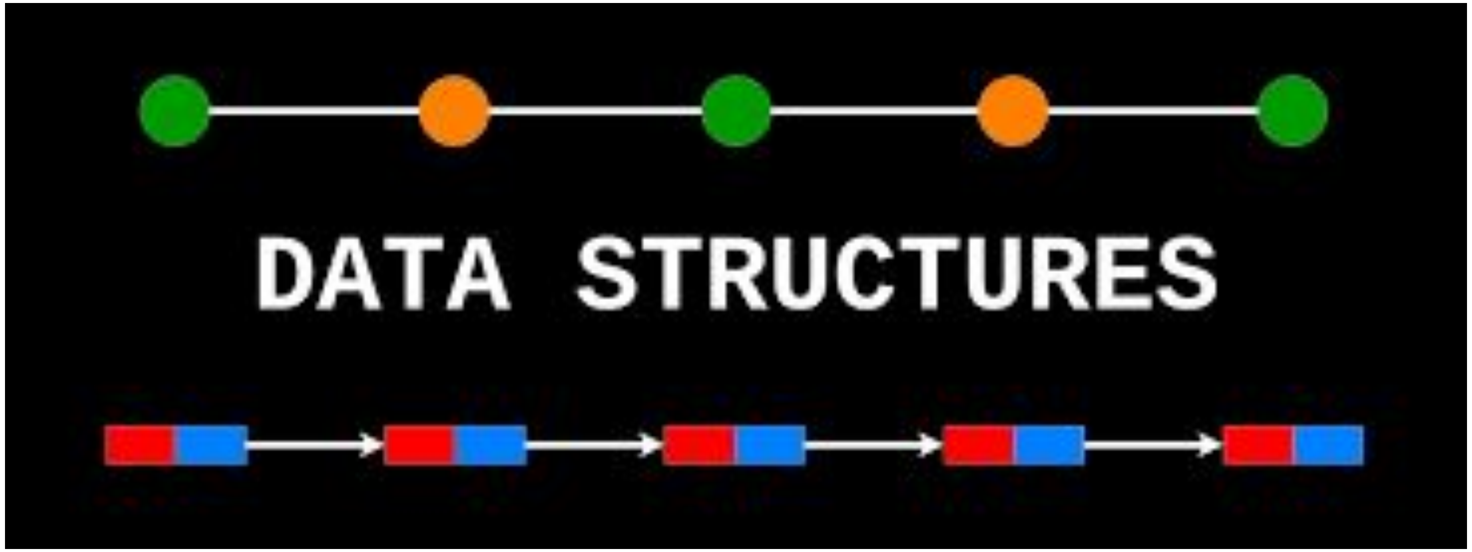
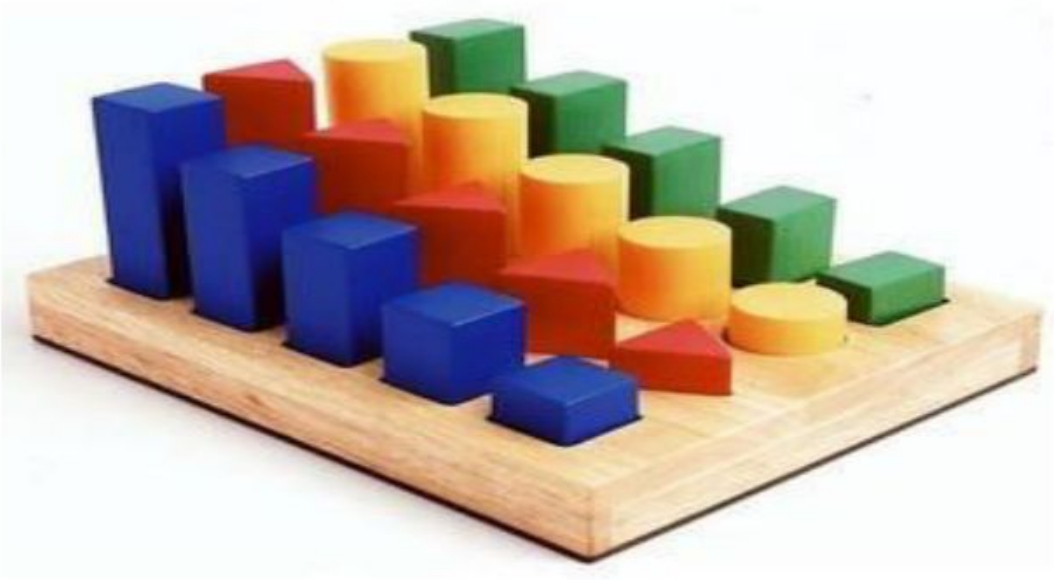
# DATA STRUCTURES USING C



**22BCA2C05**

**UNIT 2**

# Sorting



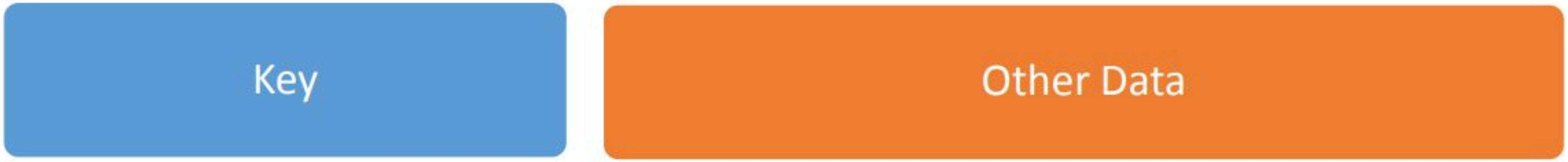
# Need for Sorting

# Sorting

Sorting means arranging data in a particular format or a particular order.

- The numbers to be sorted are part of a collection called record.
- Each record contains a key, the value to be sorted

Record



Key

Other Data

# Need for Sorting

- Searching becomes easier when data is sorted.

