Unit-2

SORTING / SEARCHING

INTRODUCTION TO SORTING

- Sorting is nothing but arranging the data in ascending or descending order. The term **sorting** came into picture, as humans realized the importance of searching quickly.

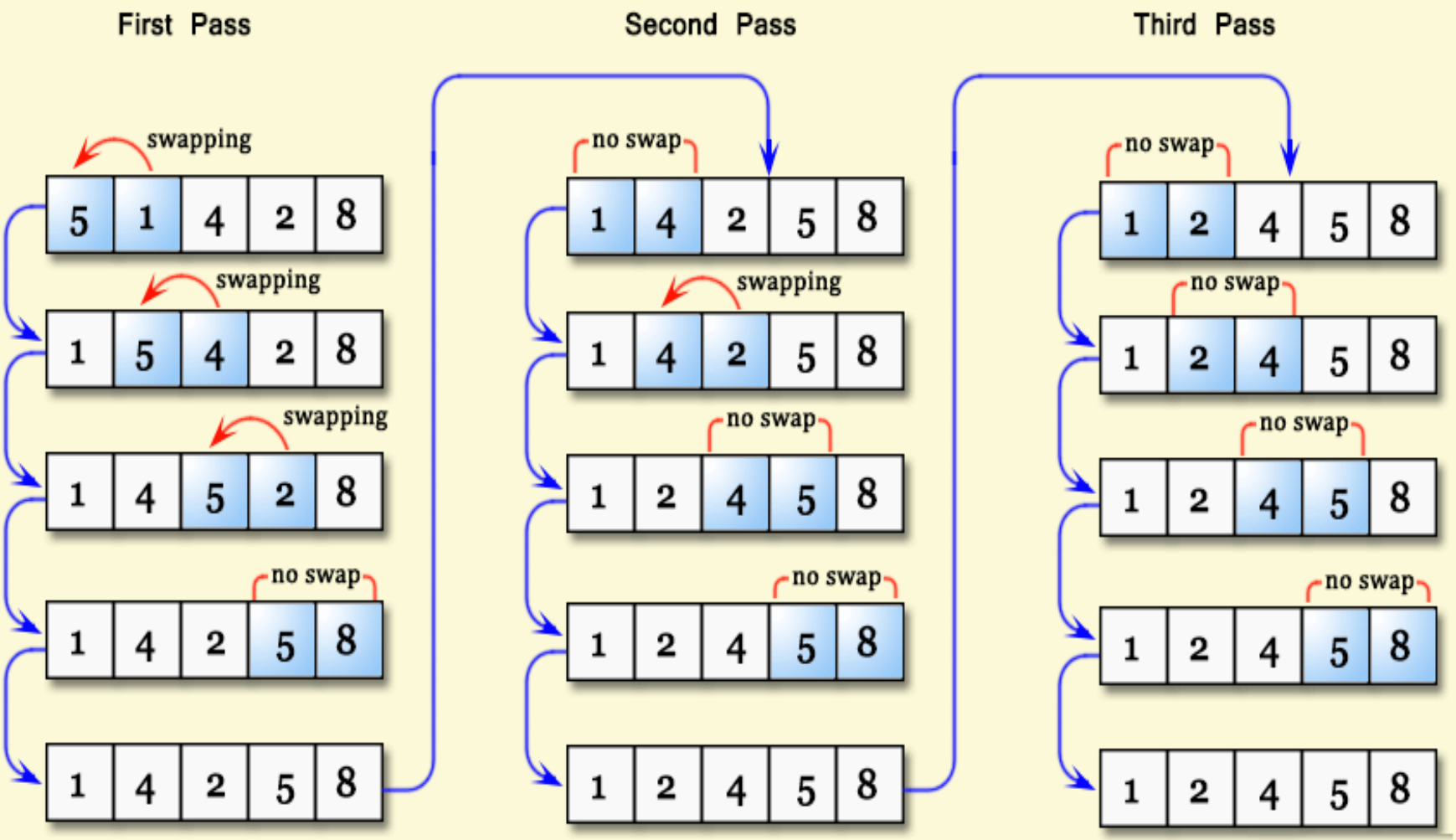
- Different types of Sorting Algorithms
- Bubble Sort
- Insertion Sort
- Selection Sort
- Quick Sort
- Merge Sort
- Bucket Sort
- Shell Sort

