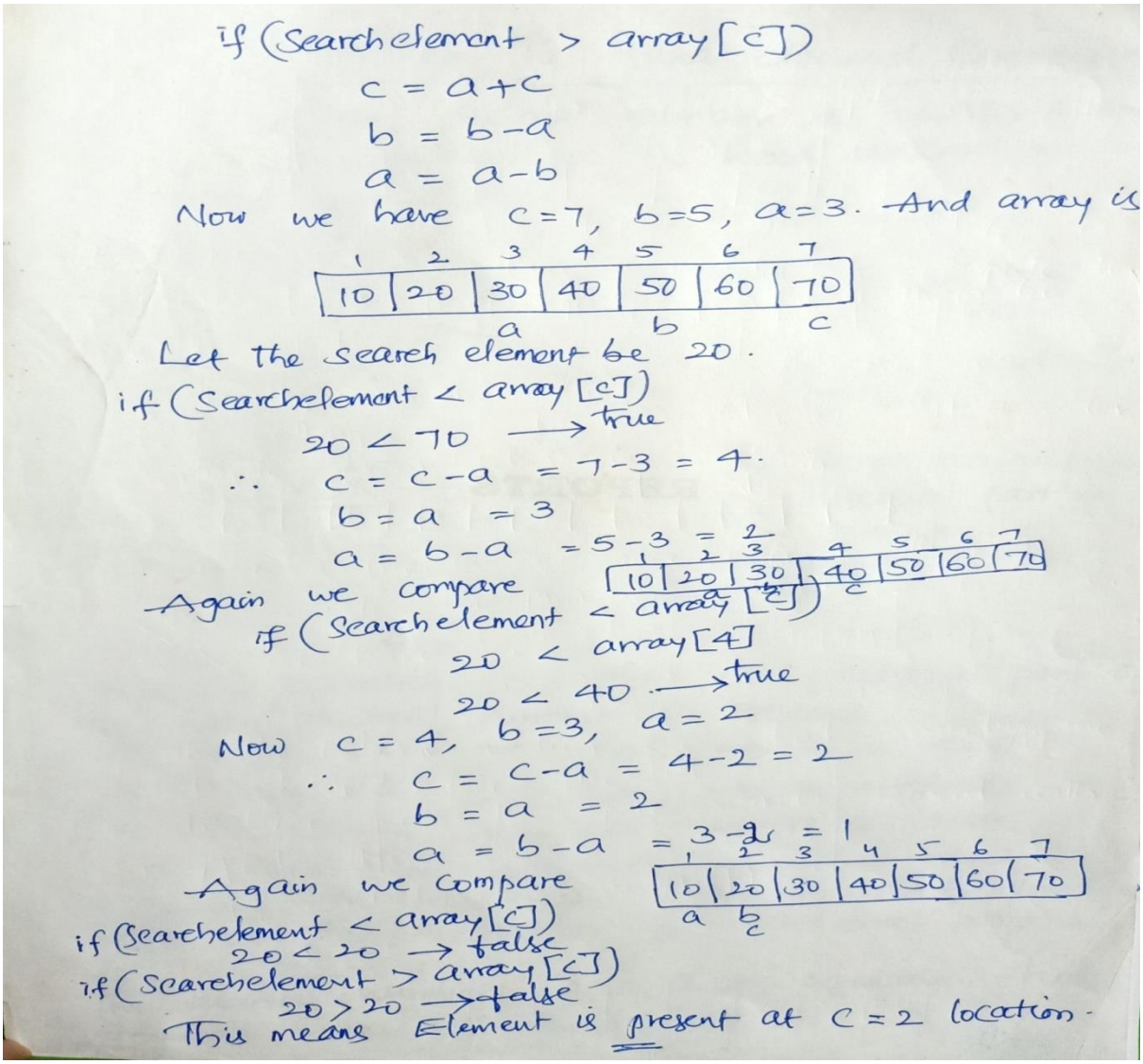
FIBONACCI SEARCH

- ✓ It was developed by Kiefer in 1953.
- ✓ In Fibonacci search we consider the indices as numbers from fibonacci series.
- ✓ To apply fibonacci search algorithm the list that contains elements should be in sorted order.
- ✓ The time complexity of fibonacci search algorithm is **O(log n)**
- ✓ It works on the principle **divide - conquer** strategy.

Example of Fibonacci Search



Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



Program for Fibonacci Search

```
#include<stdio.h>
int main(void)
{
        int n, key, i, ar[20];
        void search(int ar[], int n, int key, int f, int a, int b);
        int fib(int n);
        clrscr();
        printf("\n Enter the number of elements in array");
        scanf("%d", &n);
        printf("\n Enter the elements");
        for(i=0;i<n;i++)
```


Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

```
            scanf("%d", &ar[i]);
        printf("Enter the element to be searched");
        scanf("%d", &key);
        search(ar, n, key, n, fib(n), fib(fib(n)));
        return 0;
}
int fib(int n)
{
        int a, b, f;
        if(n<1)
                return n;
        a=0;
        b=1;
        while(b<n)
        {
                f=a+b;
                a=b;
                b=f;
        }
        return a;
}
void search(int ar[], int n, int key, int f, int b, int a)
{
        if(f<1 || f>n)
                printf("the number is not present");
        else if(key<ar[f])
        {
                if(a<=0)
                        printf("The element is not present in the list");
                else
                        search(ar, n, key, f-a, a, b-a);
        }
```

```
        else if(key>ar[f])
        {
                if(b<=1)
                        printf("The element is not present in the list");
                else
                        search(ar, n, key, f+a, b-a, a-b);
        }
        else
                printf("Element is present %d", f);
}
```

SORTING

DEFINITION

Sorting is a technique to rearrange the list of elements either in ascending or descending order, which can be numerical, alphabetical or any user-defined order.

Types of Sorting

Internal Sorting

- ✓ If the data to be sorted remains in main memory and also the sorting is carried out in main memory then it is called internal sorting.
- ✓ Internal sorting takes place in the main memory of a computer.
- ✓ The internal sorting methods are applied to small collection of data.
- ✓ The following are some internal sorting techniques:
  - ✓ **Insertion sort**
  - ✓ **Merge Sort**
  - ✓ **Quick Sort**
  - ✓ **Heap Sort**

External Sorting

- ✓ If the data resides in secondary memory and is brought into main memory in blocks for sorting and then result is returned back to secondary memory is called external sorting.
- ✓ External sorting is required when the data being sorted do not fit into the main memory.
- ✓ The following are some external sorting techniques:
  - ✓ **Two-Way External Merge Sort**
  - ✓ **K-way External Merge Sort**

INSERTION SORT

In this method, the elements are inserted at their appropriate place. Hence the name insertion sort.

- ✓ This sorting is very simple to implement.

Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

- ✓ This method is very efficient when we want to sort small number of elements.
- ✓ This method has excellent performance when almost the elements are sorted.
- ✓ It is more efficient than bubble and selection sorts.
- ✓ This sorting is stable.
- ✓ This is an in-place sorting technique.
- ✓ The time complexity of insertion sort for best case is O(n), average case and worst case is $O(n^2)$.

Algorithm for Insertion Sort

INSERTION-SORT (A, N)

Step 1: Repeat Steps 2 to 5 for I = 1 to N – 1

Step 2: SET TEMP = A[I]

Step 3: SET J = I - 1

Step 4: Repeat while TEMP <= A[J]

SET A[J + 1] = A[J]

SET J = J - 1

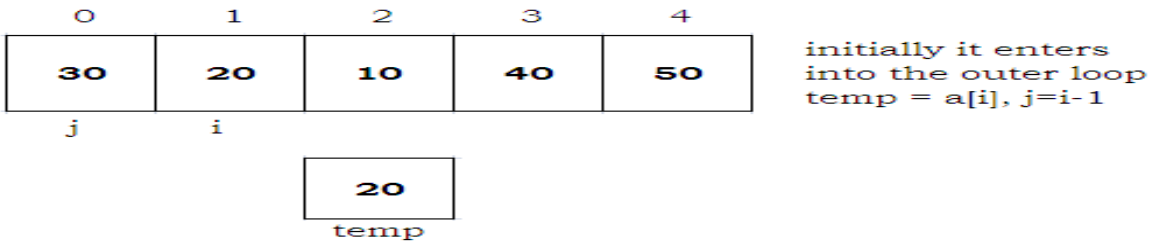
Step 5: SET A[J + 1] = TEMP

Step 6: EXIT

Example for insertion sort

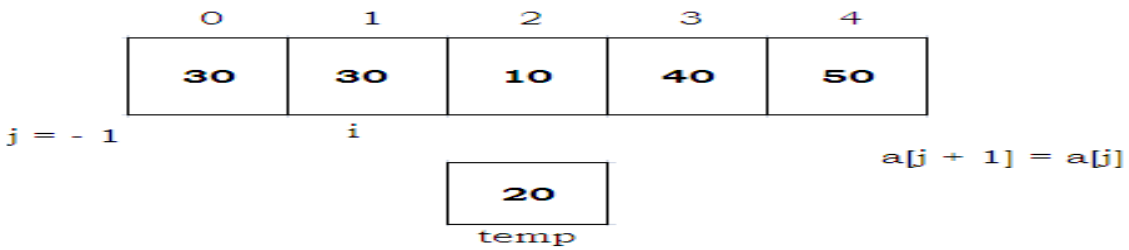
Let us consider the array of elements to sort them using insertion sort technique

30, 20, 10, 40, 50

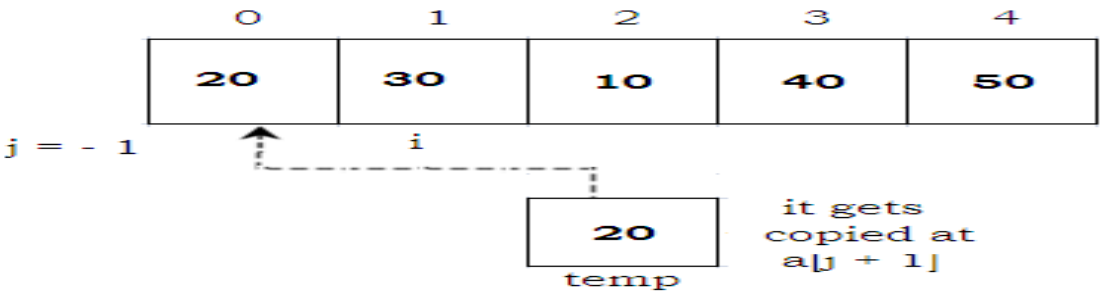


The control moves to while loop as j>=0 and a[j] > temp is true, the while is executed.

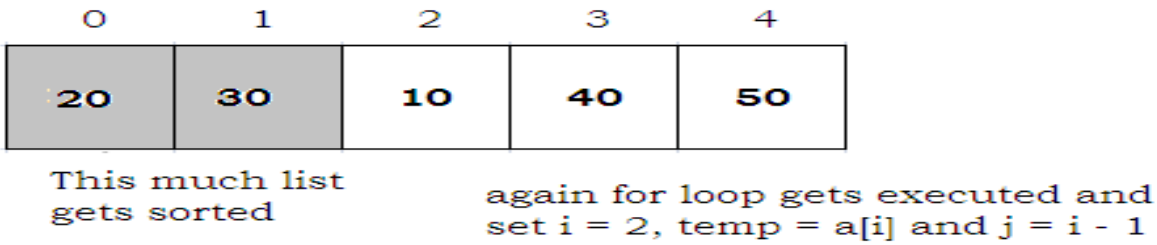
Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



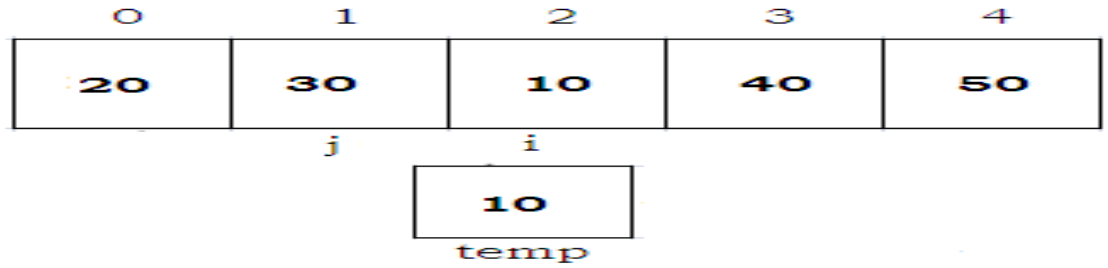
Now since j >= 0 is false, control comes out of while loop