Sorting algorithms are used for sorting. They can handle situations like :

- Records have randomly ordered keys.
- All keys are distinct
- Need guaranteed performance.

# Basics of Sorting

## Sort Stability

- The key on which the data is being sorted is not unique for each record.

## Sort efficiency

- It relays on
  - Coding time
  - Space requirement
  - Run time or Execution time

## Quiz / Assessment

1) The technique used for arranging data elements in a specific order is called as __________.

a) Arranging

b) Filtering

c) Sorting

d) Distributing

2) The Time required to Complete the execution of a sorting program is called as __________.

a) Coding Time

b) Average Time

c) Running Time

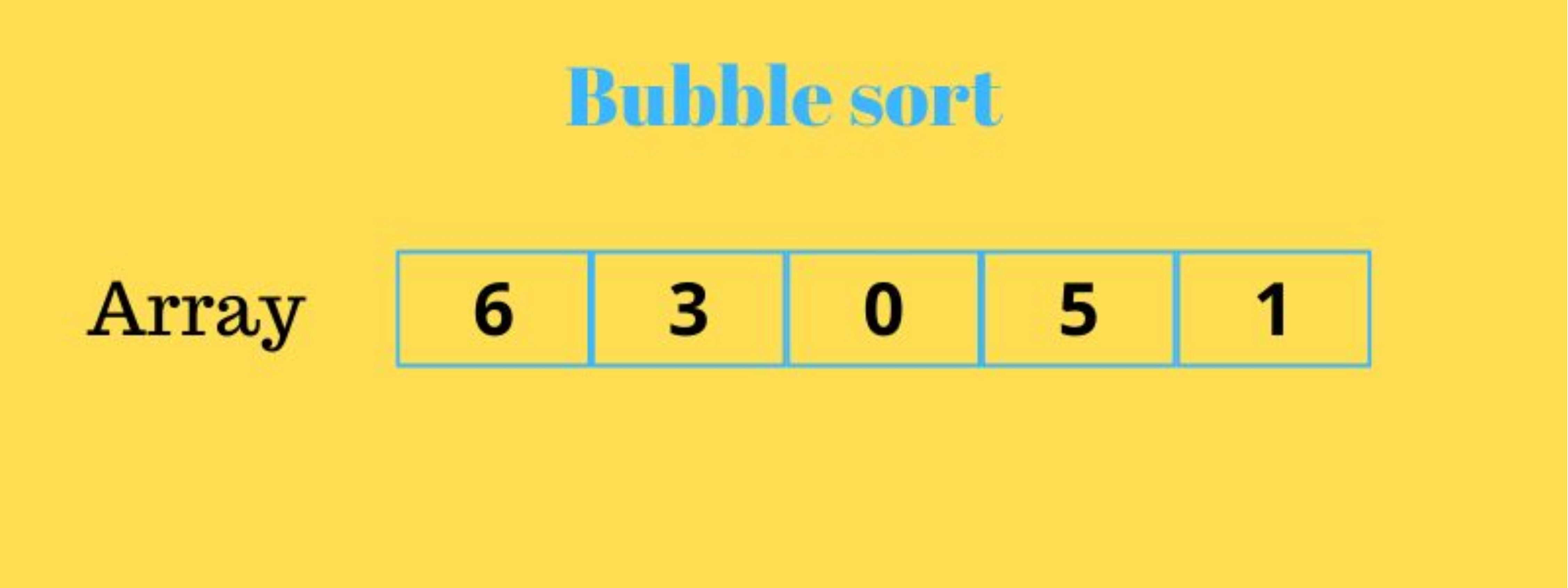
d) Total Time

# Sorting

Bubble sort, Selection sort, Insertion sort, Merge sort, Quick sort

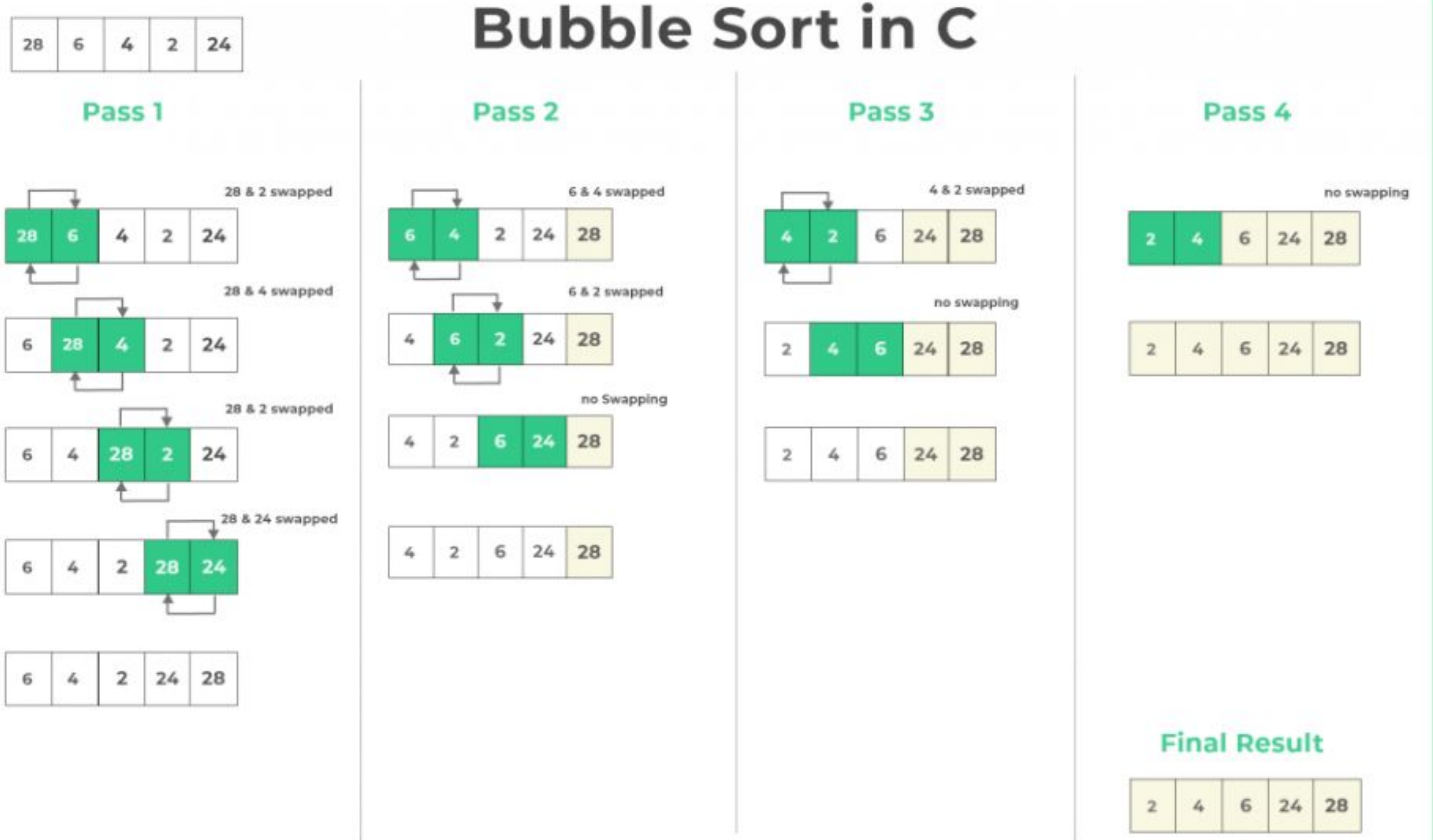
# Bubble Sort

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



| | | | | | |
|---|---|---|---|---|---|
| Array | 6 | 3 | 0 | 5 | 1 |

# Bubble Sort in C



28 6 4 2 24

## Pass 1

28 & 2 swapped

28 6 4 2 24

28 & 4 swapped

6 28 4 2 24

28 & 2 swapped

6 4 28 2 24

28 & 24 swapped

6 4 2 28 24

6 4 2 24 28

## Pass 2

6 & 4 swapped

6 4 2 24 28

6 & 2 swapped

4 6 2 24 28

no Swapping

4 2 6 24 28

4 2 6 24 28