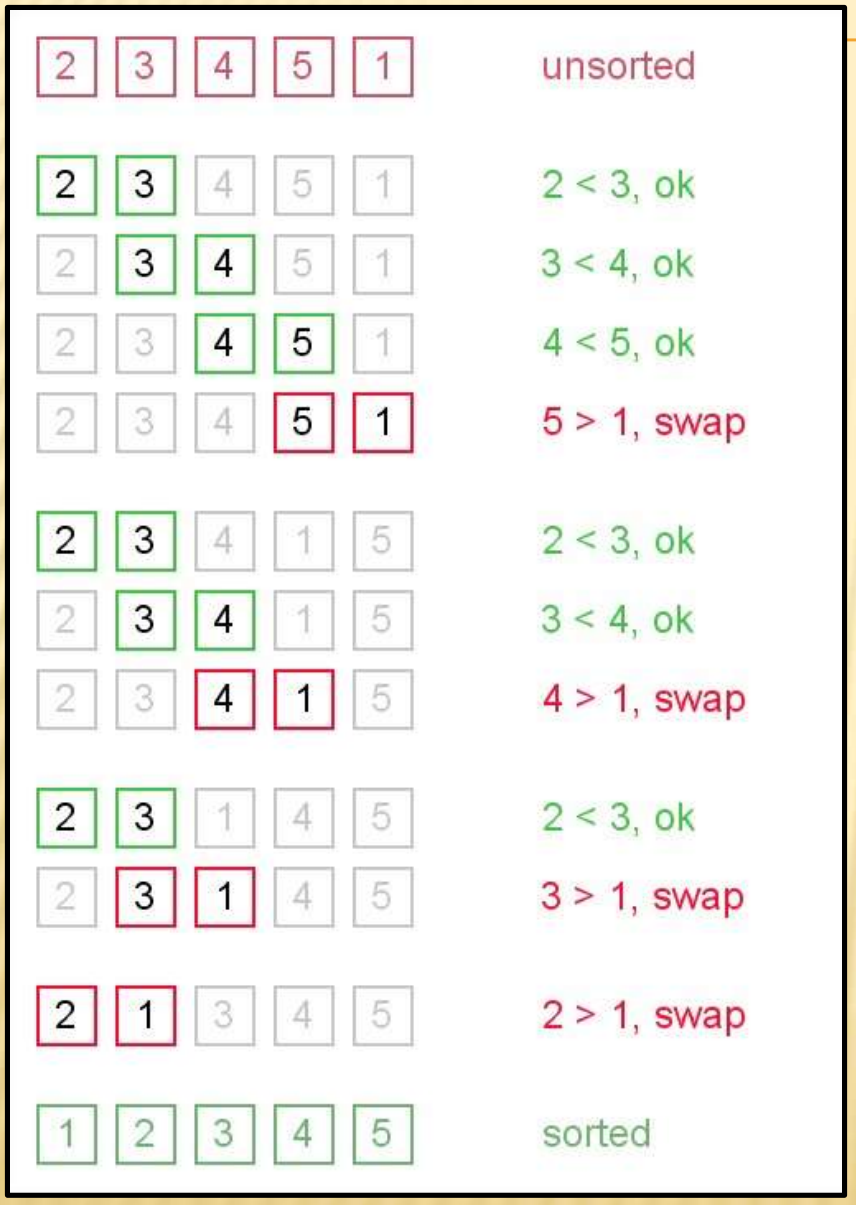
BUBBLE SORT

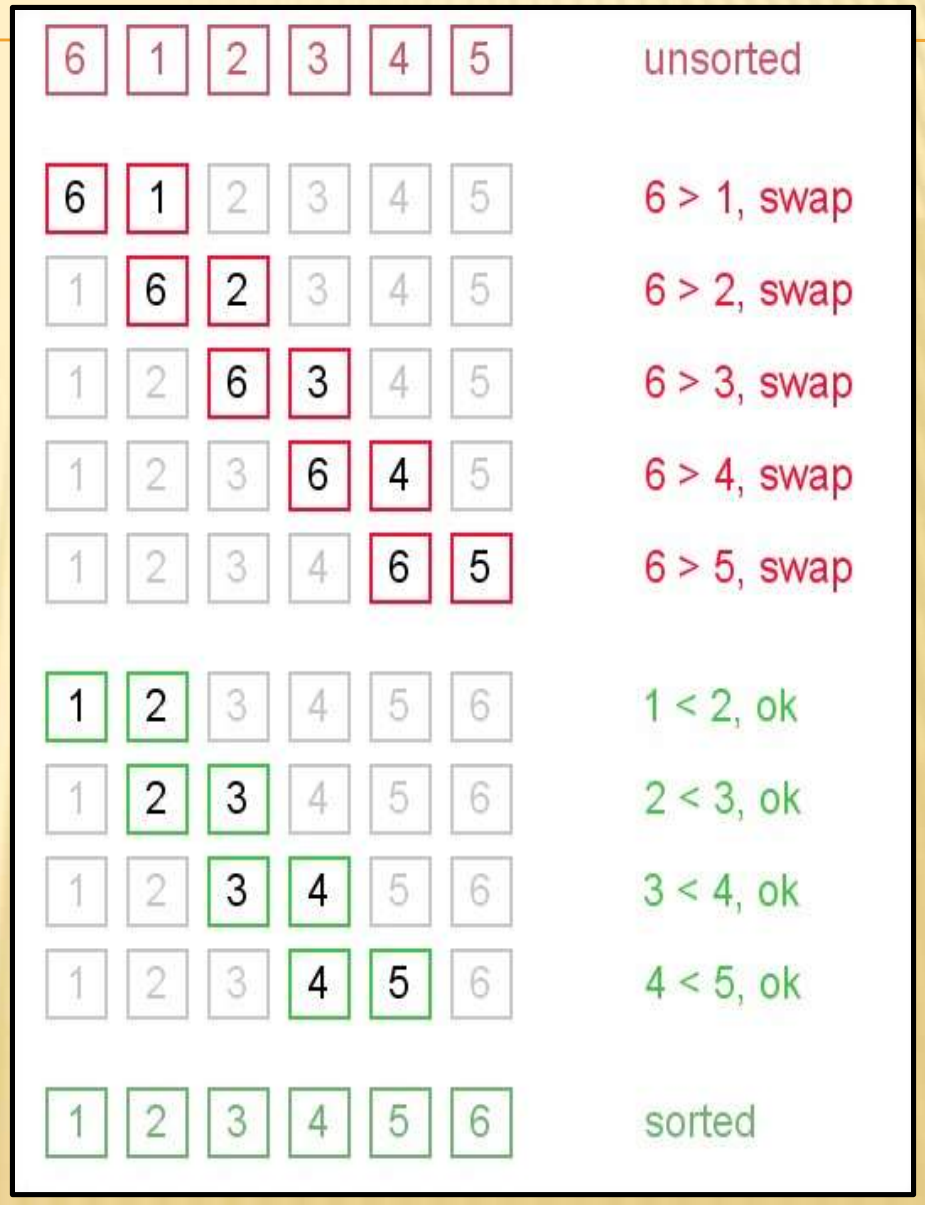
- **Bubble Sort** is a simple algorithm which is used to sort a given set of n elements provided in form of an array with n number of elements.
- Bubble Sort compares all the element one by one and sort them based on their values.
- This sorting algorithm is based on comparing and exchanging pairs of adjacent element in array.
- The bubble sort method derives it name from the fact that the smallest data item bubbles up to the top of the array.

Bubble Sort Example

Codingcompiler.com




2 3 4 5 1
unsorted
2 3 4 5 1
2 < 3, ok
2 3 4 5 1
3 < 4, ok
2 3 4 5 1
4 < 5, ok
2 3 4 5 1
5 > 1, swap
2 3 4 1 5
2 < 3, ok
2 3 4 1 5
3 < 4, ok
2 3 4 1 5
4 > 1, swap
2 3 1 4 5
2 < 3, ok
2 3 1 4 5
3 > 1, swap
2 1 3 4 5
2 > 1, swap
1 2 3 4 5
sorted


6 1 2 3 4 5
unsorted
6 1 2 3 4 5
6 > 1, swap
1 6 2 3 4 5
6 > 2, swap
1 2 6 3 4 5
6 > 3, swap
1 2 3 6 4 5
6 > 4, swap
1 2 3 4 6 5
6 > 5, swap
1 2 3 4 5 6
1 < 2, ok
1 2 3 4 5 6
2 < 3, ok
1 2 3 4 5 6
3 < 4, ok
1 2 3 4 5 6
4 < 5, ok
1 2 3 4 5 6
sorted

BUBBLE SORT ALGO.

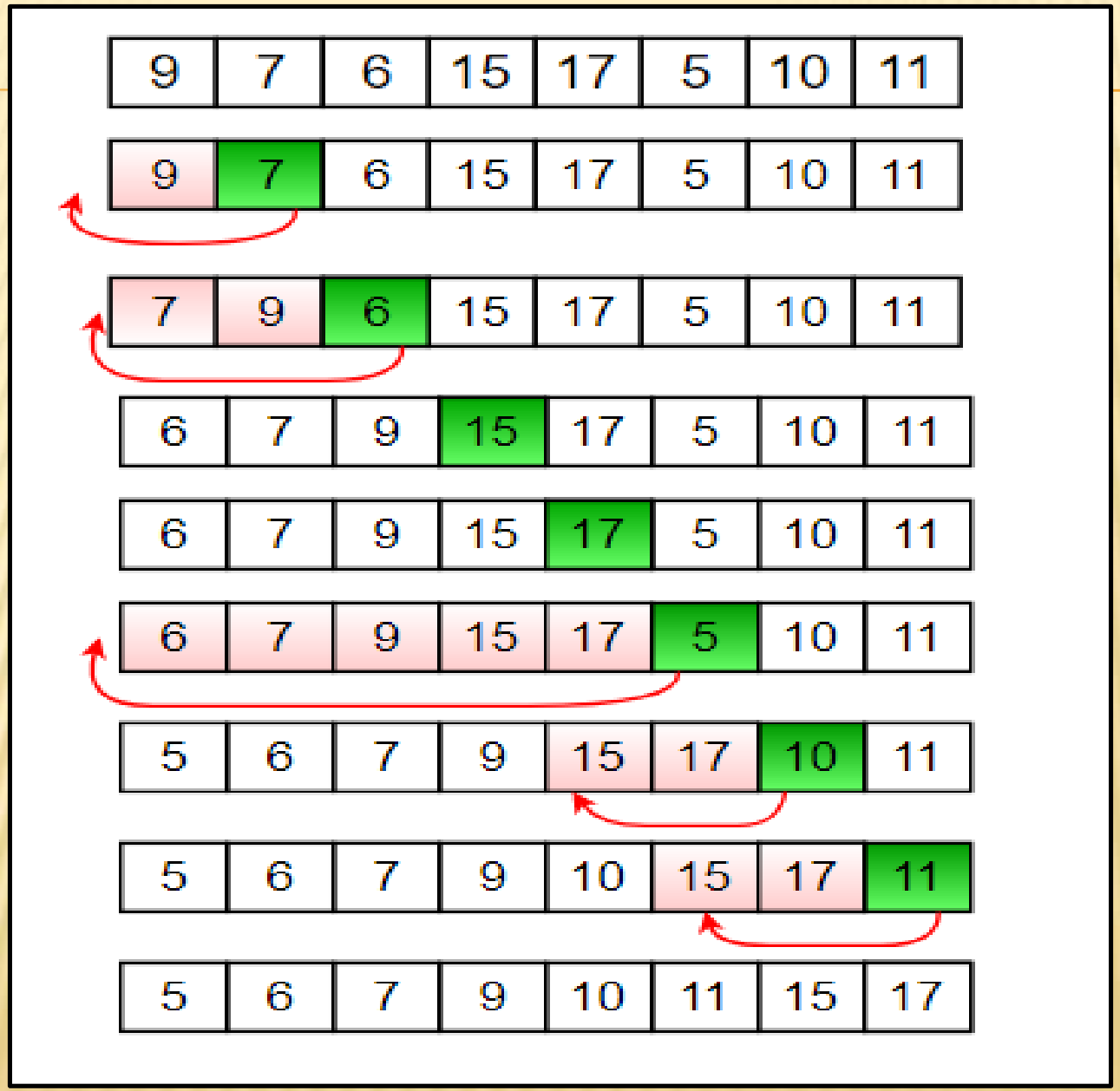
Step-1 [initialize]
i=0 , j=0
Step-2 Repeat through step-6 while (i=n-1)
Step-3 Repeat through step-5 while (j<i)
Step-4 if(a[j]>a[i+1])
t=a[j];
a[j]=a[j+1]
a[j+1]=t;
Step-5 j=j+1
Step-6 i=i-1
Step-7 exit