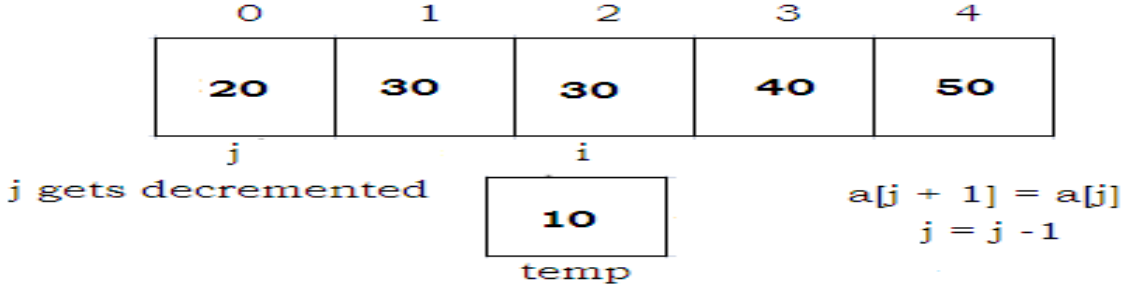


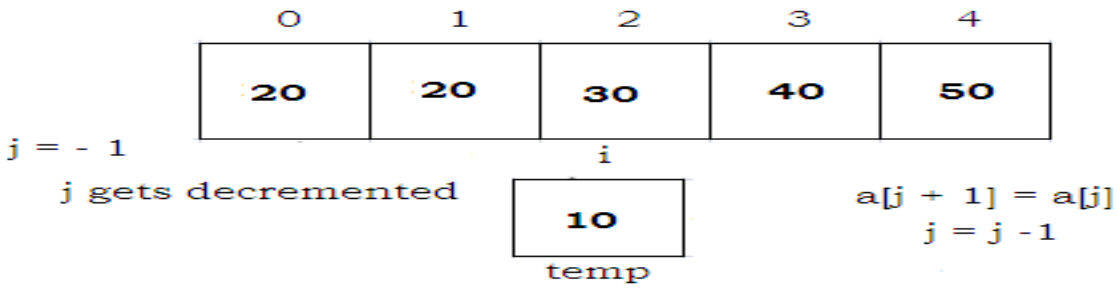
then the list becomes



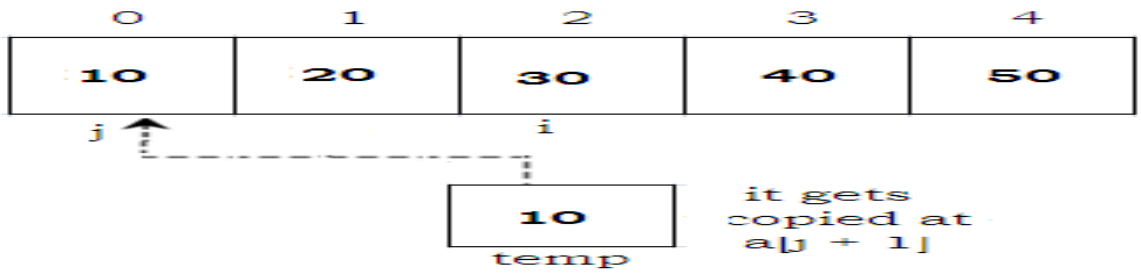
The control moves to while loop as j>=0 and a[j] > temp is true, the while is executed.



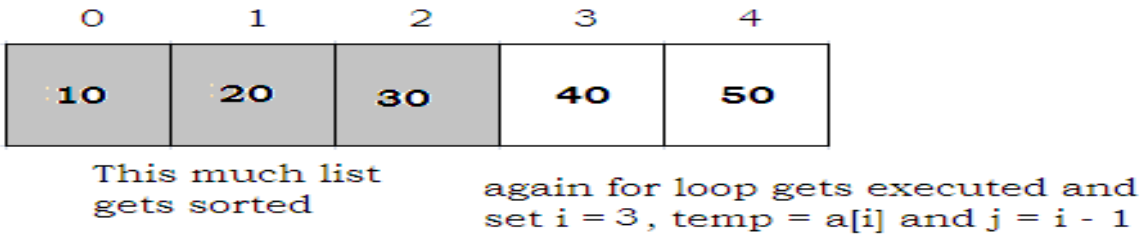




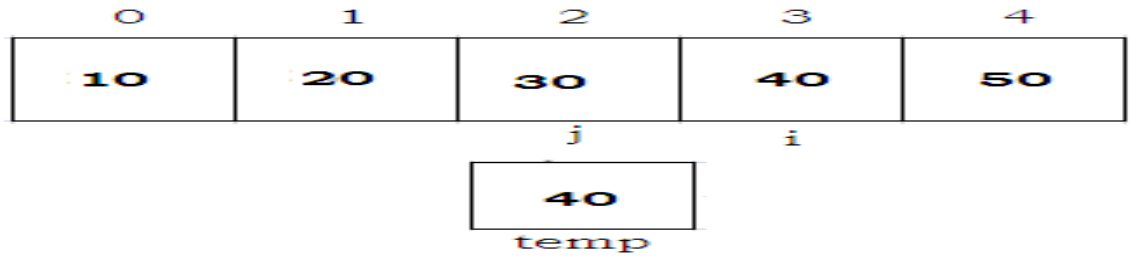
Now since j >= 0 is false, control comes out of while loop

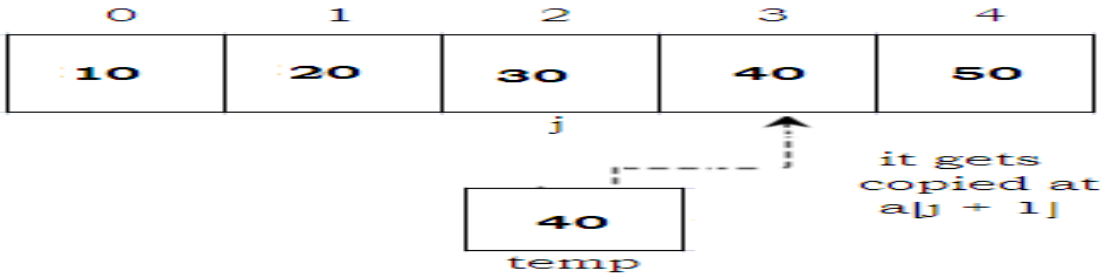


Then the list becomes



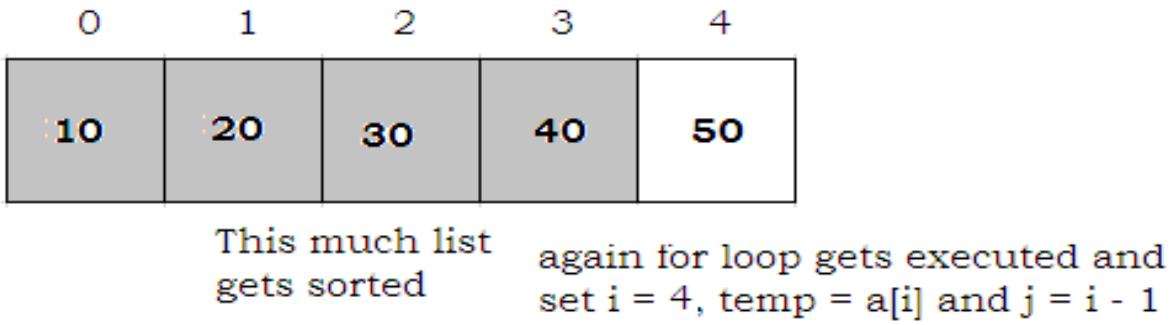
The control moves to while loop as j>=0 and a[j] > temp is false, the while is not executed.



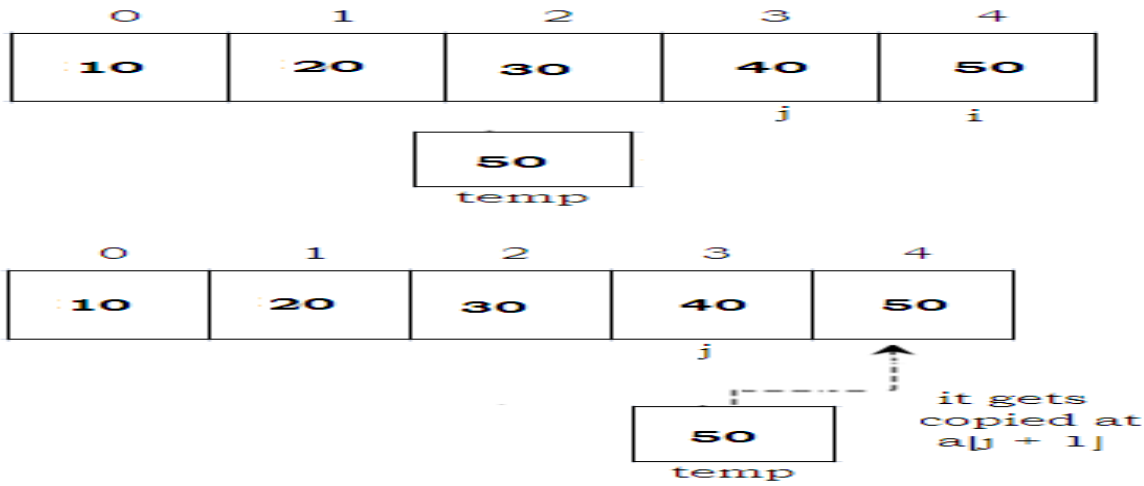


Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

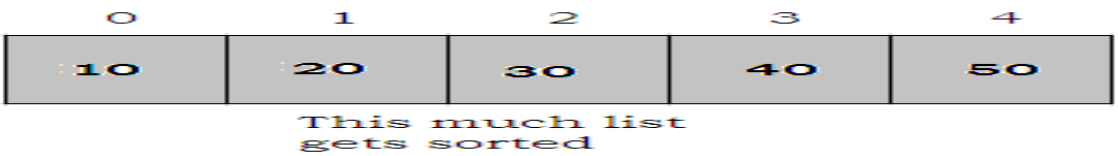
Then the list becomes



The control moves to while loop as j>=0 and a[j] > temp is false, the while is not executed.



Then the list becomes



Program to illustrate insertion sort technique.

```
#include<stdio.h>
void insert_sort(int [], int);
int main(void)
{
        int n, a[10], i;
        clrscr();
        printf(" Enter the size of the array ");
        scanf("%d", &n);
```

```
        printf(" Enter the elements of the array ");
        for(i=0; i<n; i++)
                scanf("%d", &a[i]);
        insert_sort(a, n);
        return 0;
}
void insert_sort(int a[], int n)
{
        int i,j,temp;
        for(i=1; i<n; i++)
        {
                temp = a[i];
                j = i - 1;
                while(j >= 0 && a[j] > temp)
                {
                        a[j+1] = a[j];
                        j = j - 1;
                }
                a[j+1]=temp;
        }
        printf(" \n The sorted list of elements are ");
        for(i=0; i<n; i++)
                printf("%d\t", a[i]);
}
```

SELECTION SORT

- ✓ It is **easy** and **simple** to implement
- ✓ It is used for **small** list of elements
- ✓ It uses less **memory**
- ✓ It is **efficient** than **bubble sort** technique
- ✓ It is **not efficient** when used with **large list** of elements

Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

- ✓ It is **not efficient** than **insertion sort** technique when used with **large** list
- ✓ The time complexity of selection sort is **$O(n^2)$**
- ✓ Consider an array **A** with **N** elements. First find the **smallest element** in the **array** and place it in the **first position**. Then, find the **second smallest element** in the **array** and place it in the **second position**. Repeat this procedure until the entire array is sorted.
- ✓ In **Pass 1**, find the position **POS** of the **smallest element** in the array and then **swap A[POS]** and **A[0]**. Thus, **A[0]** is sorted.
- ✓ In **Pass 2**, find the position **POS** of the **smallest element** in **sub-array of N–1 elements**. Swap **A[POS]** with **A[1]**. Now, **A[0]** and **A[1]** is sorted.
- ✓ In **Pass N–1**, find the position **POS** of the **smaller** of the elements **A[N–2]** and **A[N–1]**. Swap **A[POS]** and **A[N–2]** so that **A[0], A[1], …, A[N–1]** is sorted.

Algorithm for Selection Sort

Algorithm for Selection Sort

SELECTION SORT(A, N)

Step 1: Start

Step 2: Repeat Steps 3 and 4 for I = 1 to N

Step 3: Call SMALLEST(A, I, N, pos)

Step 4: Swap A[I] with A[pos]

Step 5: Stop

SMALLEST (A, I, N, pos)

Step 1: Start

Step 2: SET small = A[I]

Step 3: SET POS = I

Step 4: Repeat for J = I+1 to N

If small> A[J]

SET small = A[J]

Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SET pos = J

Step 4: Return pos

Step 5: Stop

Example for Selection Sort

1	2	3	4	5	6	7	8	9	Remarks
65	70	75	80	50	60	55	85	45	find the first smallest element
i								j	swap a[i] & a[j]
45	70	75	80	50	60	55	85	65	find the second smallest element
	i			j					swap a[i] and a[j]
45	**50**	75	80	70	60	55	85	65	Find the third smallest element
		i				j			swap a[i] and a[j]
45	**50**	**55**	80	70	60	75	85	65	Find the fourth smallest element
			i		j				swap a[i] and a[j]
45	**50**	**55**	**60**	70	80	75	85	65	Find the fifth smallest element
				i				j	swap a[i] and a[j]
45	**50**	**55**	**60**	**65**	80	75	85	70	Find the sixth smallest element
					i			j	swap a[i] and a[j]
45	**50**	**55**	**60**	**65**	**70**	75	85	80	Find the seventh smallest element
						i j			swap a[i] and a[j]
45	**50**	**55**	**60**	**65**	**70**	**75**	85	80	Find the eighth smallest element
							i	J	swap a[i] and a[j]
45	**50**	**55**	**60**	**65**	**70**	**75**	**80**	**85**	The outer loop ends.

Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Program for Selection Sort

```
# include<stdio.h>
void selection_sort( int low, int high );
int a[25];
int main(void)
{
        int n, i= 0;
        printf( "Enter the number of elements: " );
        scanf("%d", &n);
        printf( "\nEnter the elements:\n" );
        for(i=0; i < n; i++)
                scanf( "%d", &a[i] );
        selection_sort( 0, n-1 );
        printf( "\nThe elements after sorting are: " );
        for( i=0; i< n; i++ )
                printf( "%d\t ", a[i] );
        return 0;
}
void selection_sort( int low, int high )
{
        int i=0, j=0, temp=0, minindex;
        for( i=low; i <= high; i++ )
        {
                minindex = i;
                for( j=i+1; j <= high; j++ )
                {
                        if( a[j] < a[minindex] )
                                minindex = j;
                }
                temp = a[i];
                a[i] = a[minindex];
                a[minindex] = temp;
```


Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING


```
        }
}
```

BUBBLE SORT

- ✓ It is known as exchange sort
- ✓ It is also known as comparison sort
- ✓ It is easiest and simple sort technique but inefficient.
- ✓ It is not a stable sorting technique.
- ✓ The time complexity of bubble sort is $O(n^2)$ in all cases.
- ✓ Bubble sort uses the concept of passes.
- ✓ The phases in which the elements are moving to acquire their proper positions is called passes.
- ✓ It works by comparing adjacent elements and bubbles the largest element towards right at the end of the first pass.
- ✓ The largest element gets sorted and placed at the end of the sorted list.
- ✓ This process is repeated for all pairs of elements until it moves the largest element to the end of the list in that iteration.
- ✓ Bubble sort consists of (n-1) passes, where n is the number of elements to be sorted.
- ✓ In 1st pass the largest element will be placed in the nth position.
- ✓ In 2nd pass the second largest element will be placed in the (n-1)th position.
- ✓ In (n-1)th pass only the first two elements are compared.

Algorithm for Bubble Sort

BUBBLE_SORT(A, N)

Step 1: Repeat Step 2 For I = to N-1

Step 2: Repeat For J = to N - I

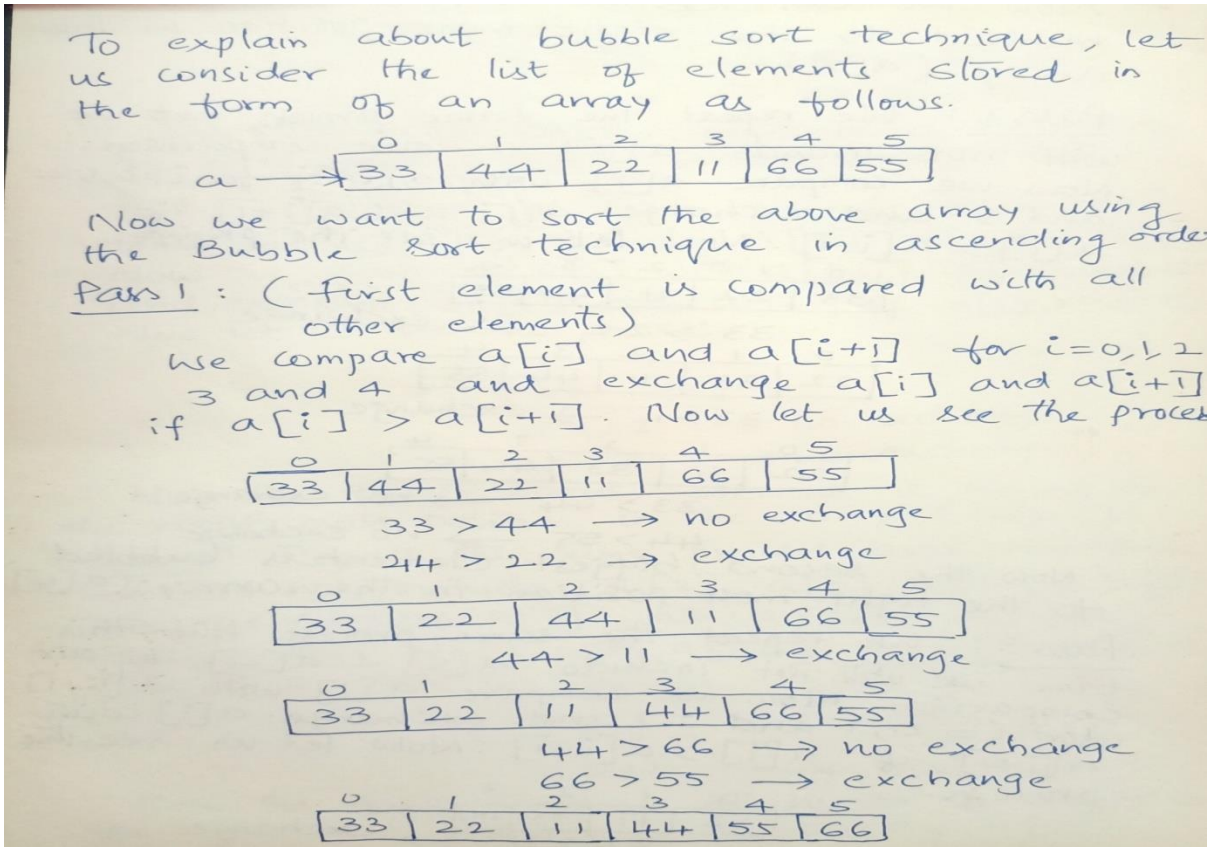
Step 3: IF A[J] > A[J + 1]

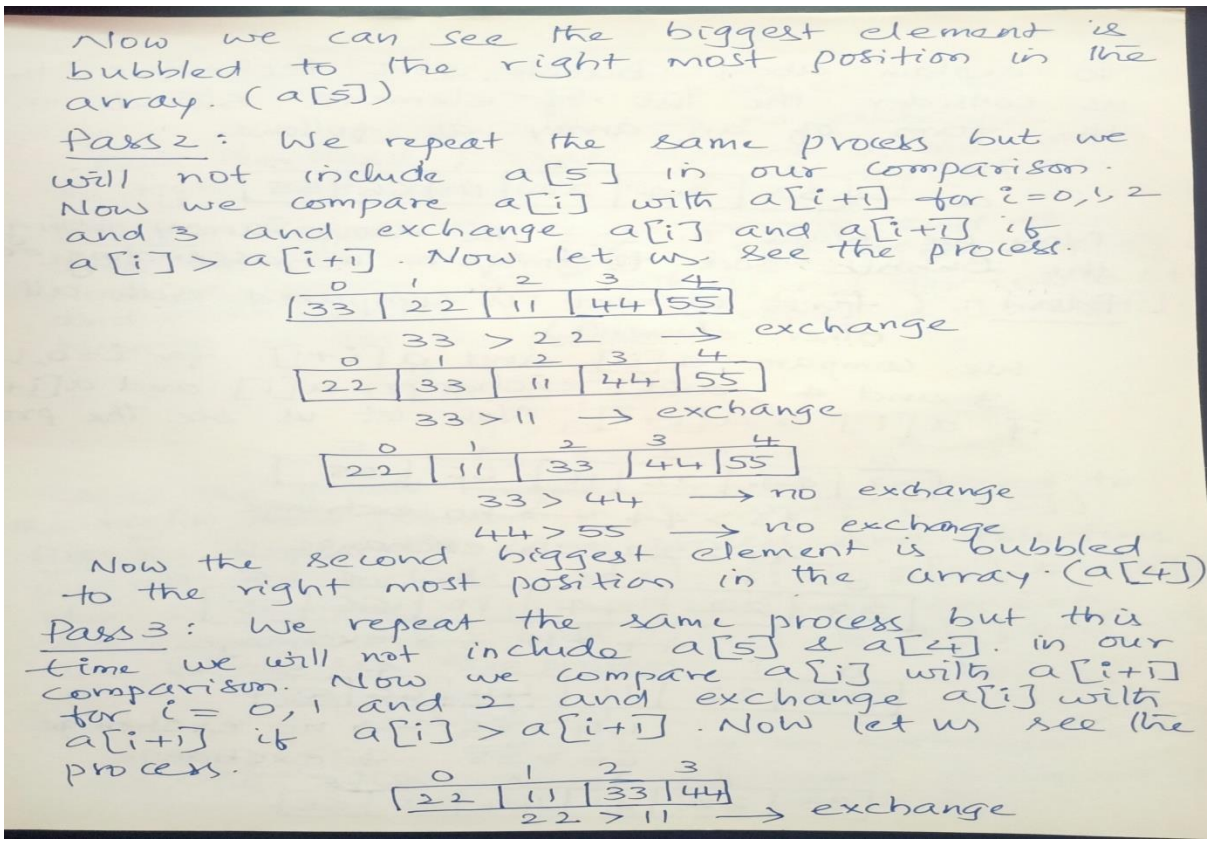
SWAP A[J] and A[J+1]

Step 4: EXIT

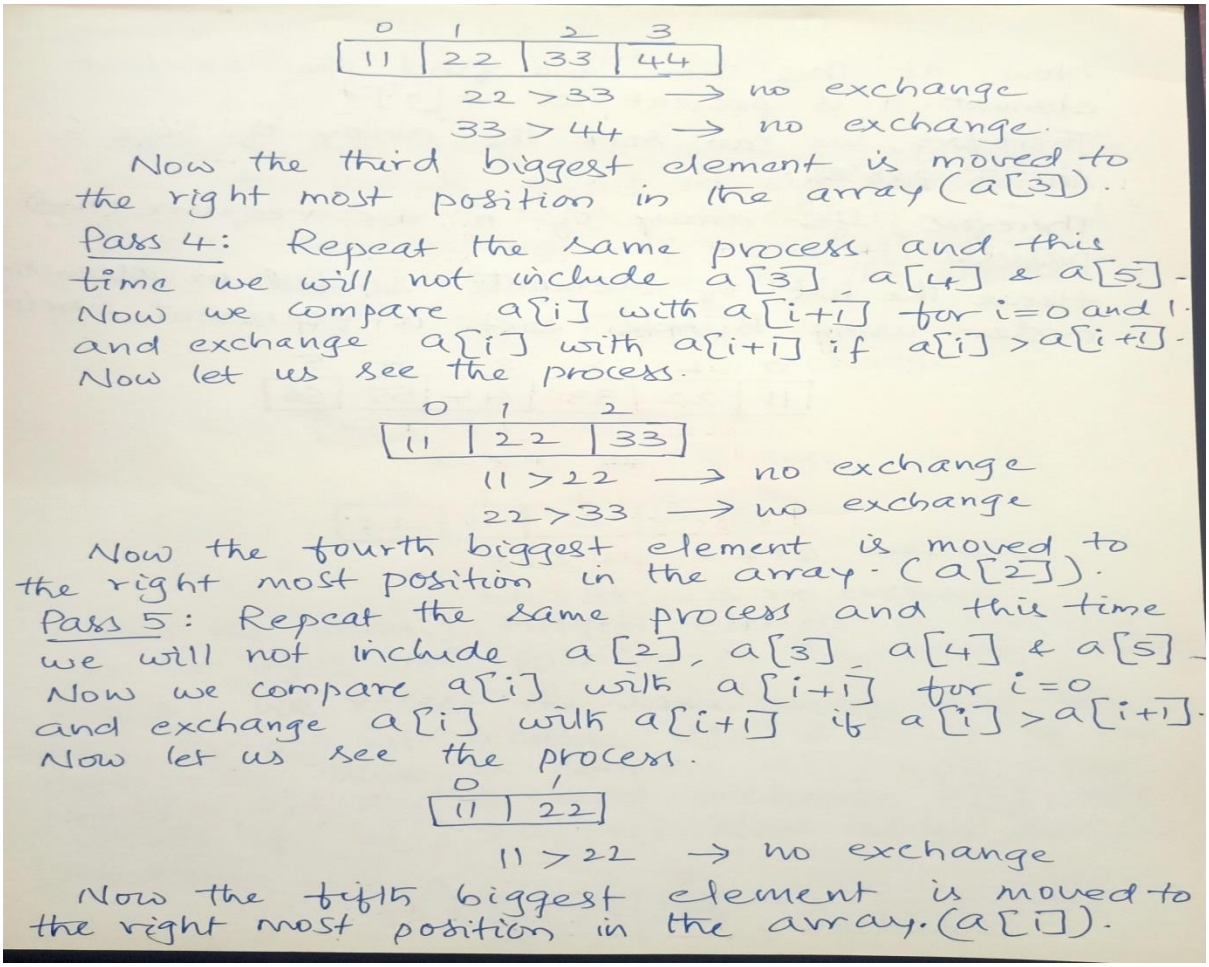
Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Example for Bubble Sort





Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

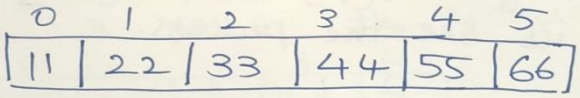


Now, at this time we find the smallest element 11 is present at a[0].

Therefore, we can sort the array of size 6 in 5 passes.

Therefore, for array of "n", we require (n-1) passes.