## Pass 3

4 & 2 swapped

4 2 6 24 28

no swapping

2 4 6 24 28

2 4 6 24 28

## Pass 4

no swapping

2 4 6 24 28

2 4 6 24 28

## Final Result

2 4 6 24 28

# Algorithm

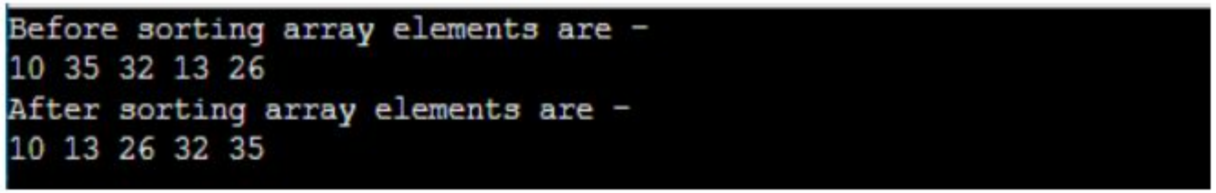
```
begin BubbleSort(arr)
  for all array elements
    if arr[i] > arr[i+1]
      swap(arr[i], arr[i+1])
    end if
  end for
  return arr
end BubbleSort
```

# Implementation of Bubble sort

```
#include<stdio.h>
 void print(int a[], int n) //function to print array elements
    {
   int i;
   for(i = 0; i < n; i++)
    {
      printf("%d ",a[i]);
    }
    }
 void bubble(int a[], int n) // function to implement bubble sort
 {
  int i, j, temp;
  for(i = 0; i < n; i++)
   {
    for(j = i+1; j < n; j++)
      {
         if(a[j] < a[i])
         {
           temp = a[i];
           a[i] = a[j];
           a[j] = temp;
         }     }   } }
```

```
void main ()
{
   int i, j,temp;
   int a[5] = { 10, 35, 32, 13, 26};
   int n = sizeof(a)/sizeof(a[0]);
   printf("Before sorting array elements are - \n");
   print(a, n);
   bubble(a, n);
   printf("\nAfter sorting array elements are - \n");
   print(a, n);
}
```

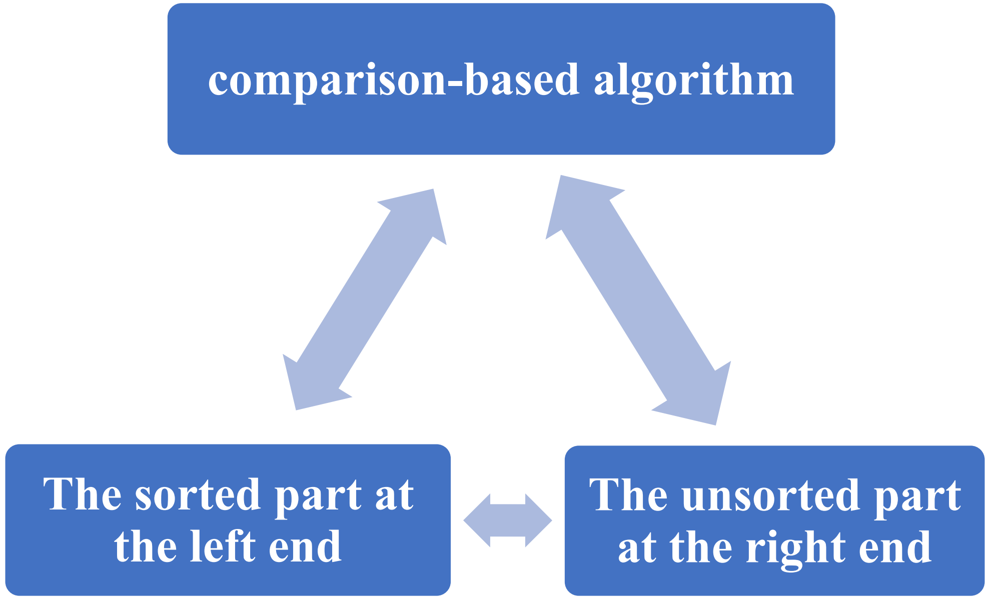
**Output**



```
Before sorting array elements are -
10 35 32 13 26
After sorting array elements are -
10 13 26 32 35
```