PROGRAM – BUBBLE SORT

```
#include <stdio.h>
#include <conio.h>
#define SIZE 4
void bubblesort(int a[],int n)
{
        int i,j,temp;
        for(i=n-1;i>0;i–)
        {
                for(j=0;j<i;j++)
                {
                        if(a[j]>a[j+1])
                        {
                                temp=a[j];
                                a[j]=a[j+1];
                                a[j+1]=temp;
                        }
                }
        }
}
void main ()
{
        int a[SIZE],i,n;
        clrscr();

        for (i=0;i<SIZE;i++)
        {
                printf("Enter value := ");
                scanf("%d",&a[i]);
        }
        bubblesort(a,SIZE);
        printf("\n after sorting");
        for(i=0;i<SIZE;i++)
        {
                printf("\n\t = %d ",a[i]);
        }
        getch();
}
```

INSERTION SORT

- They are sorted, or arranged in the ascending order of their numbers.
- Its space complexity is less. Like bubble Sort, insertion sort also requires a single additional memory space.
- Insertion sort is a simple algorithm that is relatively efficient for small and mostly sorted list, and often Is used as part of more sophisticated algorithms.
- It works by taking element from the list one by one and inserting them in their correct position into a new sorted list. Insertion sort works just like its name suggest – it inserts each element into its proper place in the final list.


9 7 6 15 17 5 10 11
9 7 6 15 17 5 10 11
7 9 6 15 17 5 10 11
6 7 9 15 17 5 10 11
6 7 9 15 17 5 10 11
6 7 9 15 17 5 10 11
5 6 7 9 15 17 10 11
5 6 7 9 10 15 17 11
5 6 7 9 10 11 15 17

INSERTION SORT ALGO.

Step-1 [initialize]

i=0 , t=0

Step-2 Repeat through step-8 for while (i<size)

Step-3 j=0

Step-4 Repeat through step-7 for while (j<i)

Step-5 if(a[j]>a[i] then

t=a[j];

a[j]=a[i]

k=i;

Repeat through step-8 while (k>j)

a[k]=a[k-1]

step-6 k--

Step-7 j=j+1

Step-8 i=i+1

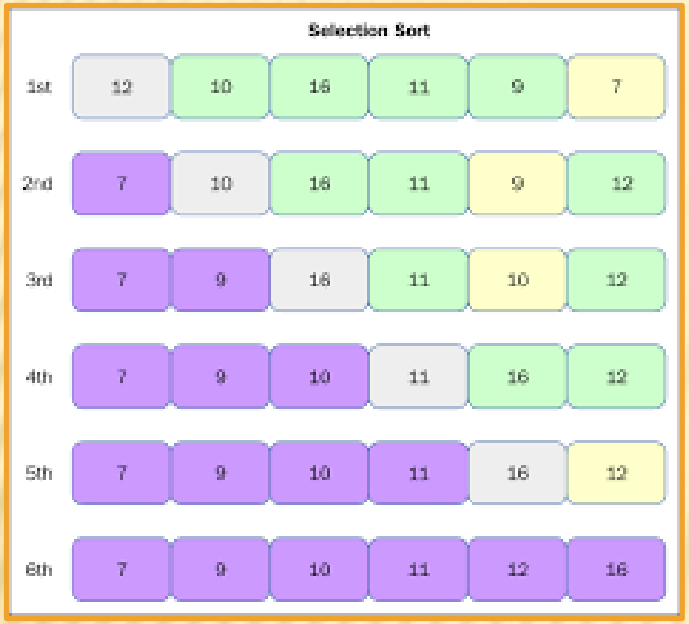
Step-9 exit

PROGRAM – INSERTION SORT

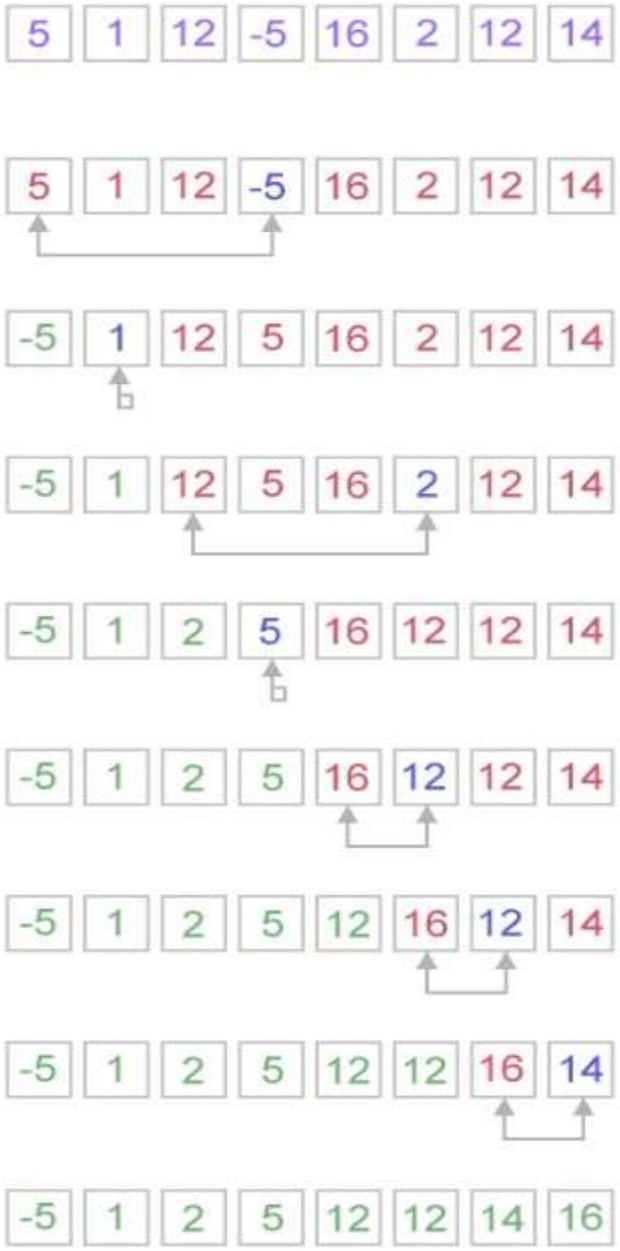
```
#include <stdio.h>
#include <conio.h>
#define size 5
void main ()
{
        int a[size],i,j,k,t;
        clrscr ();
        for(i=0;i<size;i++)
        {
                printf("Enter any values a[%d] :=",i);
                scanf("%d",&a[i]);
        }
        for(i=0;i<size;i++)
        {
                for(j=0;j<i;j++)
                {
                        if(a[j]>a[i])
                        {
                                t=a[j];
                                a[j]=a[i];
                                for(k=i;k>j;k--)
                                        a[k]=a[k-1];
                                a[k+1]=t;
        }       }       }
        printf("\n sorted values..\n");
        for(i=0;i<size;i++)
        {
                printf("\n\t%d",a[i]);
        }
        getch ();
}
```