Hence the list of elements sorted in ascending order using Bubble sort is represented below



=

Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Program for Bubble Sort

```
#include<stdio.h>
void bubble_sort(int [], int);
int main(void)
{
        int n, a[10], i;
        clrscr();
        printf(" Enter the size of the array ");
        scanf("%d", &n);
        printf(" Enter the elements of the array ");
        for(i=0; i<n; i++)
                scanf("%d", &a[i]);
        bubble_sort(a,n);
        return 0;
}
void bubble_sort(int a[], int n)
{
        int i, j, m, temp;
        for(i=1; i<n-1; i++)
        {
                for(j=0; j<n; j++)
                {
                        if(a[j] > a[j+1])
                        {
                                temp = a[j];
                                a[j] = a[j+1];
                                a[j+1] = temp;
                        }
                }
        }
        printf(" The sorted list of elements are ");
        for(i=0; i<n; i++)
```


Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING


```
            printf("%d\t", a[i]);
}
```

QUICK SORT

- ✓ It is developed by **C.A.R. Hoare**.
- ✓ It is also known as **partition exchange sort.**
- ✓ This sorting algorithm uses **divide** and **conquer** strategy.
- ✓ In this method, the **division** is carried out **dynamically**.
- ✓ It contains three steps:
- ✓ **Divide –** split the array into two sub arrays so that each element in the right sub array is greater than the middle element and each element in the left sub array is less than the middle element. The splitting is done based on the middle element called **pivot**. All the elements **less** than **pivot** will be in the left sub array and all the elements **greater** than **pivot** will be on right sub array.
- ✓ **Conquer –** recursively sort the two sub arrays.
- ✓ **Combine –** combine all the sorted elements in to a single list.
- ✓ Consider an array **A[i]** where **i** is ranging from **0 to n – 1** then the division of elements is as follows:

A[0]……A[m – 1], A[m], A[m + 1] …….A[n]

- ✓ The **partition algorithm** is used to arrange the elements such that all the elements are **less** than **pivot** will be on left sub array and **greater** then **pivot** will be on right sub array.
- ✓ The time complexity of quick sort algorithm in worst case is **$O(n^2)$**, best case and average case is **O(n log n)**.
- ✓ It is **faster** than other sorting techniques whose time complexity is **O(n log n)**

Algorithm for Quick Sort

```
QUICK_SORT (A, LOW, HIGH)
Step 1: IF (LOW < HIGH)
            CALL PARTITION (A, LOW, HIGH, MID)
```


Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CALL QUICKSORT(A, LOW, MID - 1)

CALL QUICKSORT(A, MID + 1, HIGH)

Step 2: EXIT

Algorithm for Partition

PARTITION (A, LOW, HHIGH, MID)

Step 1: SET PIVOT = A[LOW], I =LOW, J = HIGH

Step 2: Repeat Steps 3 to 5 while I <= LOW

Step 3: Repeat while A[LOW] <= A[PIVOT]

SET I = I + 1

Step 4: Repeat while A[j] >= PIVOT

SET J = J – 1

Step 5: Repeat if I <= J

SWAP A[I], A[J]

Step 6: SWAP A[LOW], A[J]

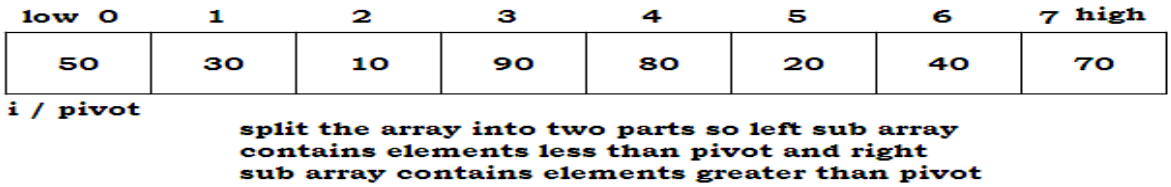
Step 7: Return J

Step 8: EXIT

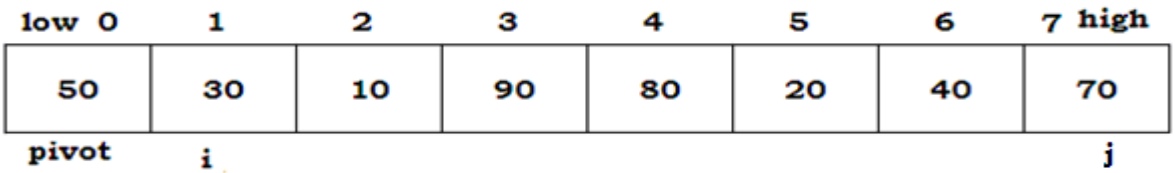
Example for Quick Sort

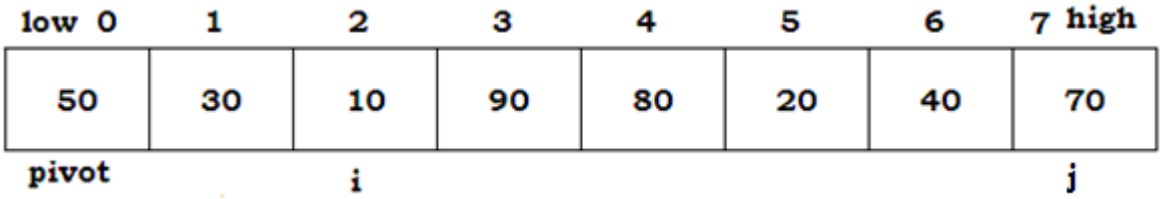
Let us consider the array of elements to sort them using quick sort technique

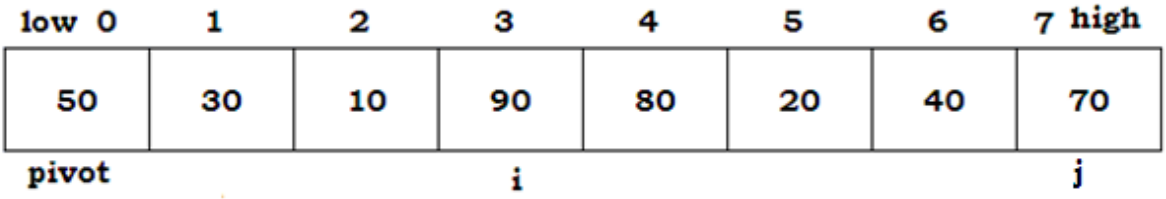
50, 30, 10, 90, 80, 20, 40, 70



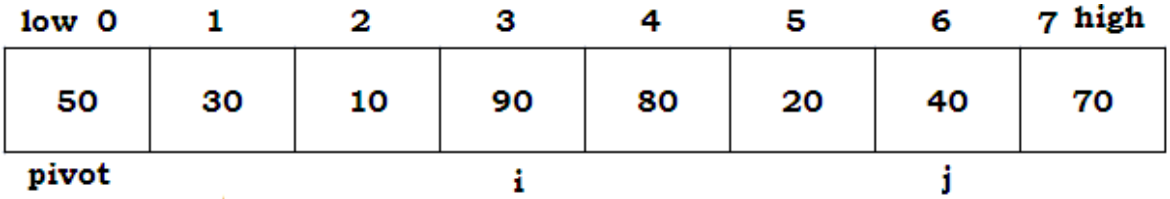
We will increment i, if(a[i] <= pivot), we will continue incrementing i until the condition is false.



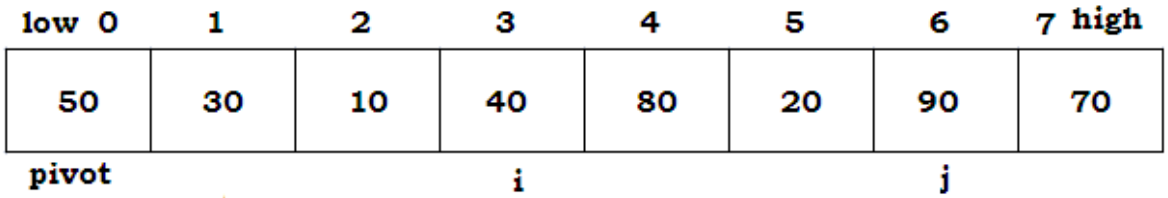




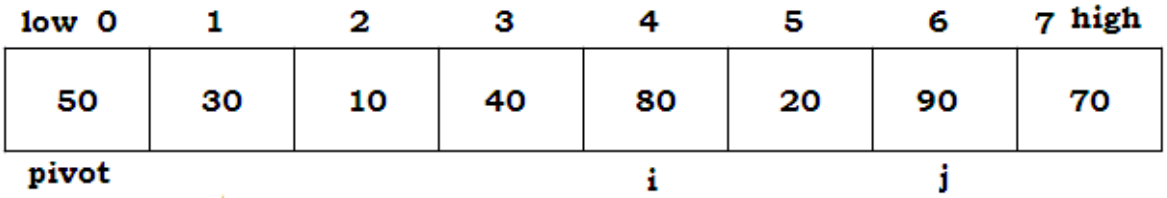
Now a[i] > pivot, so stop incrementing i. As a[j] > pivot we will decrement j until it becomes false



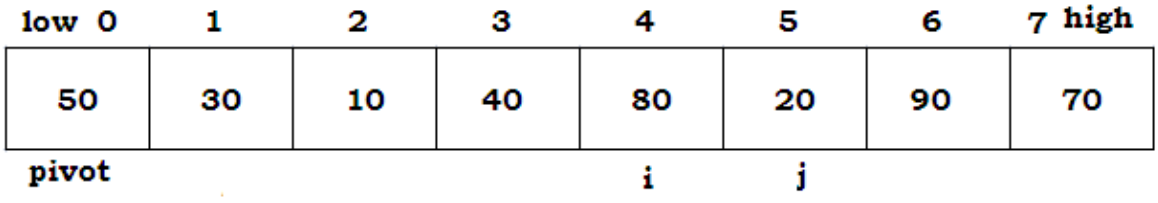
Now we cannot decrement j because a[j] < pivot. Hence we swap a[i] and a[j] since i < j.



Now again a[i] < pivot so increment i

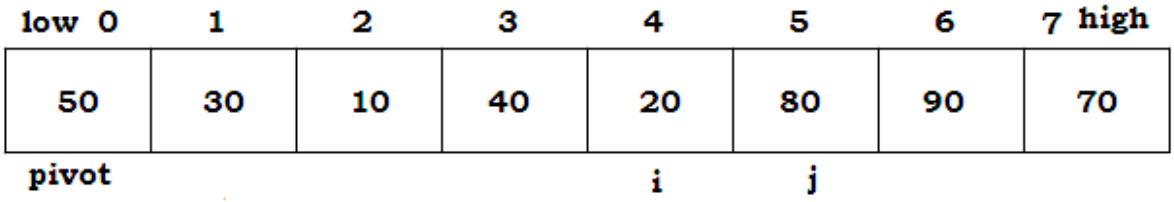


Now a[i] > pivot so stop incrementing i and a[j] > pivot so decrement j

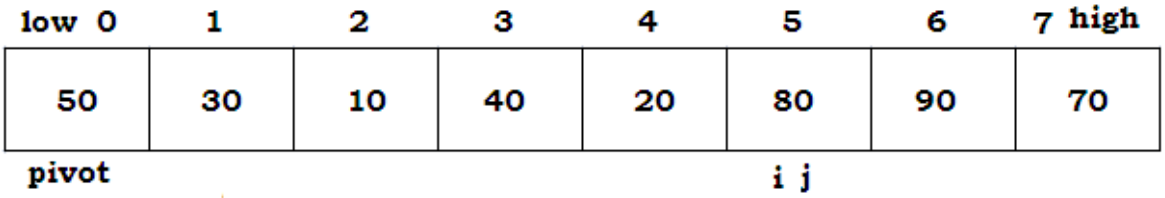


Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Now a[j] < pivot so stop decrementing j. since i <j swap a[i] and a[j]

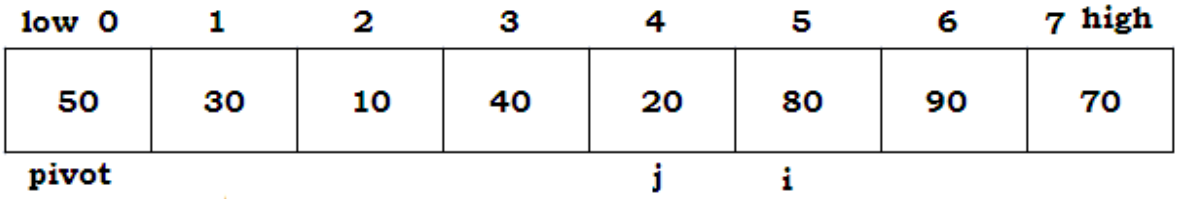


Now again a[i] < pivot so increment i

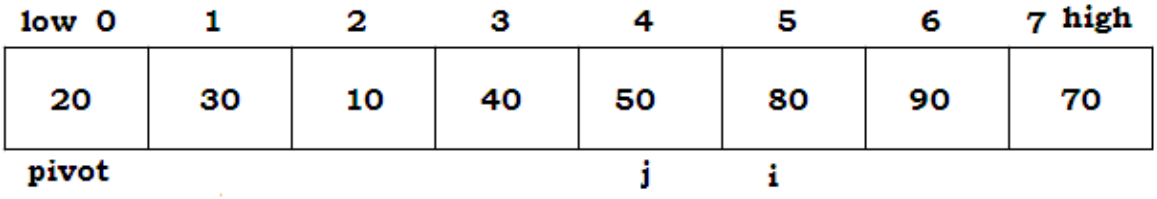


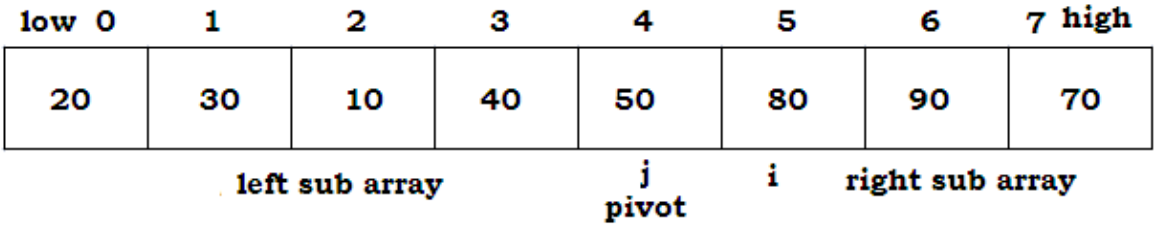
Now a[i] > pivot, so stop incrementing i. As a[j] > pivot we will decrement j until it becomes false

.