# Selection sort

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

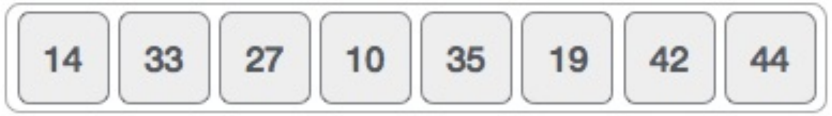

comparison-based algorithm
The sorted part at the left end
The unsorted part at the right end

**Selection sort is generally used when -**

- A small array is to be sorted
- Swapping cost doesn't matter
- It is compulsory to check all elements

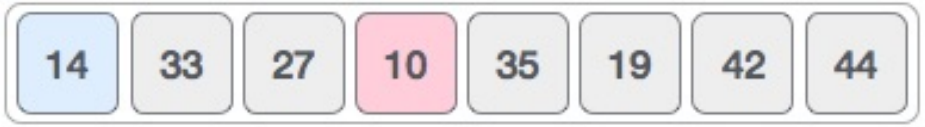
# Working Algorithm

**Let the elements of the array are**



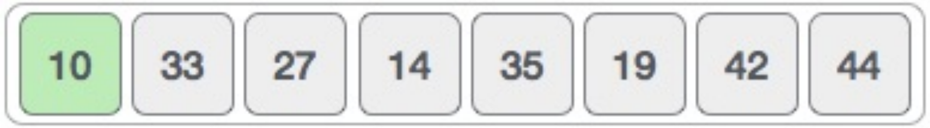
|  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| 14 | 33 | 27 | 10 | 35 | 19 | 42 | 44 |

For the first position in the sorted list, the whole list is scanned, and find that 10 is the lowest value.



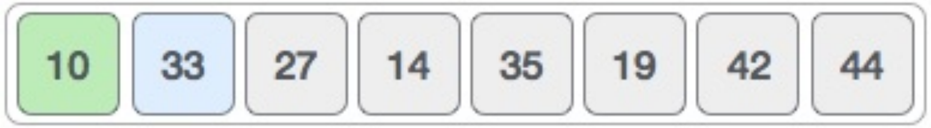
|  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| 14 | 33 | 27 | 10 | 35 | 19 | 42 | 44 |

So **we replace 14 with 10**. After one iteration 10, which happens to be the minimum value in the list, appears in the first position of the sorted list.



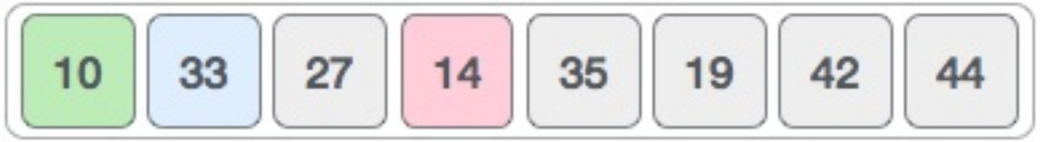
|  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| 10 | 33 | 27 | 14 | 35 | 19 | 42 | 44 |