SELECTION SORT

- Selection sort is conceptually the most simplest sorting algorithm.
  - This algorithm will first find the **smallest** element in the array and swap it with the element in the **first** position,
  - then it will find the **second smallest** element and swap it with the element in the **second** position,
  - and it will keep on doing this until the entire array is sorted.



Example. Sort {5, 1, 12, -5, 16, 2, 12, 14} using selection sort.



SELECTION SORT ALGO.

Step-1 [initialize]

i=0 ,j=0, t=0

Step-2 Repeat through step-6 for while (i<size)

Step-3 Repeat through step-5 for while (j<size)

Step-4 if(a[i]>a[j] then

t=a[i];

a[i]=a[j]

a[j]=t

step-5 j=j+1

Step-6 i=i+1

Step-7 exit

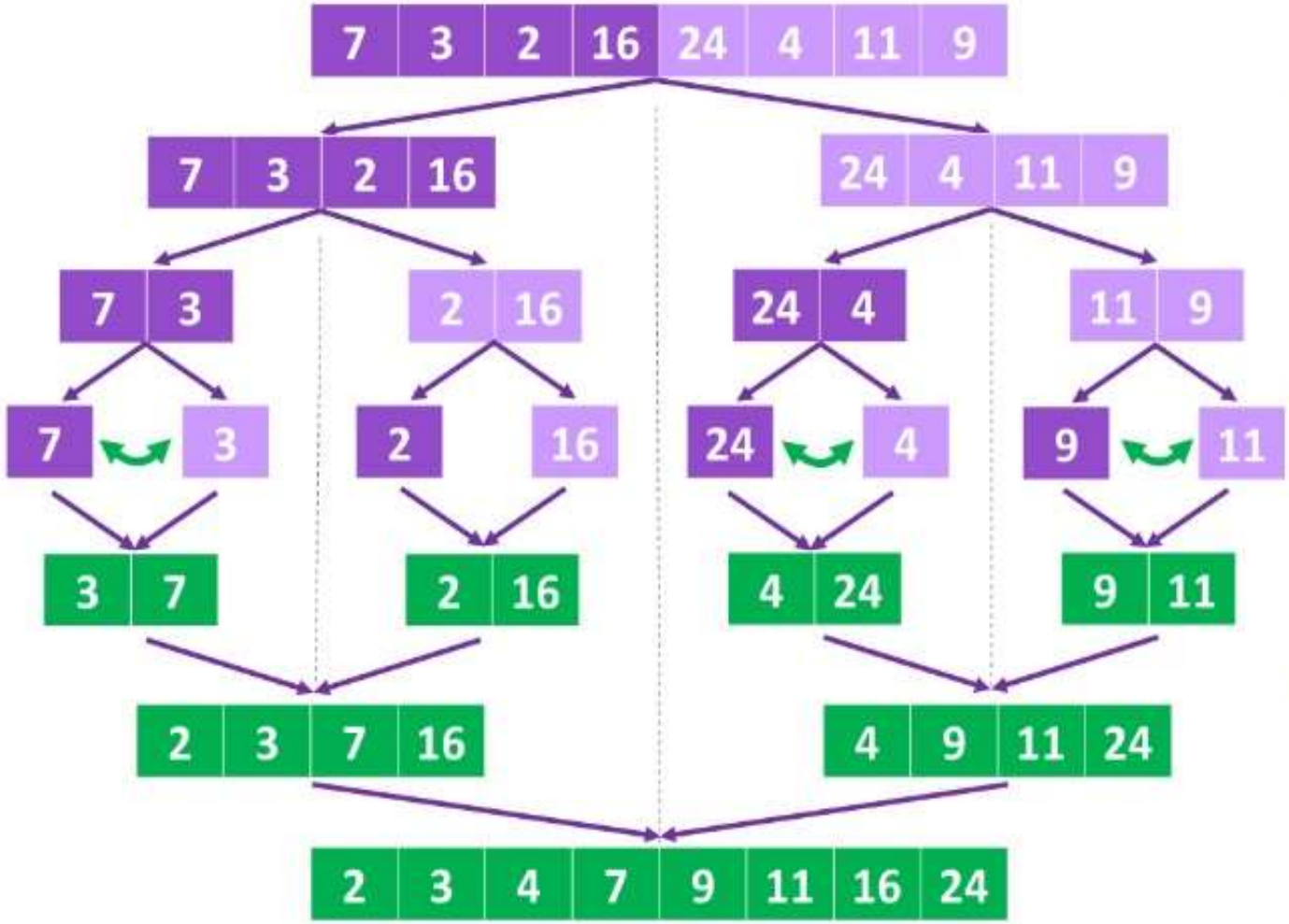
PROGRAM – SELECTION SORT

```
#include <stdio.h>
#include <conio.h>
#define SIZE 5
void select_sort(int[]);
void main()
{
        int a[SIZE],i;
        clrscr();
        printf("\n\n");
        for(i=0;i<SIZE;i++)
        {
                printf("Enter valuea[%d]",i);
                scanf("%d",&a[i]);
        }
        select_sort(a);
        getch();
}
void select_sort(int arr[])
{
        int i,j,t=0;
        for(i=0;i<SIZE;i++)
        {
                for(j=i+1;j<SIZE;j++)
                {
                        if(arr[i]>arr[j])
                        {
                                t=arr[i];
                                arr[i]=arr[j];
                                arr[j]=t;
        }               }               }
        printf("sorted element");
        for (i=0;i<SIZE;i++)
        {
                printf("\n %d",arr[i]);
        }
        getch();
}
```

MERGE SORT

- Merge Sort is a Divide and Conquer algorithm.
- It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves.
- **The merge() function** is used for merging two halves.