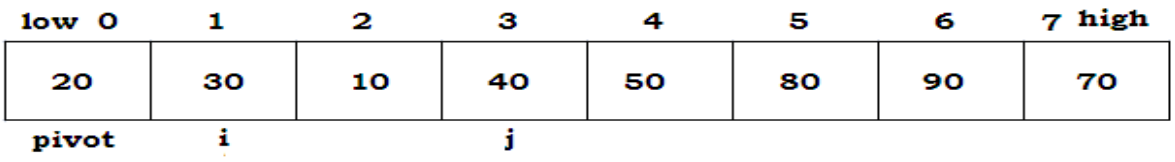


As a[i] > pivot and a[j] < pivot and j crossed i we will swap a[low] and a[j]



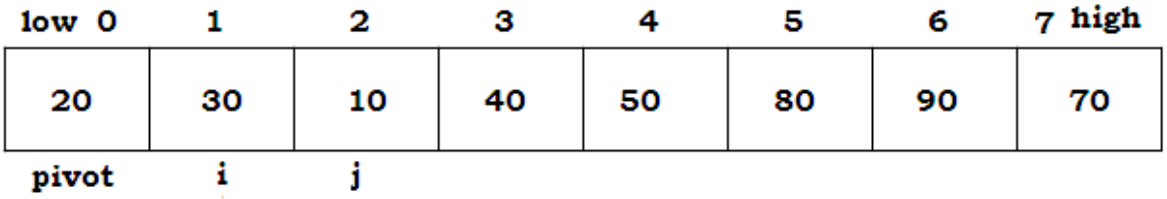


We will now start left array to be sorted and then right sub array. The new pivot for the left sub array is 20

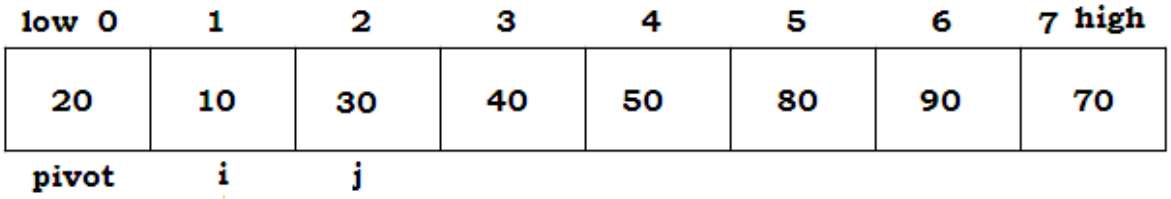


Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

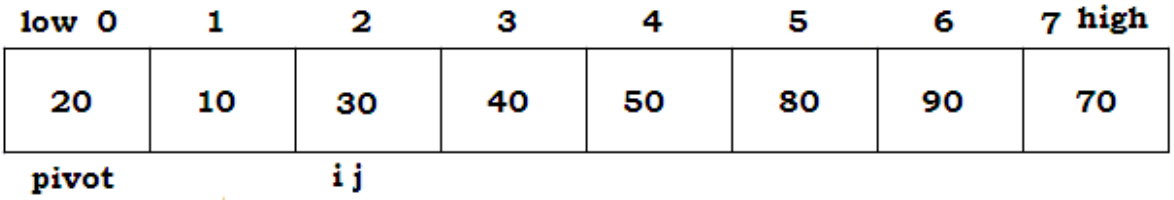
Now since a[i] > pivot stop incrementing i and a[j] > pivot so decrement j



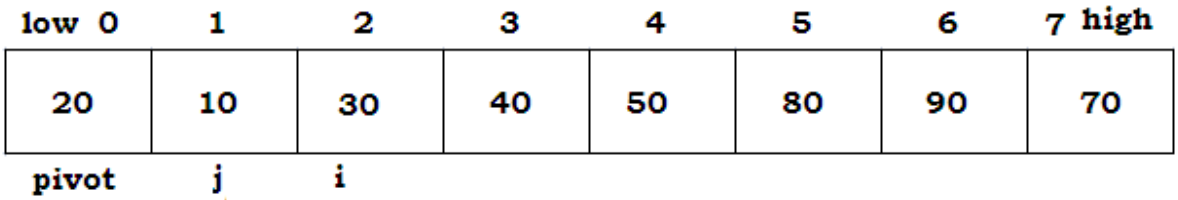
Now j cannot be decremented and i < j so swap a[i] and a[j]



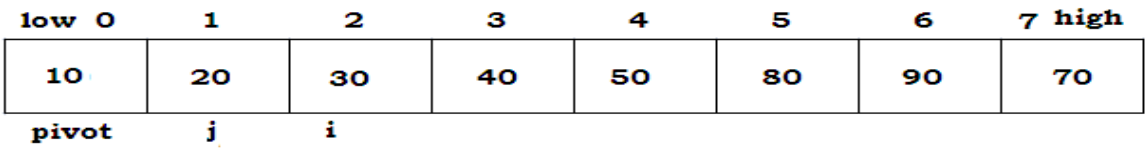
Now again a[i] < pivot so increment i



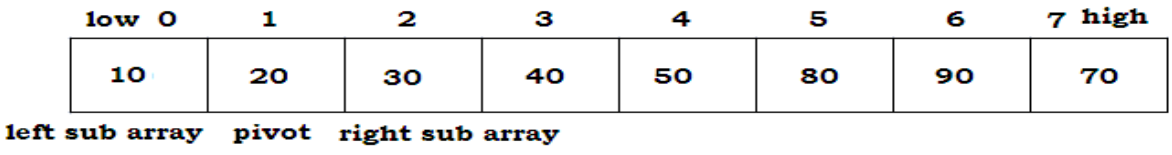
Now a[i] > pivot so stop incrementing i and a[j] > pivot so decrement j

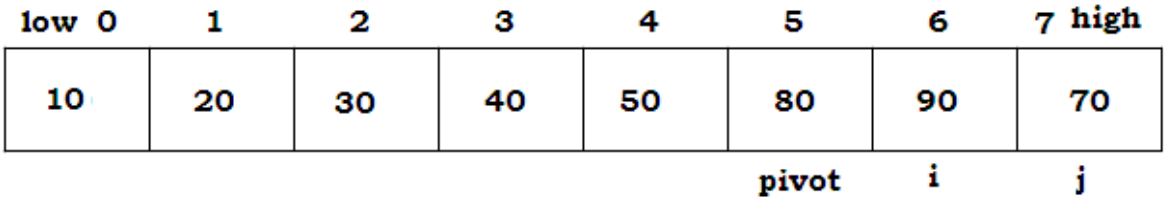


Since a[j] < pivot so j cannot be decremented and j crossed i so swap a[low] and a[j]

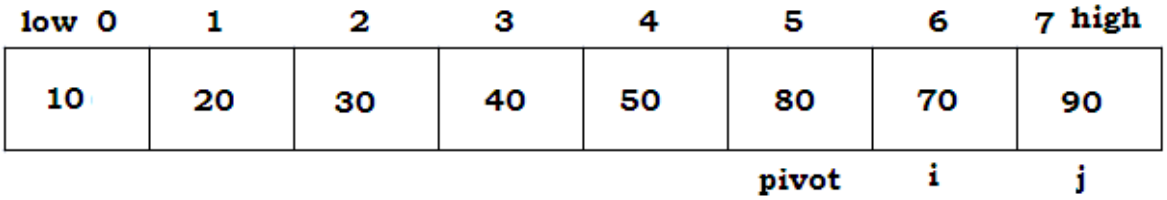


There is one element in the left sub array hence all the elements in the right sub array is to be sorted.

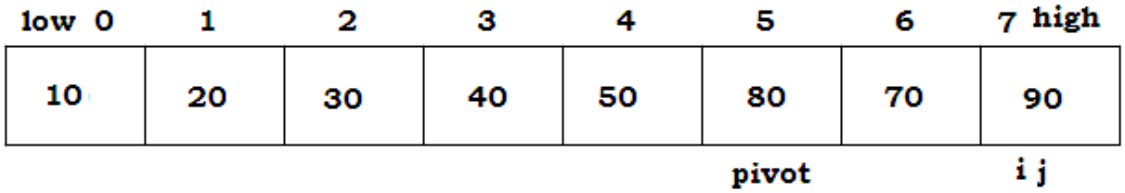




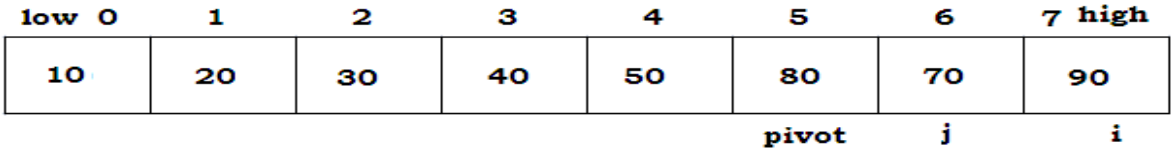
Since a[i] > pivot and a[j] < pivot we stop incrementing I and decrementing j and I < j we swap a[i] and a[j]



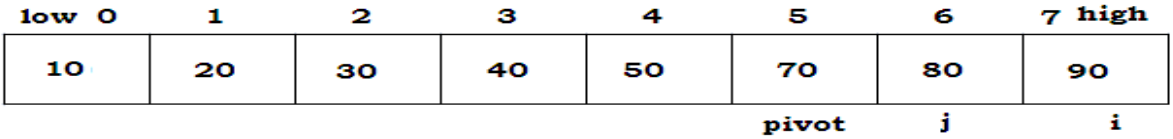
Since a[i] < pivot so increment i

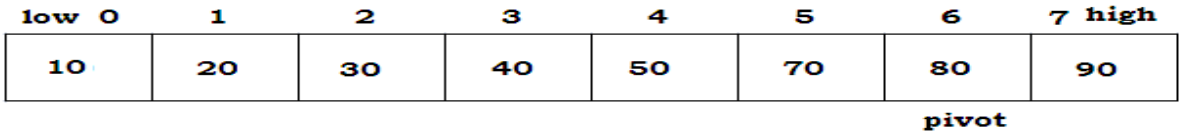


Since a[i] > pivot so stop incrementing i and a[j] > pivot so decrement j

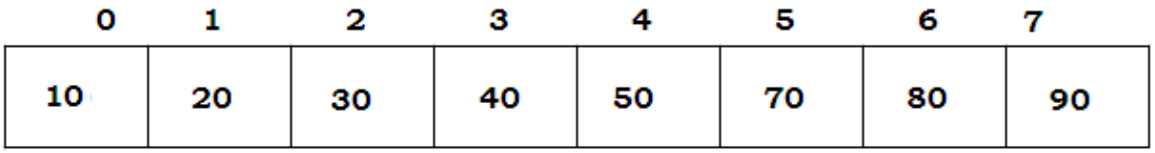


Since a[j] < pivot so j cannot be decremented and j crossed i so swap a[low] and a[j]





Now the left contains 70 and right contains 90 we cannot further subdivide the array. Hence if we look at the array all the elements are sorted.



Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Program for Quick Sort

```
#include <stdio.h>
#define size 100
int partition(int a[], int low, int high);
void quick_sort(int a[], int low, int high);
int main(void)
{
        int a[size], i, n;
        printf("\n Enter the number of elements in the array: ");
        scanf("%d", &n);
        printf("\n Enter the elements of the array: ");
        for(i=0;i<n;i++)
        {
                scanf("%d", &a[i]);
        }
        quick_sort(a, 0, n-1);
        printf("\n The sorted array is: \n");
        for(i=0;i<n;i++)
                printf(" %d\t", a[i]);
        return 0;
}
int partition(int a[], int low, int high)
{
        int left, right, temp, mid, flag;
        mid = left = low;
        right = high;
        flag = 0;
        while(flag != 1)
        {
                while((a[mid] <= a[right]) && (mid!=right))
                        right--;
                if(mid==right)
```


Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING


```
                flag =1;
        else if(a[mid]>a[right])
        {
                temp = a[mid];
                a[mid] = a[right];
                a[right] = temp;
                mid = right;
        }
        if(flag!=1)
        {
                while((a[mid] >= a[left]) && (mid!=left))
                        left++;
                if(mid==left)
                        flag =1;
                else if(a[mid] <a[left])
                {
                        temp = a[mid];
                        a[mid] = a[left];
                        a[left] = temp;
                        mid = left;
                }
        }
    }
    return mid;
}
void quick_sort(int a[], int low, int high)
{
    int mid;
    if(low<high)
    {
        mid = partition(a, low, high);
        quick_sort(a, low, mid -1);
```

```
            quick_sort(a, mid+1, high);
        }
}
```

RADIX SORT

- ✓ It is a linear sorting algorithm.
- ✓ It is also known as bucket sort technique or binsort technique or card sort technique since it uses buckets for sorting.
- ✓ It can be applied for integers as well as letters. For integers it used 10 buckets and for letters it uses 26 buckets.
- ✓ If the input is integers then we sort them from least significant digit to most significant digit.
- ✓ The number passes used in radix sort depends on the number of digits.
- ✓ The time complexity of radix sort in all cases is O(n log n)
- ✓ It takes more space compared to other sorting algorithms.
- ✓ It is used only for digits and letters
- ✓ It depends on the number of digits and letters.