For the second position, where 33 is residing, we start scanning the rest

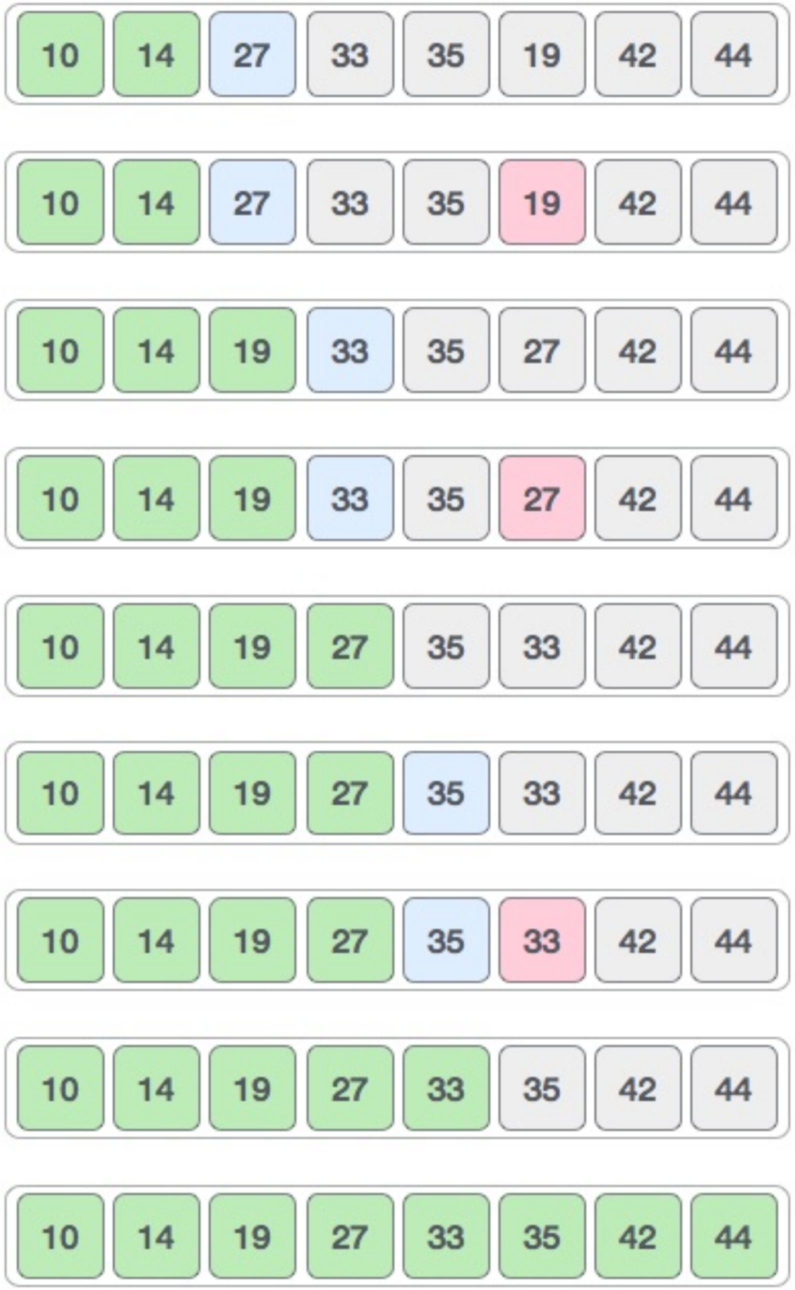


|  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| 10 | 33 | 27 | 14 | 35 | 19 | 42 | 44 |

We find that 14 is the second lowest value in the list and it should appear at the second place. We swap these values.

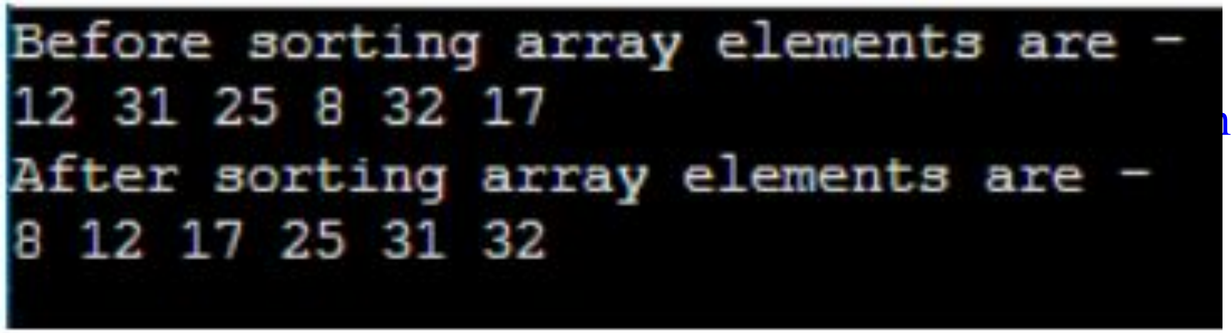


The same process is applied to the rest of the items in the array.



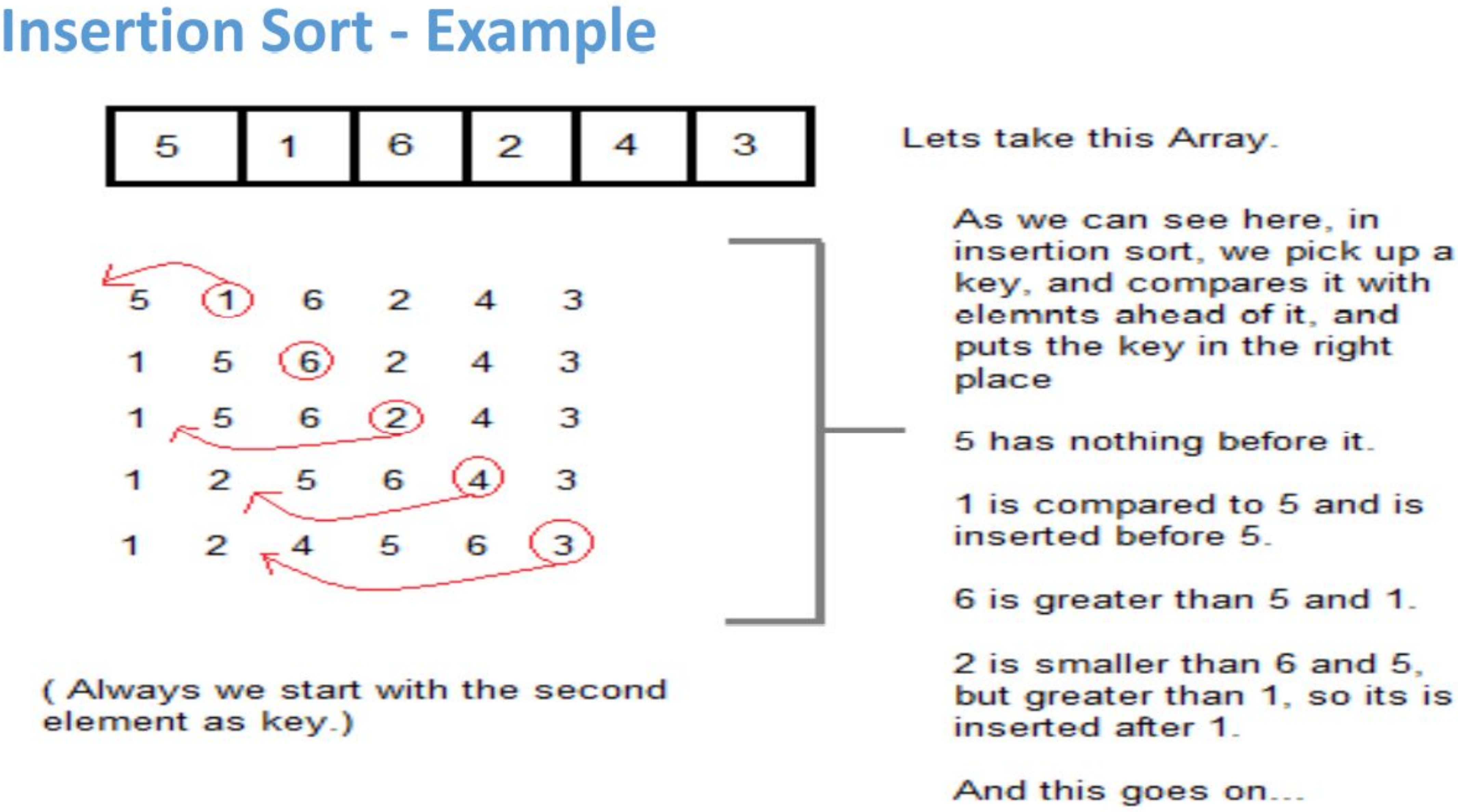
```
#include <stdio.h>
 void selection(int arr[], int n)
{
   int i, j, small;

for(i=0;i<n-1; i++)    // One by one move boundary of unsorted subarray
   {
      small = i; //minimum element in unsorted array
     for (j = i+1; j < n; j++)
      if (arr[j] < arr[small])
         small = j;
// Swap the minimum element with the first element
   int temp = arr[small];
   arr[small] = arr[i];
   arr[i] = temp;
   } }
```

```
void printArr(int a[], int n) /* function to print the array */
{
   int i;
   for (i = 0; i < n; i++)
      printf("%d ", a[i]);
}
 int main()
{
   int a[] = { 12, 31, 25, 8, 32, 17 };
   int n = sizeof(a) / sizeof(a[0]);
   printf("Before sorting array elements are - \n");
   printArr(a, n);

                          nts are - \n");
```



```
Before sorting array elements are -
12 31 25 8 32 17
After sorting array elements are -
8 12 17 25 31 32
```