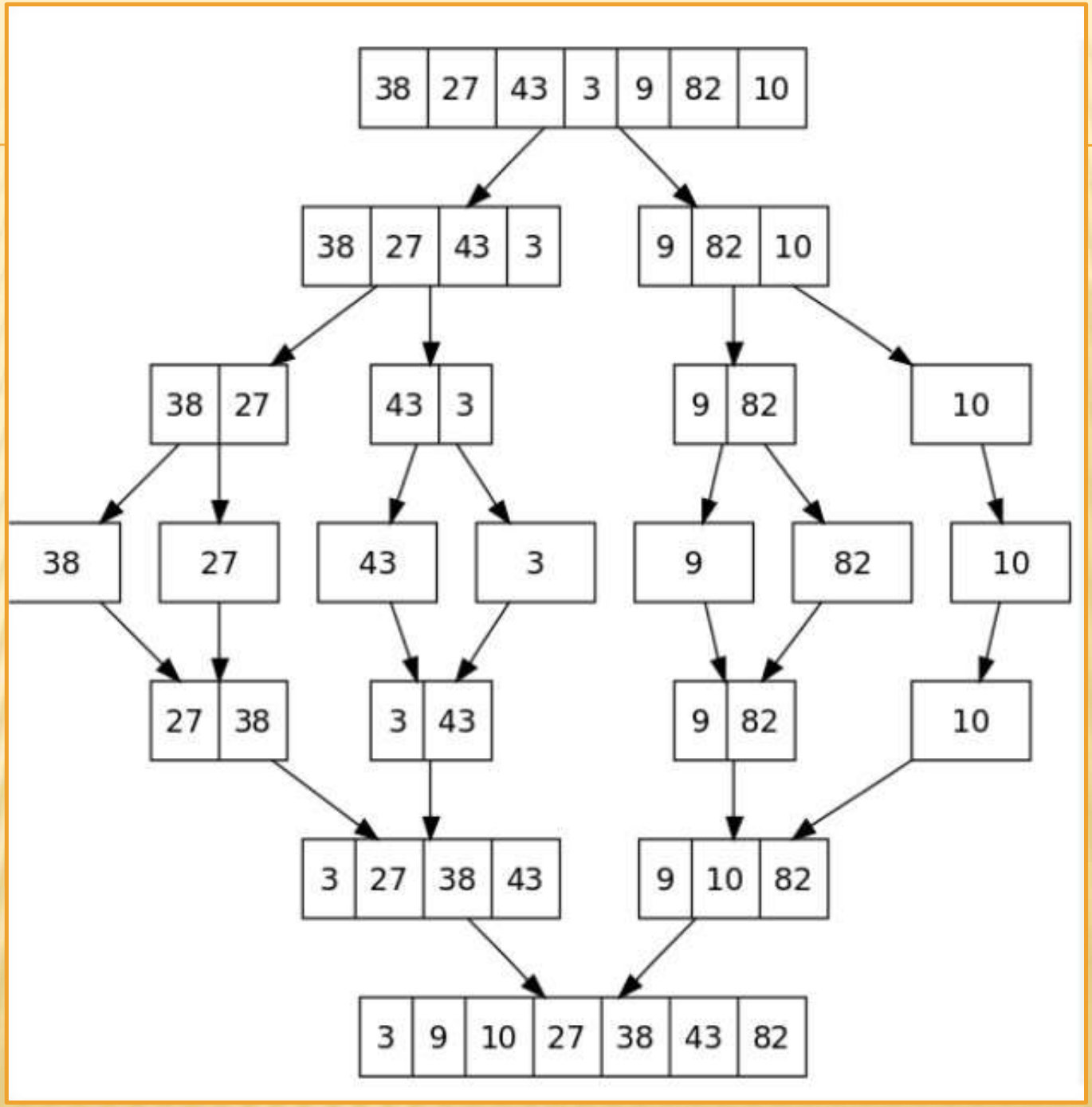
Merge Sort



Step 1:
Split sub-lists in two until you reach pair of values.

Step 3:
Sort/swap pair of values if needed.

Step 4:
Merge and sort sub-lists and repeat process till you merge to the full list.


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

MARGE SORT ALGO

Step-1 [initialize]

i=0,j=0,k=0

Step-2 Repeat through step-9 while (i<n)

Step-3 Repeat through step-8 while (j<i)

Step-4 if(a[j] > a[i])

t=a[j]

a[j]=a[i]

Step-5 Repeat through step-7 while (k>j)

Step-6 a[k]=a[k-1]

a[k+1=t;

Step-7 K- -

Step-8 j=j+1

Step-9 i=i+1

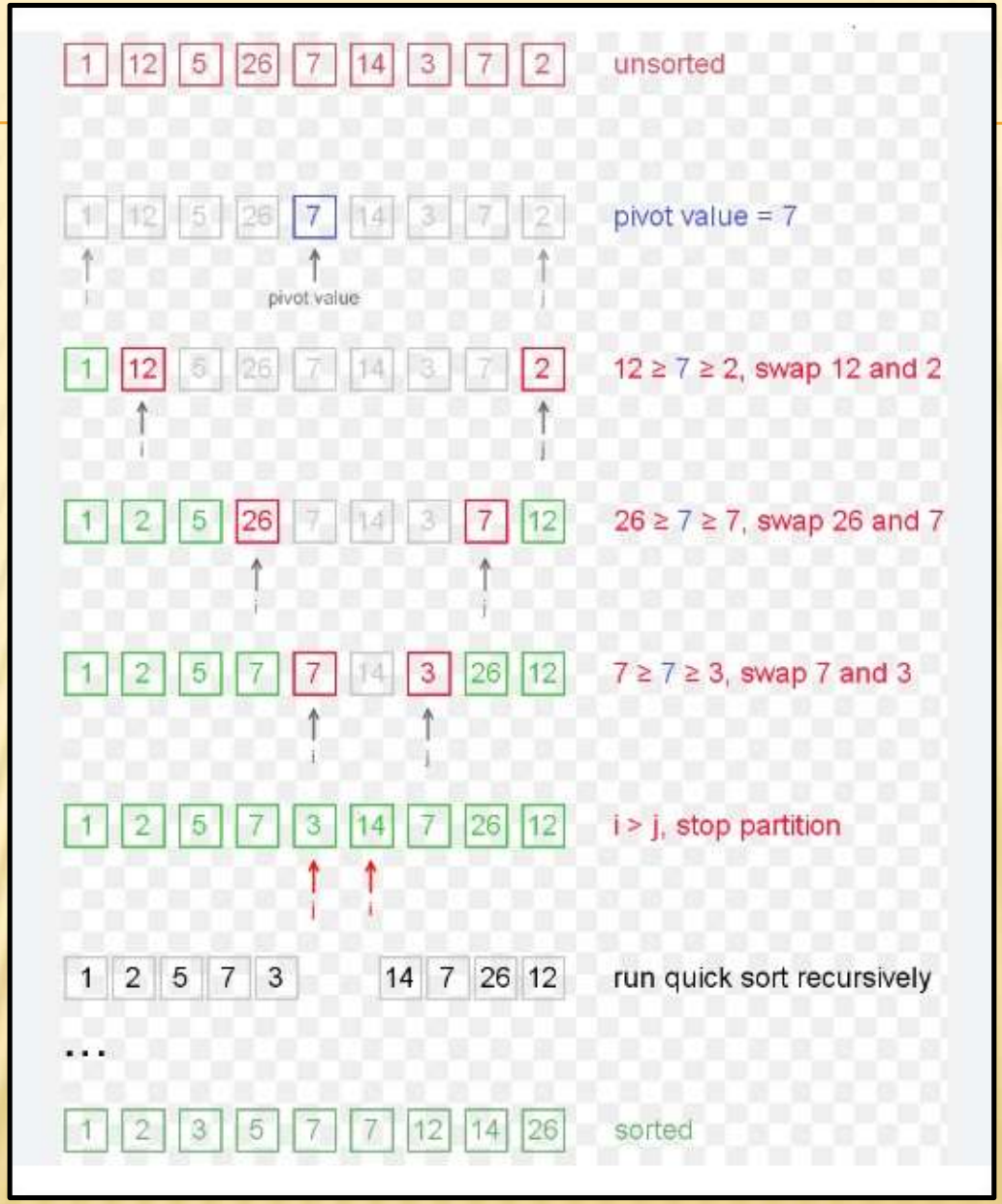
Step-10 exit

PROGRAM – MERGE SORT

```
// merge sort
#include <stdio.h>
#include <conio.h>
#define SIZE 5
void merge(int a[],int n)
{
        int i,j,k,t;
        for(i=0;i<n;i++)
        {
                for(j=0;j<i;j++)
                {
                        if(a[j]>a[i])
                        {
                                t=a[j];
                                a[j]=a[i];
                                for(k=i;k>j;k–)
                                        a[k]=a[k-1];
                                a[k+1]=t;
}       }       }       }
void main()
{       int a[SIZE],n,i;
        clrscr();
        for(i=0;i<SIZE;i++)
        {       printf("\t Enter valuea[%d]:",i);
                scanf("%d",&a[i]);
        }
        printf("\n\n\t BEFORE SORTING VALUE");
        printf("\n\t ====================");
        for(i=0;i<SIZE;i++)
                printf("\n\t\t %d",a[i]);
        merge(a,SIZE);
        printf("\n\t AFTER SORTING VALUE");
        printf("\n\t ====================");
        for(i=0;i<SIZE;i++)
                 printf("\n\t\t%d",a[i]);
        getch();
}
```