Algorithm for Radix Sort

RadixSort (A, N)

Step 1: Find the largest number in A as LARGE

Step 2: SET NOP = Number of digits in LARGE

Step 3: SET PASS = 0

Step 4: Repeat Step 5 while PASS <= NOP-1

Step 5: SET I = 0 and INITIALIZE buckets

Step 6: Repeat Steps 7 to 9 while I<N-1

Step 7: SET DIGIT = digit at PASSth place in A[I]

Step 8: Add A[I] to the bucket numbered DIGIT

Step 9: INCEREMENT bucket count for bucket numbered DIGIT

Step 10: Collect the numbers in the bucket

Step 11: EXIT

Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Example for Radix Sort

Let us consider the array of elements to sort them using radix sort technique

345, 654, 924, 123, 567, 472, 555, 808, 911

In the first pass, the numbers are sorted according to the digit at one's place

Number	0	1	2	3	4	5	6	7	8	9
345						345				
654					654					
924					924					
123				123						
567								567		
472			472							
555						555				
808									808	
911		911								

After the first pass the numbers are collected bucket by bucket. Thus the new list for the second pass is

911, 472, 123, 654, 924, 345, 555, 567, 808

In the second pass the numbers are sorted according to the digit at ten's place.

Number	0	1	2	3	4	5	6	7	8	9
911		911								
472								472		
123			123							
654						654				
924			924							
345					345					
555						555				
567							567			
808	808									

Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

After the second pass the numbers are collected bucket by bucket. Thus the new list for the third pass is

808, 911, 123, 924, 345, 654, 555,567, 472

In the third pass the numbers are sorted according to the digit at hundred place.

Number	0	1	2	3	4	5	6	7	8	9
808									808	
911										911
123		123								
924										924
345				345						
654							654			
555						555				
567						567				
472					472					

After the third pass the numbers are collected bucket by bucket. Thus the new list formed is the final result. It is

123, 345, 472, 555, 567, 654, 808, 911, 924

Program for Radix Sort

```
#include<stdio.h>
int main(void)
{
    int a[100][100], i, n, r=0, c=0, b[100], temp;
    printf(" Enter the size of the array ");
    scanf("%d", &n);
    for(r=0;r<100;r++)
    {
        for(c=0;c<100;c++)
            a[r][c] = 1000;
    }
```


Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

```
        printf(" Enter the array elements ");
        for(i=0;i<n;i++)
        {
                scanf("%d", &b[i]);
                r = b[i] /100;
                c = b[i] % 100;
                a[r][c] = b[i];
        }
        for(r=0;r<100;r++)
        {
                for(c=0;c<100;c++)
                {
                        for(i=0;i<n;i++)
                        {
                                if(a[r][c] =  =b[i])
                                {
                                        printf("\n\t");
                                        printf("%d", a[r][c]);
                                }
                        }
                }
        }
        return 0;
}
```

MERGE SORT

- ✓ This sorting algorithm uses **divide** and **conquer** strategy.
- ✓ In this method, the division is carried out **dynamically**.
- ✓ It contains three steps:
- ✓ **Divide –** split the array into two sub arrays **s1** and **s2** with each **n/2** elements. If **A** is an array containing **zero** or **one** element, then it is already sorted. But if there are more elements in the array, divide **A**

into two sub-arrays, **s1** and **s2**, each containing half of the elements of **A**.

- ✓ **Conquer –** sort the two sub arrays **s1** and **s2**.
- ✓ **Combine –** combine or merge **s1** and **s2** elements into a unique sorted list.
- ✓ The time complexity of merge sort is **O(n log n)** in all cases.

Algorithm for Merge Sort

```
MERGE_SORT(A,LOW, HIGH)
Step 1: IF LOW < HIGH
            SET MID = (LOW +HIGH)/2
            CALL MERGE_SORT (A, LOW, MID)
            CALL MERGE_SORT (A, MID + 1, HIGH)
            COMBINE (A, LOW, MID, HIGH)
Step 2: EXIT
```

Algorithm for Combine

```
COMBINE (A, LOW, MID, HIGH)
Step 1: SET I = LOW, J = MID + 1, INDEX = LOW
Step 2: Repeat while (I <= MID) AND (J<=HIGH)
                IF A[I] < A[J]
                        SET TEMP[INDEX] = A[I]
                        SET I = I + 1
                        SET INDEX = INDEX + 1
                ELSE
                        SET TEMP[INDEX] = A[J]
                        SET J = J + 1
                        SET INDEX = INDEX + 1

Step 3: [Copy the remaining elements of right sub-array, if any]
        IF I > MID
        Repeat while J <= HIGH
```

SET TEMP[INDEX] = A[J]

SET J = J + 1

SET INDEX = INDEX + 1

[Copy the remaining elements of left sub-array, if any]

ELSE

IF A[I]<= MID

SET TEMP[INDEX] = A[I]

SET I = I + 1

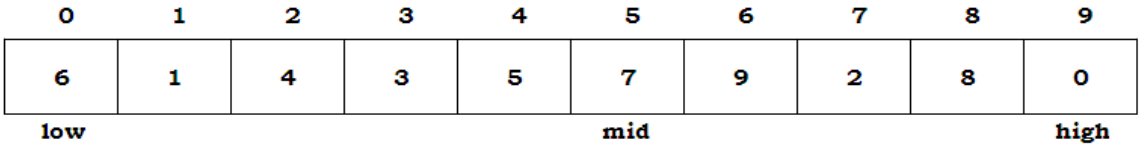
SET INDEX = INDEX + 1

Step 4: EXIT

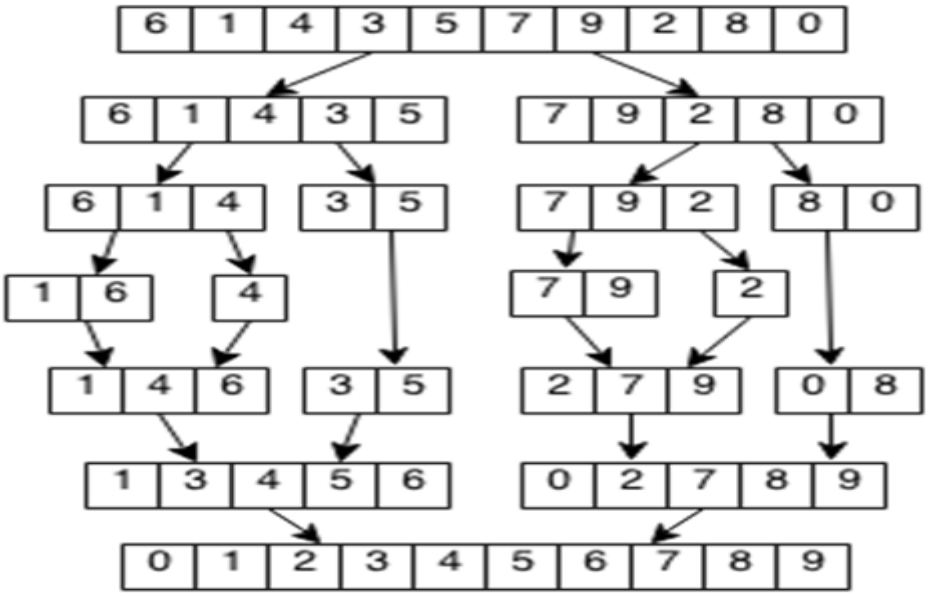
Example for Merge Sort

Let us consider the array of elements to sort them using Merge sort technique

6, 1, 4, 3, 5, 7, 9, 2, 8, 0



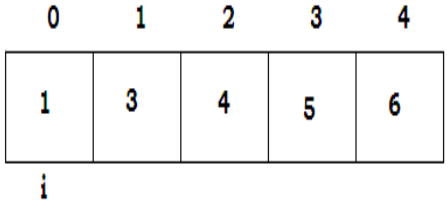
We then first make the two sublists and combine the two sorted sublists as a unique sorted list.

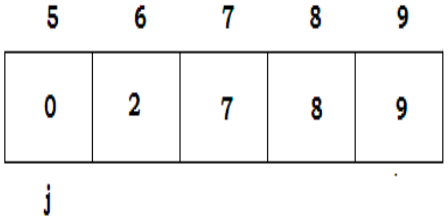


Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Now let us see the combine operations

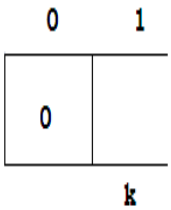
1, 3, 4, 5, 6, 0, 2, 7, 8, 9



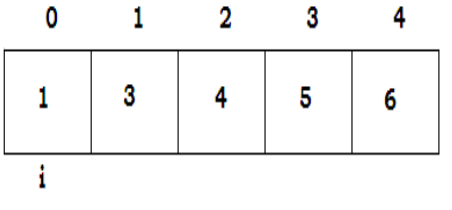


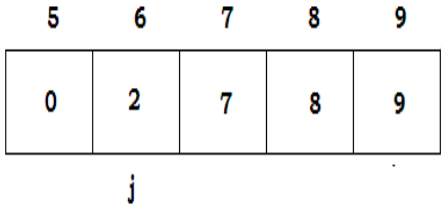

```
if(a[i] <= a[j])
{
      temp[k] = a[i];
      k++;
      i++;
}
else
{
      temp[k] = a[j];
      k++;
      j++;
}
```

Initially k = 0. Then k will be increment



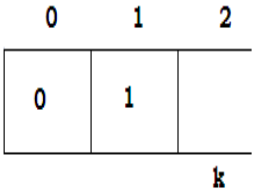
Now i remains there and j is incremented.



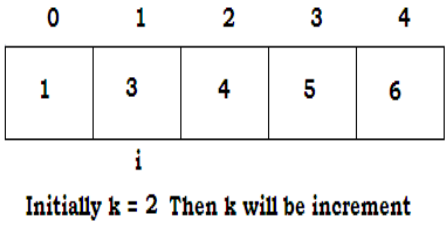


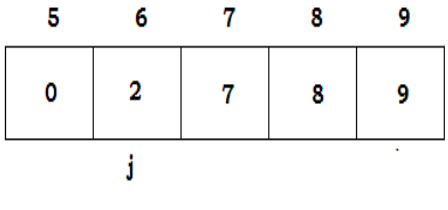

```
if(a[i] <= a[j])
{
      temp[k] = a[i];
      k++;
      i++;
}
else
{
      temp[k] = a[j];
      k++;
      j++;
}
```

Initially k = 1 Then k will be increment



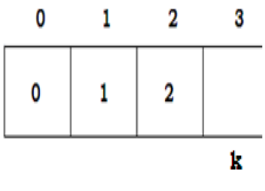
Now j remains there and i is incremented.





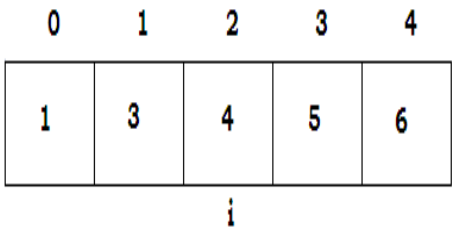

```
if(a[i] <= a[j])
{
      temp[k] = a[i];
      k++;
      i++;
}
else
{
      temp[k] = a[j];
      k++;
      j++;
}
```

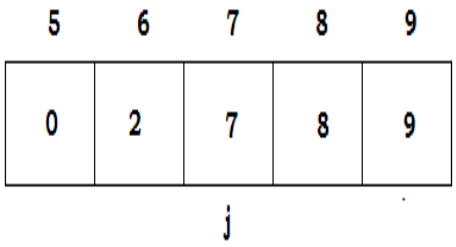
Initially k = 2 Then k will be increment



Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

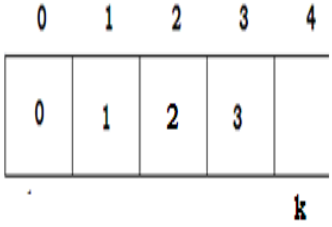
Now i remains there and j is incremented.