# Insertion sort



## Insertion Sort - Example

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 5 | 1 | 6 | 2 | 4 | 3 |

Lets take this Array.

5 1 6 2 4 3

1 5 6 2 4 3

1 5 6 2 4 3

1 2 5 6 4 3

1 2 4 5 6 3

As we can see here, in insertion sort, we pick up a key, and compares it with elemnts ahead of it, and puts the key in the right place

5 has nothing before it.

1 is compared to 5 and is inserted before 5.

6 is greater than 5 and 1.

2 is smaller than 6 and 5, but greater than 1, so its is inserted after 1.

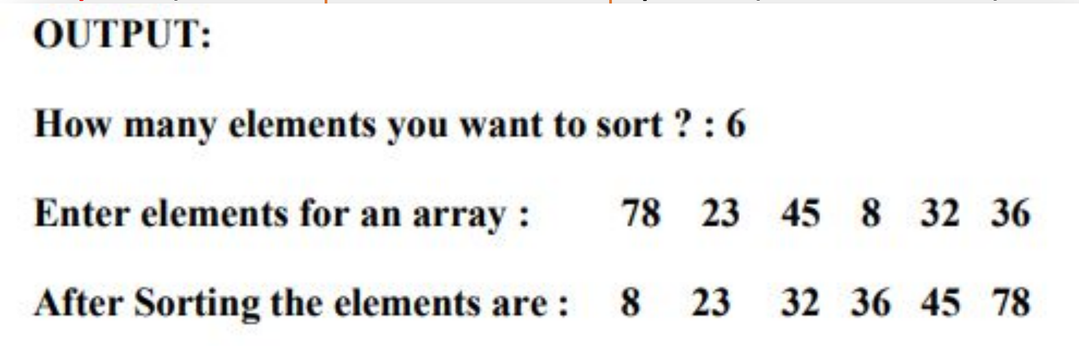
And this goes on...

( Always we start with the second element as key.)

# Insertion sort

```
#include<stdio.h>

#include<conio.h>

void main( )

{

 int a[10],i,j,k,n;

 clrscr( );

 printf("How many elements you want to sort?\n");

 scanf("%d",&n);

 printf("\nEnter the Elements into an array:\n");

 for (i=0;i<n;i++)

 scanf("%d",&a[i]);
```

```
for(i=1;i<n;i++)

 {

 k=a[i];

 for(j= i-1; j>=0 && k<a[j]; j--)

 a[j+1]=a[j];

a[j+1]=k;

 } printf("\n\n Elements after sorting: \n");

 for(i=0;i<n;i++)

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



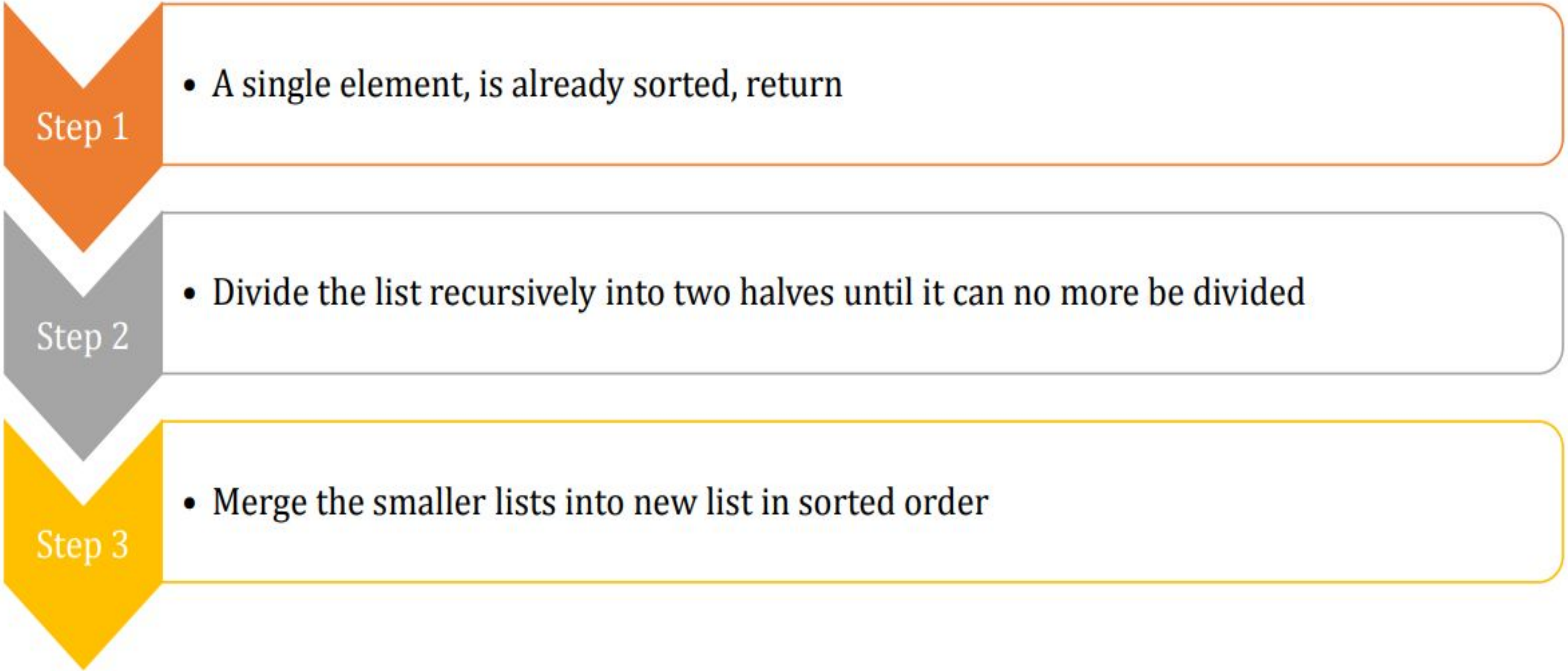
```
OUTPUT:

How many elements you want to sort ? : 6

Enter elements for an array :     78   23   45   8   32   36

After Sorting the elements are :  8    23   32   36  45   78
```

```
#include<stdio.h>

//#include<conio.h>

void main( )

{

 int a[10],i,j,k,n;

// clrscr( );

 printf("How many elements you want to

sort?\n");

 scanf("%d",&n);

 printf("\nEnter the Elements into an array:\n");

 for (i=0;i<n;i++)

 scanf("%d",&a[i]);
```

```
for(i=1;i<n;i++)
 {
 k=a[i];
  printf ("element in K %d\n", k);

 for(j= i-1; j>=0 && k<a[j]; j--)
  //printf ("element in j index %d\n", j);

 a[j+1]=a[j];

 printf ("element in a[j] %d\n", a[j]);
  printf ("element in a[j+1] %d\n", a[j+1]);

a[j+1]=k;
 //printf("element in K after%d\n", k);
 //  printf ("element in K before a[1] %d\n", a[j+1]);


 } printf("\n\n Elements after sorting: \n");
 for(i=0;i<n;i++)
 printf("%d\n", a[i]);
 //getch( );
}
```

# Merge Sort

## Algorithm



Step 1

- A single element, is already sorted, return

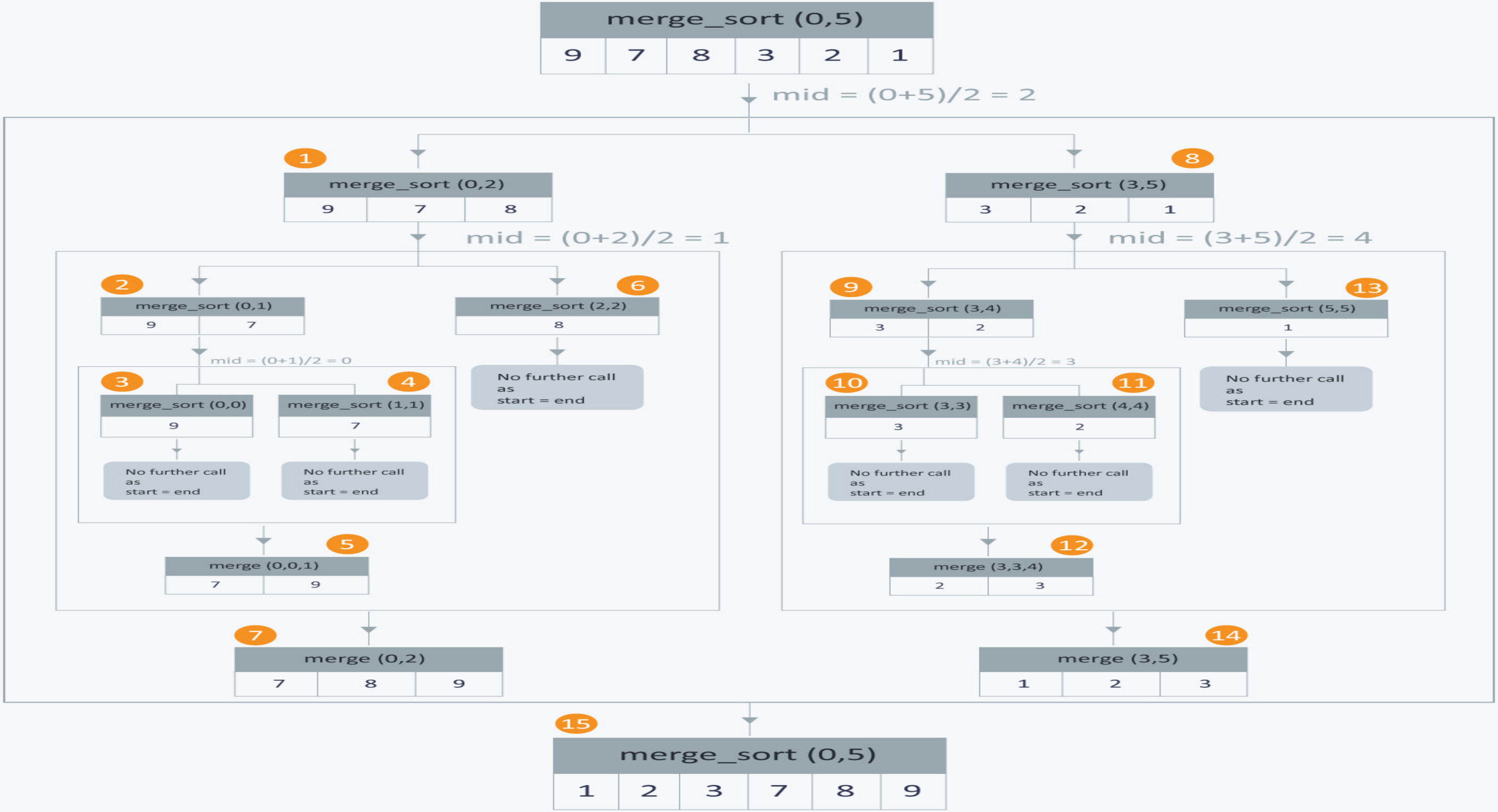