QUICK SORT

- **Quick sort** is a highly efficient **sorting** algorithm and is based on partition of array of **data** into smaller arrays.
- This algorithm is quite efficient for large-sized **data** sets as its average and worst case complexity are of $O(n^2)$, where n is the number of items.
- Partitioning **places all the elements less than the pivot in the left part of the array, and all elements greater than the pivot in the right part of the array.**


1 12 5 26 7 14 3 7 2
unsorted
pivot value = 7
pivot value
12 ≥ 7 ≥ 2, swap 12 and 2
26 ≥ 7 ≥ 7, swap 26 and 7
7 ≥ 7 ≥ 3, swap 7 and 3
i > j, stop partition
1 2 5 7 3 14 7 26 12
run quick sort recursively
...
1 2 3 5 7 7 12 14 26
sorted

QUICK_SORT ALGO.

- Quick_sort(a,first,last)

Step-1 [initialize]
low=first, high=last, pivot=a[(first+last)/2]

Step-2 Repeat through step-7 while (low<=high)

Step-3 Repeat through step-4 while (a[low]<pivot)

Step-4 low=low+1

Step-5 Repeat through step-6 while (a[high]>pivot)

Step-6 high=high-1

Step-7 If(low<=high)
t=a[low] a[low]=a[high]
a[high]=t low=low+1 high=high-1

Step-8 If(first < high)
quick_sort(a,first,high)

Step-9 if(low<last)
quick_sort(a,low,last)

Step-10 exit

PROGRAM QUICK SORT

```
#include <stdio.h>
#include <conio.h>
#define size 5
void quick_sort(int a[],int first,int last)
{
        int t,low,high,pivot;
        low=first;
        high=last;
        pivot=a[(first+last)/2];
        do
        {
                while(a[low]<pivot)
                        low++;
                while(a[high]>pivot)
                        high--;
                if(low<=high)
                {
                        t=a[low];
                        a[low++]=a[high];
                        a[high--]=t;
                }
        }while(low<=high);
        if(first<high)
                quick_sort(a,first,high);
        if(low<last)
                quick_sort(a,low,last);
}
```

```
void main ()
{
        int a[size],i;
        clrscr();
        for(i=0;i<size;i++)
        {
                printf("\n Enter values a[%d]:",i);
                scanf("%d",&a[i]);
        }
        printf("\n the values Before sorting");
        for(i=0;i<size;i++)
        {
                printf("%d",a[i]);
        }
        printf("\n");
        quick_sort(a,0,size-1);
        printf("\n the values After sorting");
        for(i=0;i<size;i++)
        {
                printf("%d",a[i]);
        }
        getch();
}
```

BUCKET SORT

- Bucket sort is a sorting algorithm that separate the elements into multiple groups said to be buckets.
- Elements in bucket sort are first uniformly divided into groups called buckets, and then they are sorted by any other sorting algorithm.
- The basic procedure of performing the bucket sort is given as follows -
  - First, partition the range into a fixed number of buckets.
  - Then, toss every element into its appropriate bucket.
  - After that, sort each bucket individually by applying a sorting algorithm.
  - And at last, connect all the sorted buckets.

Now, create buckets with a range from 0 to 25. The buckets range are 0-5, 5-10, 10-15, 15-20, 20-25. Elements are inserted in the buckets according to the bucket range. Suppose the value of an item is 16, so it will be inserted in the bucket with the range 15-20. Similarly, every item of the array will insert accordingly.

This phase is known to be the **scattering of array elements**.

Now, **sort** each bucket individually. The elements of each bucket can be sorted by using any of the stable sorting algorithms.

At last, **gather** the sorted elements from each bucket in order

Now, the array is completely sorted.

PROGRAM BUCKET SORT...1

```
#include <stdio.h>
#include <conio.h>
void main ()
{
        int a[100],i,j=0,k,n,max=1,ten=1,low=0,temp,skp;
        clrscr();
        printf("Enter the limit ==");
        scanf("%d",&n);

        for(i=1;i<=n;i++)
        {
                printf("\n Enter the number==>");
                scanf("%d",&a[i]);
        }
        for(i=1;i<=n;i++)
        {
                while((a[i]/ten)!=0)
                {
                        j++;
                        ten=ten*10;
                }
                if(j<max)
                        max=j;
                j=0;
                ten=1;
        }
```

CONT....

```
for(i=1;i<=max;i++)
        {
                low=0;
                k=1;
                while(low<10)
                {
                        for(j=k;j<=n;j++)
                        {
                                if((a[j]/ten)%10==low)
                                {
                                        temp=a[j];
                                        for(skp=j;skp>k;skp–)
                                        {
                                                a[skp]=a[skp-1];
                                        }
                                        a[skp]=temp;
                                        k++;
                                }
                        }
                        low++;
                }
                ten=ten*10;
        }
        printf("\n The sorted data ==>");
        for(i=1;i<=n;i++)
                printf("%d\t",a[i]);
        getch();
}
```

SHELL SORT

- Shell sort is the generalization of insertion sort, which overcomes the drawbacks of insertion sort by comparing elements separated by a gap of several positions.
- It is a sorting algorithm that is an extended version of insertion sort.
- Shell sort has improved the average time complexity of insertion sort.
- As similar to insertion sort, it is a comparison-based and in-place sorting algorithm.
- Shell sort is efficient for medium-sized data sets.

SHELL SORT ALGO.

Step-1 [initialize]

i=0 , t=0

Step-2 Repeat through step-8 for while (i<size)

Step-3 j=0

Step-4 Repeat through step-7 for while (j<i)

Step-5 if(a[j]>a[i] then

t=a[j];

a[j]=a[i]

k=i;

Repeat through step-8 while (k>j)

a[k]=a[k-1]

step-6 k- -

Step-7 j=j+1

Step-8 i=i+1

Step-9 exit

PROGRAM – SELL SORT

```
#include <stdio.h>
#include <conio.h>
#define size 5
void main ()
{
        int a[size],i,j,k,t;
        clrscr ();
        for(i=0;i<size;i++)
        {
                printf("Enter any values a[%d] :=",i);
                scanf("%d",&a[i]);
        }
        for(i=0;i<size;i++)
        {
                for(j=0;j<i;j++)
                {
                        if(a[j]>a[i])
                        {
                                t=a[j];
                                a[j]=a[i];
                                for(k=i;k>j;k--)
                                        a[k]=a[k-1];
                                a[k+1]=t;
        }       }       }
        printf("\n sorted values..\n");
        for(i=0;i<size;i++)
        {
                printf("\n\t%d",a[i]);
        }
        getch ();
}
```

SEARCHING

INTRODUCTION TO SEARCHING

- What is searching in array?