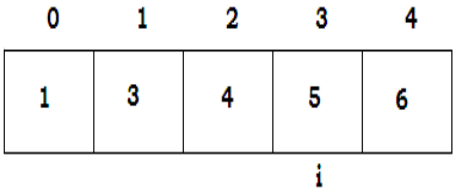


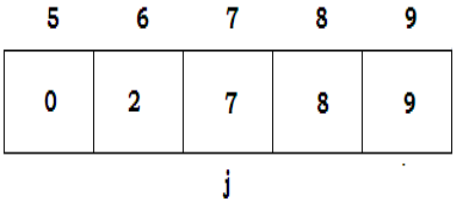

```
if(a[i] <= a[j])
{
        temp[k] = a[i];
        k++;
        i++;
}
else
{
        temp[k] = a[j];
        k++;
        j++;
}
```

Initially k = 3 Then k will be increment



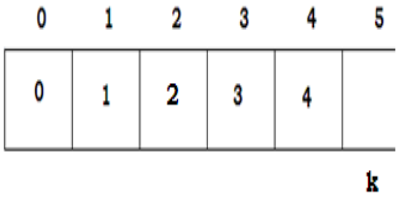
Now j remains there i is incremented



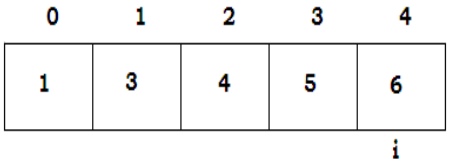


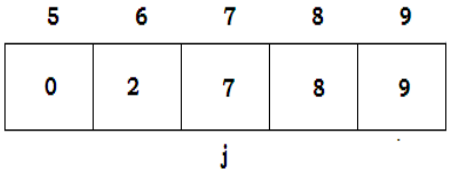

```
if(a[i] <= a[j])
{
        temp[k] = a[i];
        k++;
        i++;
}
else
{
        temp[k] = a[j];
        k++;
        j++;
}
```

Initially k = 4 Then k will be increment



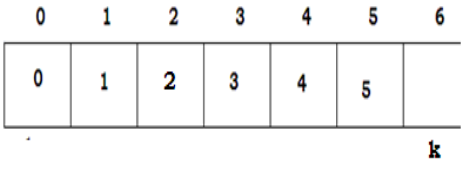
Now again i is incremented





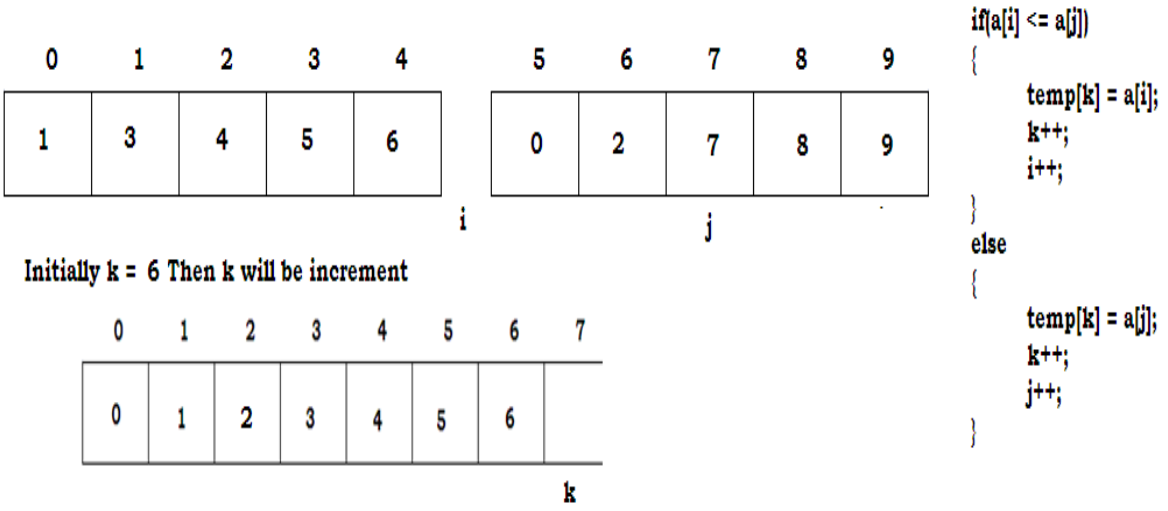

```
if(a[i] <= a[j])
{
        temp[k] = a[i];
        k++;
        i++;
}
else
{
        temp[k] = a[j];
        k++;
        j++;
}
```

Initially k = 5 Then k will be increment

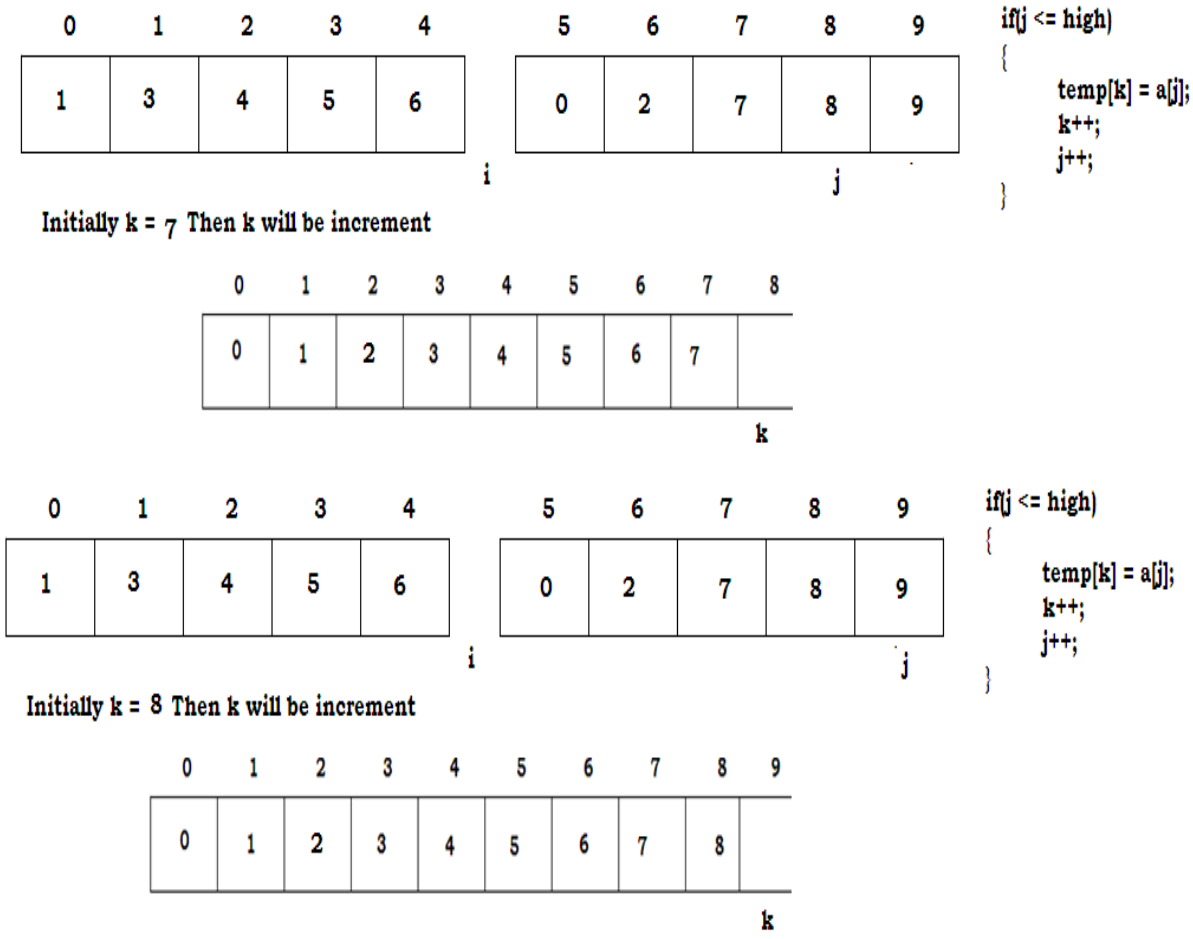


Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

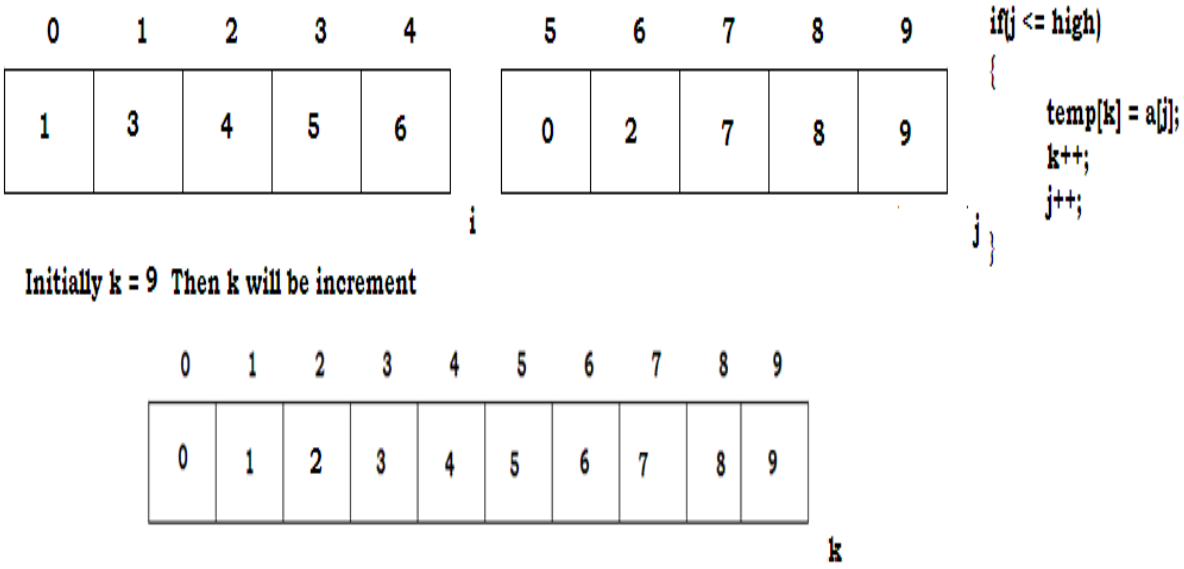
Now again i is incremented.



Now again i is incremented. But the left sub list is completed then j is incremented until the right sub list is completed



Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



Finally we see the array is in sorted order.

Program for Merge Sort

```
#include <stdio.h>
#define size 100
void combine(int a[], int, int, int);
void merge_sort(int a[],int, int);
int main(void)
{
        int a[size], i, n;
        printf("\n Enter the number of elements in the array : ");
        scanf("%d", &n);
        printf("\n Enter the elements of the array: ");
        for(i=0;i<n;i++)
                scanf("%d", &a[i]);
        merge_sort(a, 0, n-1);
        printf("\n The sorted array is: \n");
        for(i=0;i<n;i++)
                printf(" %d\t", a[i]);
        return 0;
}
```

```
void combine(int a[], int low, int mid, int high)
{
        int i=low, j=mid+1, index=low, temp[size], k;
        while((i<=mid) && (j<=high))
        {
                if(a[i] < a[j])
                {
                        temp[index] = a[i];
                        i++;
                }
                else
                {
                        temp[index] = a[j];
                        j++;
                }
                index++;
        }
        if(i>mid)
        {
                while(j<=high)
                {
                        temp[index] = a[j];
                        j++;
                        index++;
                }
        }
        else
        {
                while(i<=mid)
                {
                        temp[index] = a[i];
                        i++;
```

```
                index++;
            }
        }
        for(k=low;k<index;k++)
            a[k] = temp[k];
}
void merge_sort(int a[], int low, int high)
{
        int mid;
        if(low<high)
        {
            mid = (low+high)/2;
            merge_sort(a, low, mid);
            merge_sort(a, mid+1, high);
            combine(a, low, mid, high);
        }
}
```

HEAP SORT

- ✓ Heap is a complete binary tree and also a Max(Min) tree.
- ✓ A Max(Min) tree is a tree whose root value is larger(smaller) than its children.
- ✓ This sorting technique has been developed by J.W.J. Williams.
- ✓ It is working under two stages.
  - Heap construction
  - Deletion of a Maximum element key
- ✓ The heap is first constructed with the given numbers. The maximum key value is deleted for n -1 times to the remaining heap. Hence we will get the elements in decreasing order. The elements are deleted one by one and stored in the array from last to first. Finally we get the elements in ascending order.
- ✓ The important points about heap sort technique are:

- ✓ The time complexity of heap sort is O(n log n)
- ✓ This is a in-place sorting algorithm.
- ✓ For random input it works slower than quick sort
- ✓ Heap sort is not a stable sorting method
- ✓ The space complexity of heap sort is O(1).

Algorithm Heap Sort

- ✓ Build a max heap from the input data.
- ✓ At this point, the largest item is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of heap by 1. Finally, heapify the root of tree.
- ✓ Repeat above steps while size of heap is greater than 1

Procedure for Working of Heap Sort

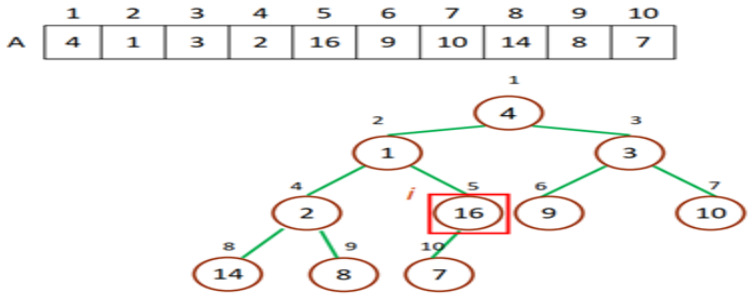
Initially on receiving an unsorted list,

- ✓ First step in heap sort is to build Max-Heap.
- ✓ Repeat Second, Third and Fourth steps, until we have the complete sorted list in our array.
- ✓ Second step - Once heap is built, the first element of the Heap is largest, so we exchange first and last element of a heap.
- ✓ Third step - We delete and put last element(largest) of the heap in our array.
- ✓ Fourth step - Then we again make heap using the remaining elements, to again get the largest element of the heap and put it into the array. We keep on doing the same repeatedly until we have the complete sorted list in our array.

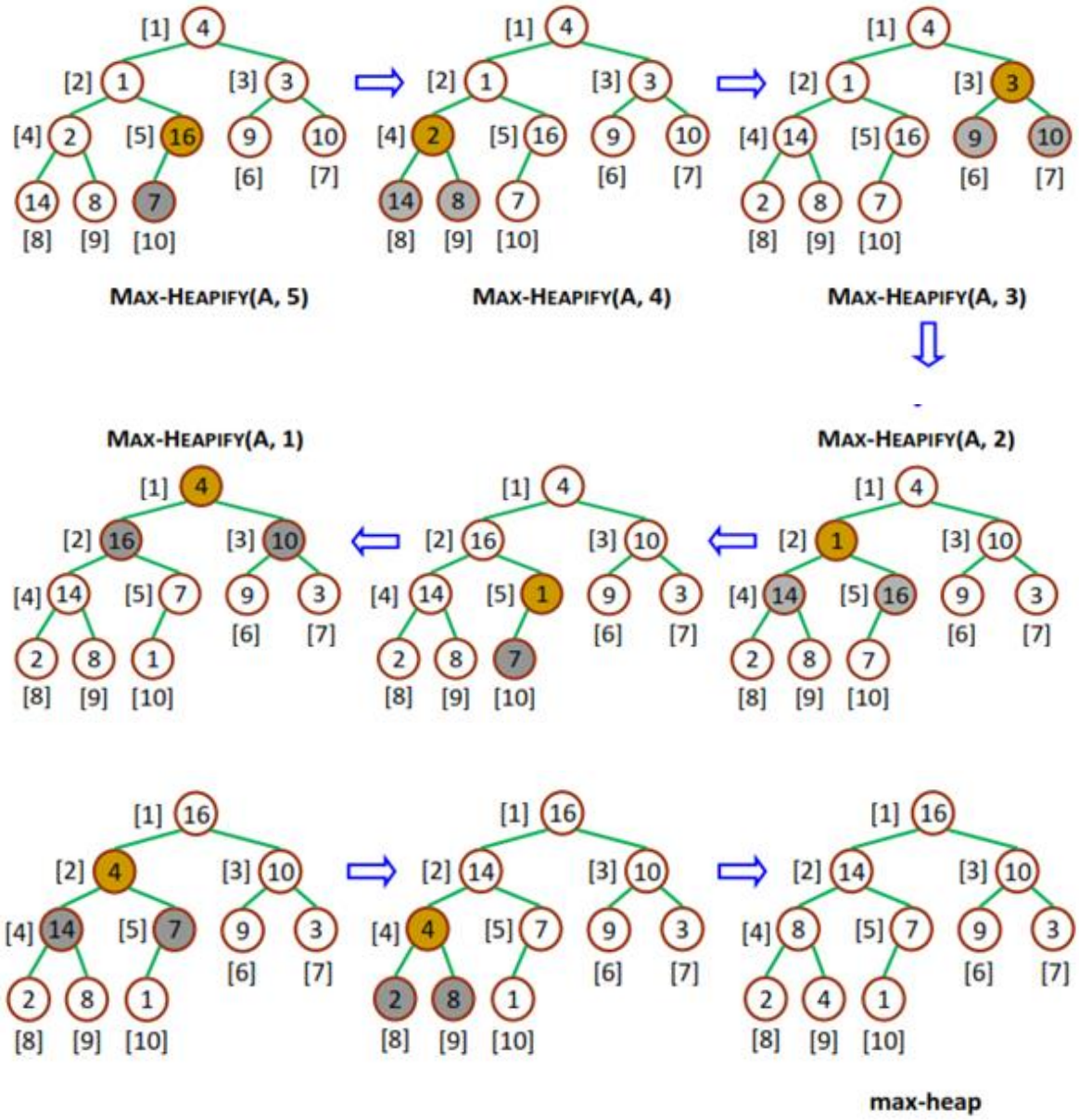
Example for Heap Sort

Let us consider the array of elements to sort them using heap sort technique

4, 1, 3, 2, 16, 9, 10, 14, 8, 7

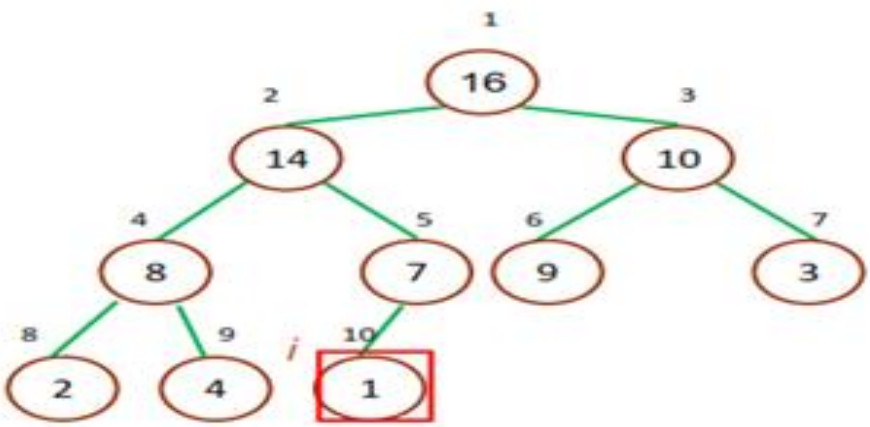


Stage -1 construction of heap

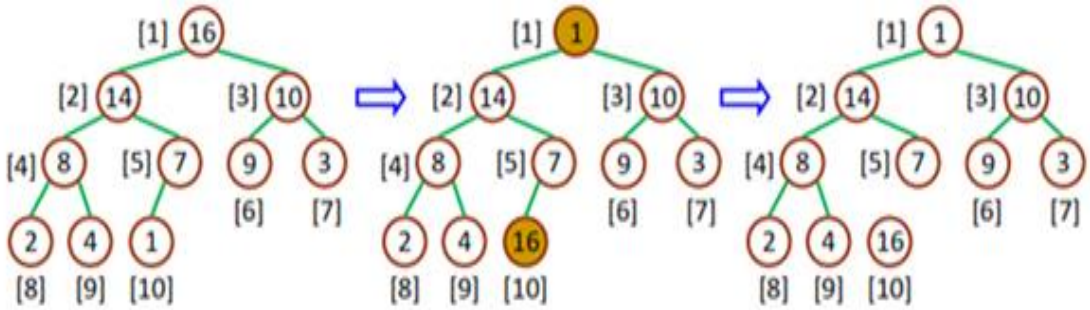


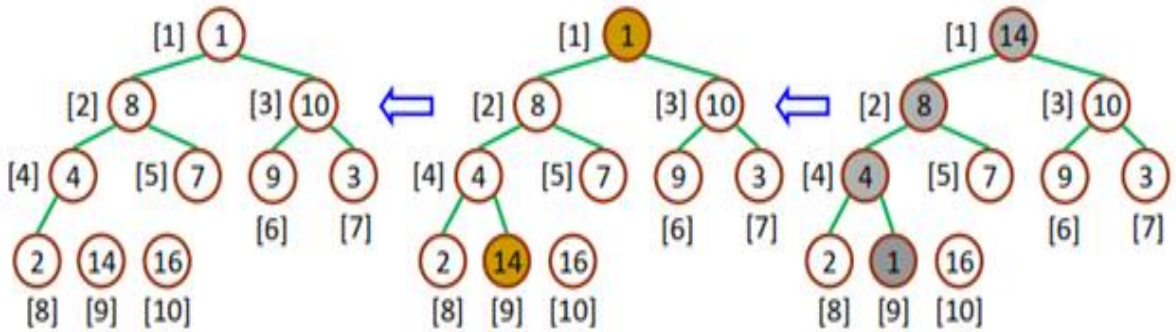
Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Stage – 2 deletion of maximum key element

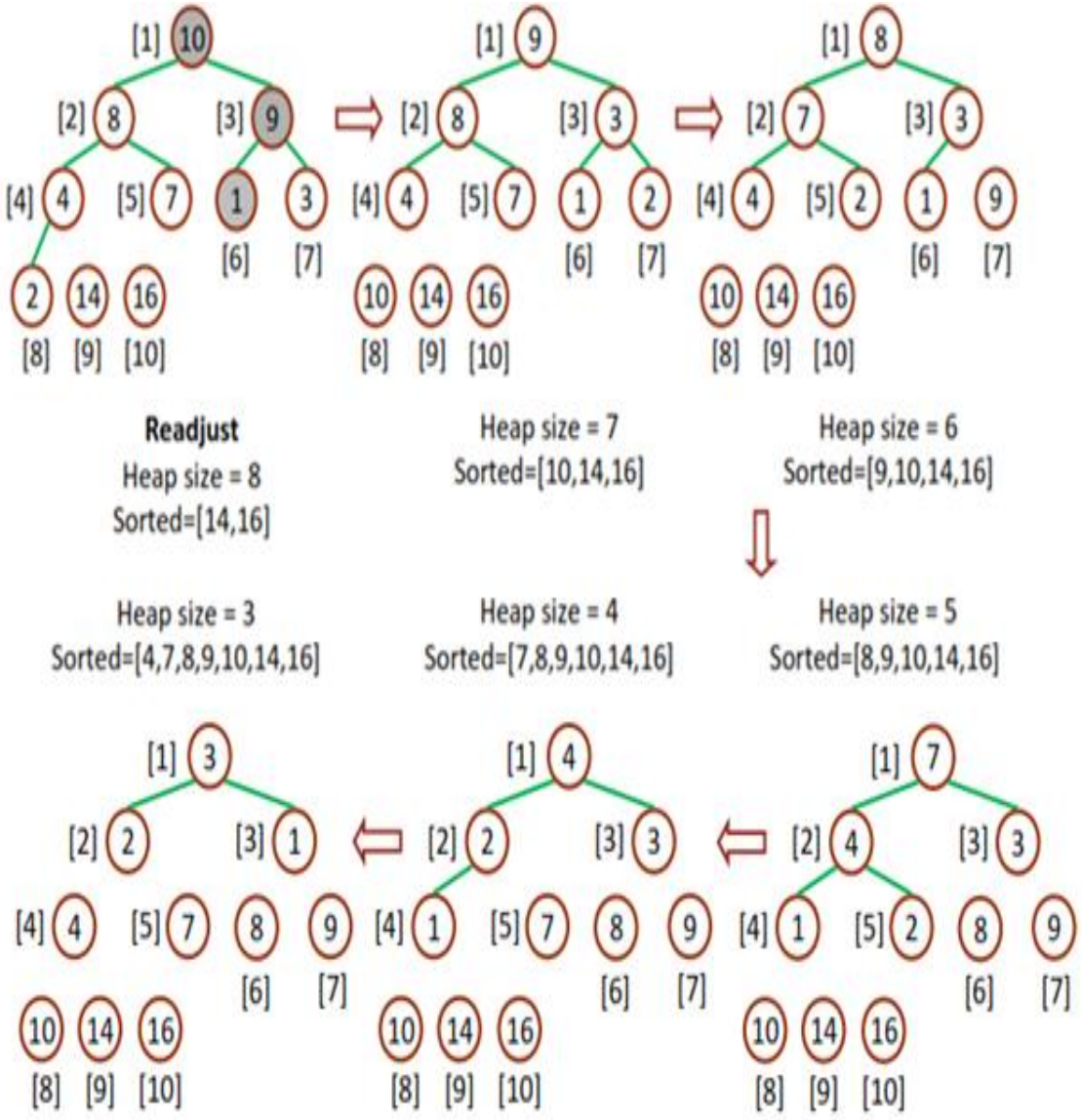


	1	2	3	4	5	6	7	8	9	10
A	1	2	3	4	7	8	9	10	14	16





Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



Program for Heap Sort

```
#include<stdio.h>
void heap_sort(int[], int);
void makeheap(int[], int);
int main(void)
{
        int a[10], n, i;
        printf(" Enter the size of the array ");
        scanf("%d", &n);
        printf(" Enter the array elements ");
```


Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING


```
        for(i=0;i<n;i++)
                scanf("%d", &a[i]);
        makeheap(a, n);
        heap_sort(a, n);
        printf(" The elements after sorting are ");
        for(i=0;i<n;i++)
                printf("\t%d", a[i]);
        return 0;
}
void makeheap(int a[], int n)
{
        int i, val, j, parent;
        for(i=1;i<n;i++)
        {
                val = a[i];
                j = i;
                parent = (j - 1) / 2;
                while(j>0 && parent < val)
                {
                        a[j] = a[parent];
                        j = parent;
                        parent = (j - 1) / 2;
                }
                a[j] = val;
        }
}

void heap_sort(int a[], int n)
{
        int i, j, k, temp;
        for(i=n-1;i>0;i--)
        {
```


Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING


```
            temp = a[i];
            a[i] = a[0];
            k = 0;
            if(i == 1)
                    j = -1;
            else
                    j = 1;
            if(i > 2 && a[2] > a[1])
                    j = 2;
            while(j >=0 && temp < a[j])
            {
                    a[k] = a[j];
                    k = j;
                    j = 2 * k +1;
                    if(j+1 <= i-1 && a[j] < a[j+1])
                            j++;
                    if(j > i-1)
                            j = -1;
            }
            a[k] = temp;
        }
}
```