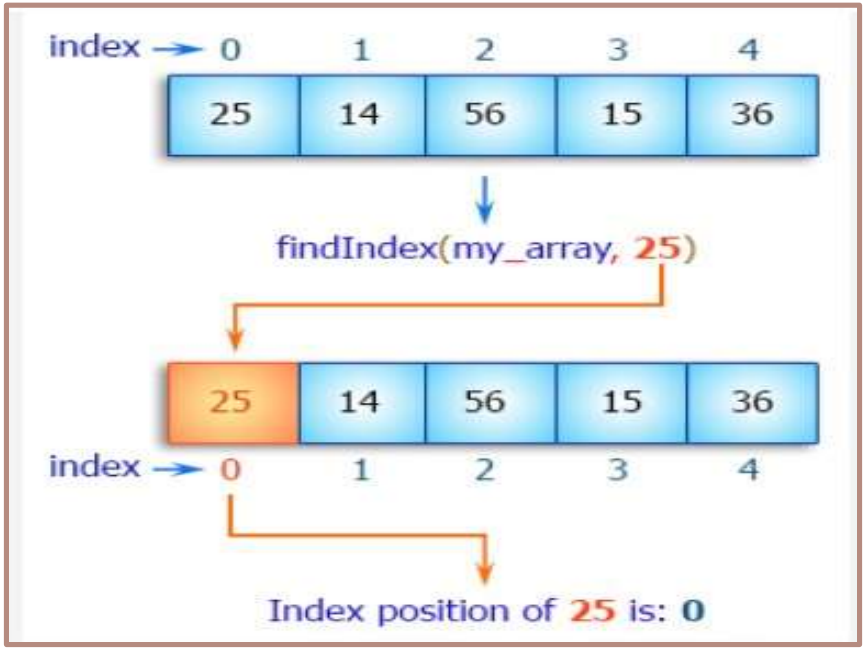
→Searching an array means **to find a particular element in the array.**

The search can be used to return the position of the element or check if it exists in the array.

- **Different types of Searching**
- Index search
- Linear/Sequential search
- Binary search

INDEX SEARCH

- Index search is **special search.**
- This search method is used to search a record in a file.
- Searching a record refers to the searching of location loc in memory where the file is stored.



INDEX SEARCH ALGORITHM

- Where a→ Represent the unsorted list index

- Step-1 [initialize]
- i=0 , found=0
- Step-2 repeat step-4 for (i<n)
- Step-3 if (i==found)
- --found=1
- --o/p→search is successful
- --o/p→value founded at position (a[i])
- Step-4 i=i+1
- Step-5 if found=0
- --o/p→ values not found
- Step-6 exit

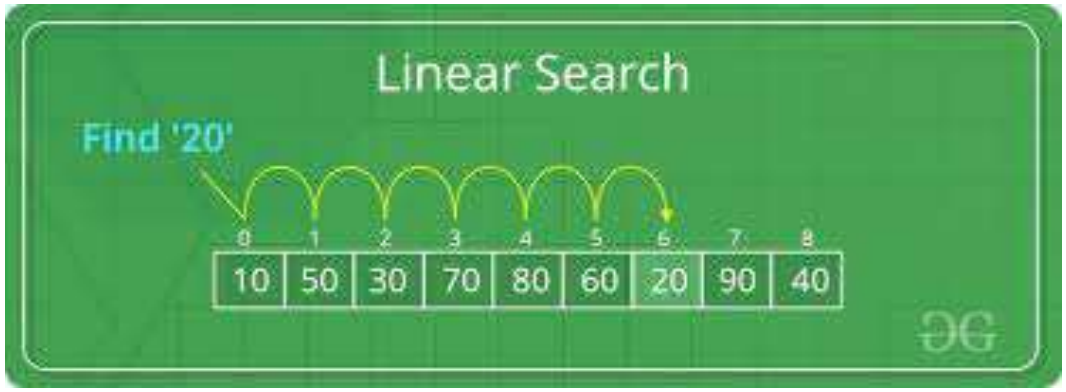
PROGRAM INDEX SEARCH

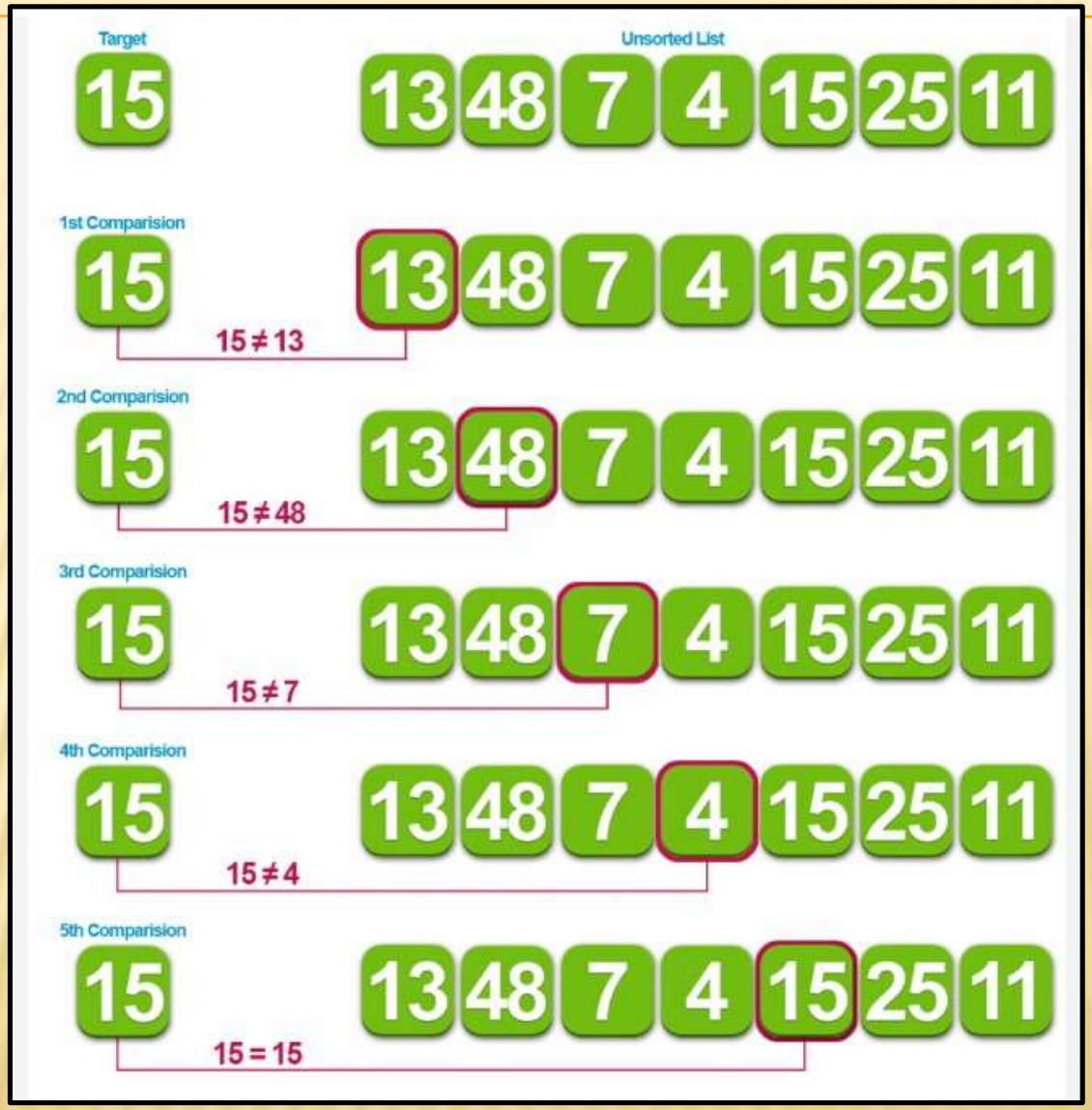
```
#include <stdio.h>
#include <conio.h>
void main ()
{
        int l,n,found,*a,f=0;
        clrscr();
        printf("\n Enter index number :");
        scanf("%d",&n);
        printf("\n Enter array valus:");
        for(i=0;i<n;i++)
        {
                printf("\n a[%d]=",i);
                scanf("%d",&a[i]);
        }
        printf("\n Enter number index to found:");
        scanf("%d",&found);
        for(i=0;i<n;i++)
        {
                if(i==found)
                {
                        printf("\n value founded at position[%d]\n\n value is a[%d] : %d",i,i,a[i]);
                        f=1;
                }
        }
        if(f==0)
        {
                printf("\n values not found \n please check the index");
        }
        getch();
}
```

LINEAR SEARCH

- A *linear search or sequential search* is a method for finding an element within a list. It sequentially checks each element of the list until a match is ***found*** or the whole list has been searched.

- Each an element from the unordered list




Target
Unsorted List
15
13 48 7 4 15 25 11
1st Comparision
15
13 48 7 4 15 25 11
15 ≠ 13
2nd Comparision
15
13 48 7 4 15 25 11
15 ≠ 48
3rd Comparision
15
13 48 7 4 15 25 11
15 ≠ 7
4th Comparision
15
13 48 7 4 15 25 11
15 ≠ 4
5th Comparision
15
13 48 7 4 15 25 11
15 = 15

LINEAR SEARCH ALGORITHM

- Linear(a,n)
- Where a→ Represent the unsorted list
- n→found the key from the list
- Step-1 [initialize]
- i=0 , found=0
- Step-2 repeat step-4 for i<size
- Step-3 if a[i]=key)
- --found=1
- --o/p→search is successful
- --o/p→key is searching at position(i+1)
- Step-4 i=i+1
- Step-5 if found=0
- --o/p→no searching
- Step-6 exit

PROGRAM LINEAR SEARCH

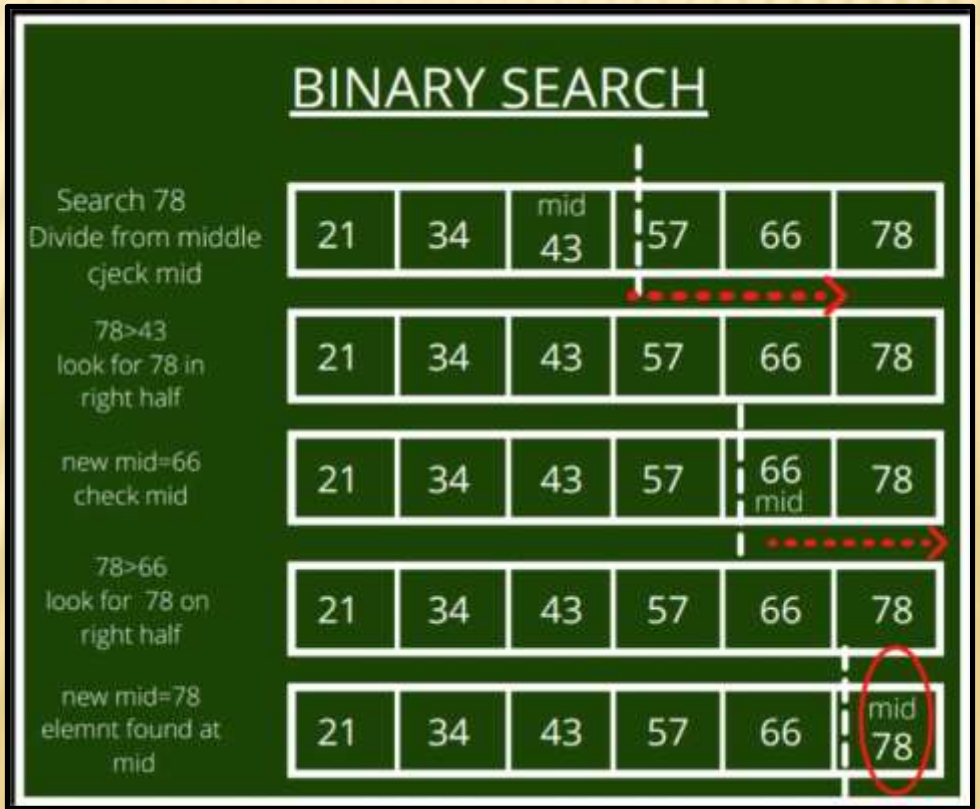
```
#include <stdio.h>
#include <conio.h>
#define size 5
void linear(int arr[],int);
void main()
{
        int i,a[size],n;
        clrscr();
        for(i=0;i<size;i++)
        {
                printf("\n\t a[%d] :=",i);
                scanf("%d",&a[i]);
        }
        printf("\n\t Enter the want to find = ");
        scanf("%d",&n);
        linear(a,n);
        getch();

}

void linear(int arr[],int key)
{
        int i=0,found=0;
        for(i=0;i<size;i++)
        {
                if(arr[i]==key)
                {
                        found=i;
                        printf("\n Search is sucess");
                        printf("\n key is position:%d",i+1);
                }
        }
        if(found==0)
        {
                printf("\n search element exit the list");
        }
}
```

BINARY SEARCH

- What is binary search?
- Binary search is **an efficient algorithm for finding an item from a sorted list of items.** It works by repeatedly dividing in half the portion of the list that could contain the item, until you've narrowed down the possible locations to just one.

- Each an element from the unordered list


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


BINARY SEARCH
Search 78
Divide from middle
cjeck mid
78>43
look for 78 in
right half
new mid=66
check mid
78>66
look for 78 on
right half
new mid=78
elemnt found at
mid
21 34 43 57 66 78
mid
66
mid
mid
78

BINARY SEARCH ALGORITHM

- Binary_search(a,n)
- Where a→ represents unsorted list.
- n→search the key
- Step-1 [intialize]
- low=0 , high=n-1 , flag=0
- Step-2 Repeat through step-4 while (low<=high)
- Step-3 mid=(low+high)/2
- Step-4 if(n<arr[mid]) then
- high=mid-1
- else if (n>arr[mid]) then
- low=mid+1
- else if(n==arr[mid])
- o/p–search successful location element “mid+1“
- flag=1
- Step-5 if flag==0 then
- o/p—search is un-successful
- Step-6 exit

PROGRAM BINARY SEARCH

```
#include <stdio.h>
#include <conio.h>
#define size 5
void binary(int[],int key);
void main()
{
        int i,a[size],t,j,n;
        int flag=0;
        clrscr();
        for(i=0;i<size;i++)
        {
                printf("\n Enter values");
                scanf("%d",&a[i]);
        }
        for(i=0;i<size;i++)
        {
                for(j=i+1;j<size;j++)
                {
                        if(a[i]>a[j])
                        {
                                t=a[i];
                                a[i]=a[j];
                                a[j]=t;
        }               }       }
        printf("\n\t before Sorting");
        for(i=0;i<size;i++)
        {
                printf("\n\t%d",a[i]);
        }
        printf("\n Enter Values for search:");
        scanf("%d",&n);
        binary(a,n);
        getch();
}
```

```
void binary(int arr[],int n)
{
        int high=0,mid=0,low=0,flag=0;
        high=size-1;
        mid=(low+high)/2;

        while(high>=low && flag!=1)
        {
                if(n<arr[mid])
                        high=mid-1;
                else if(n>arr[mid])
                        low=mid+1;
                else if(n==arr[mid])
                {
                        printf("\n search is
sucessful");
                        printf("\n found at
%d location ",mid+1);
                        flag=1;
                }
                mid=(high+low)/2;
        }
        if(flag==0)
        {
                printf("\n search is un-sucessful");
        }
}
```

The end...