Step 2

- Divide the list recursively into two halves until it can no more be divided

Step 3

- Merge the smaller lists into new list in sorted order

# Merge Sort



merge_sort (0,5)

9 | 7 | 8 | 3 | 2 | 1

mid = (0+5)/2 = 2

1 merge_sort (0,2)

9 | 7 | 8

mid = (0+2)/2 = 1

2 merge_sort (0,1)

9 | 7

mid = (0+1)/2 = 0

3 merge_sort (0,0)

9

No further call
as
start = end

4 merge_sort (1,1)

7

No further call
as
start = end

5 merge (0,0,1)

7 | 9

6 merge_sort (2,2)

8

No further call
as
start = end

7 merge (0,2)

7 | 8 | 9

8 merge_sort (3,5)

3 | 2 | 1

mid = (3+5)/2 = 4

9 merge_sort (3,4)

3 | 2

mid = (3+4)/2 = 3

10 merge_sort (3,3)

3

No further call
as
start = end

11 merge_sort (4,4)

2

No further call
as
start = end

12 merge (3,3,4)

2 | 3

13 merge_sort (5,5)

1

No further call
as
start = end

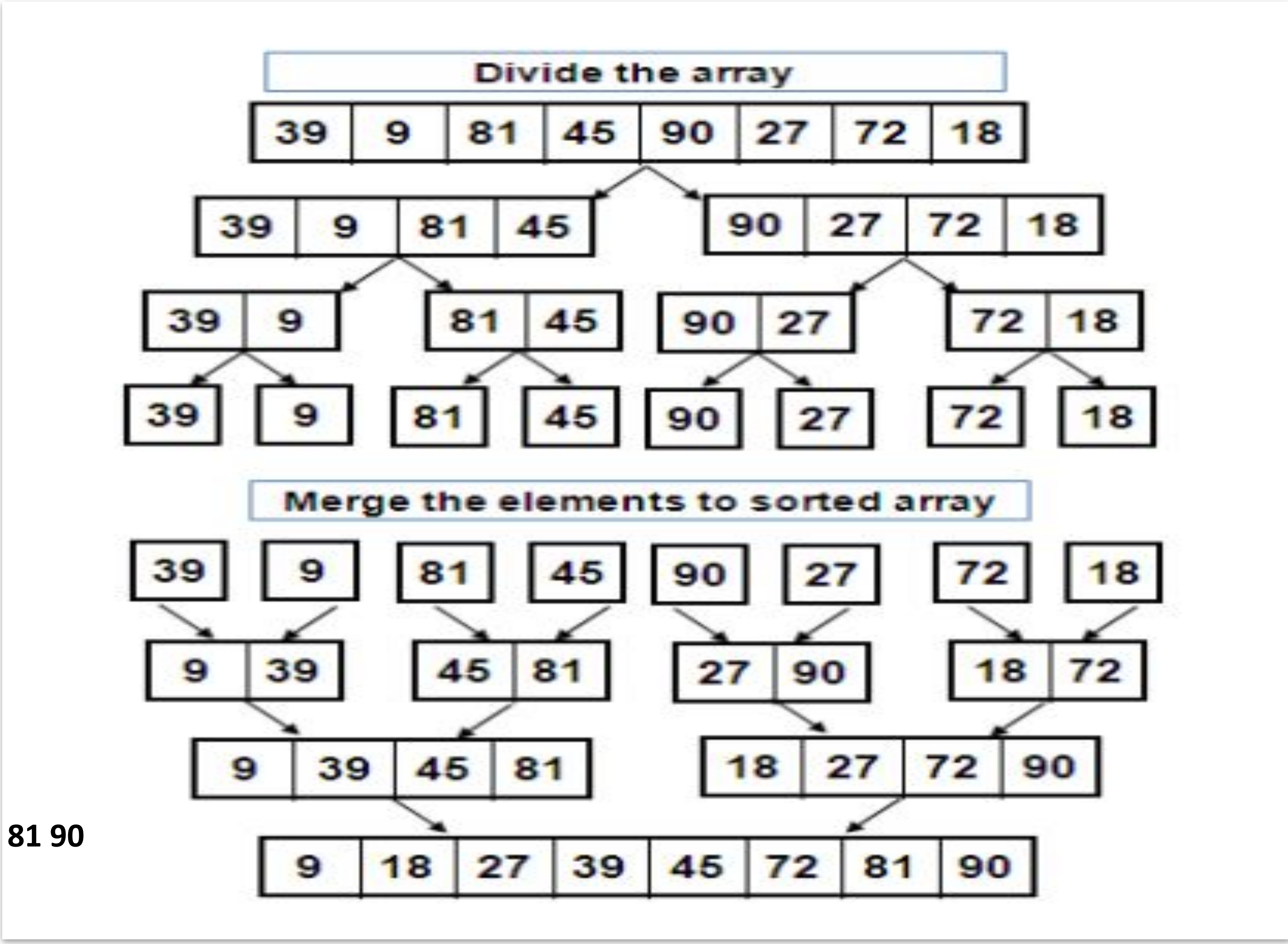
14 merge (3,5)

1 | 2 | 3

15 merge_sort (0,5)

1 | 2 | 3 | 7 | 8 | 9

# EX: Consider Element are: 39 9 81 45 90 27 72 18

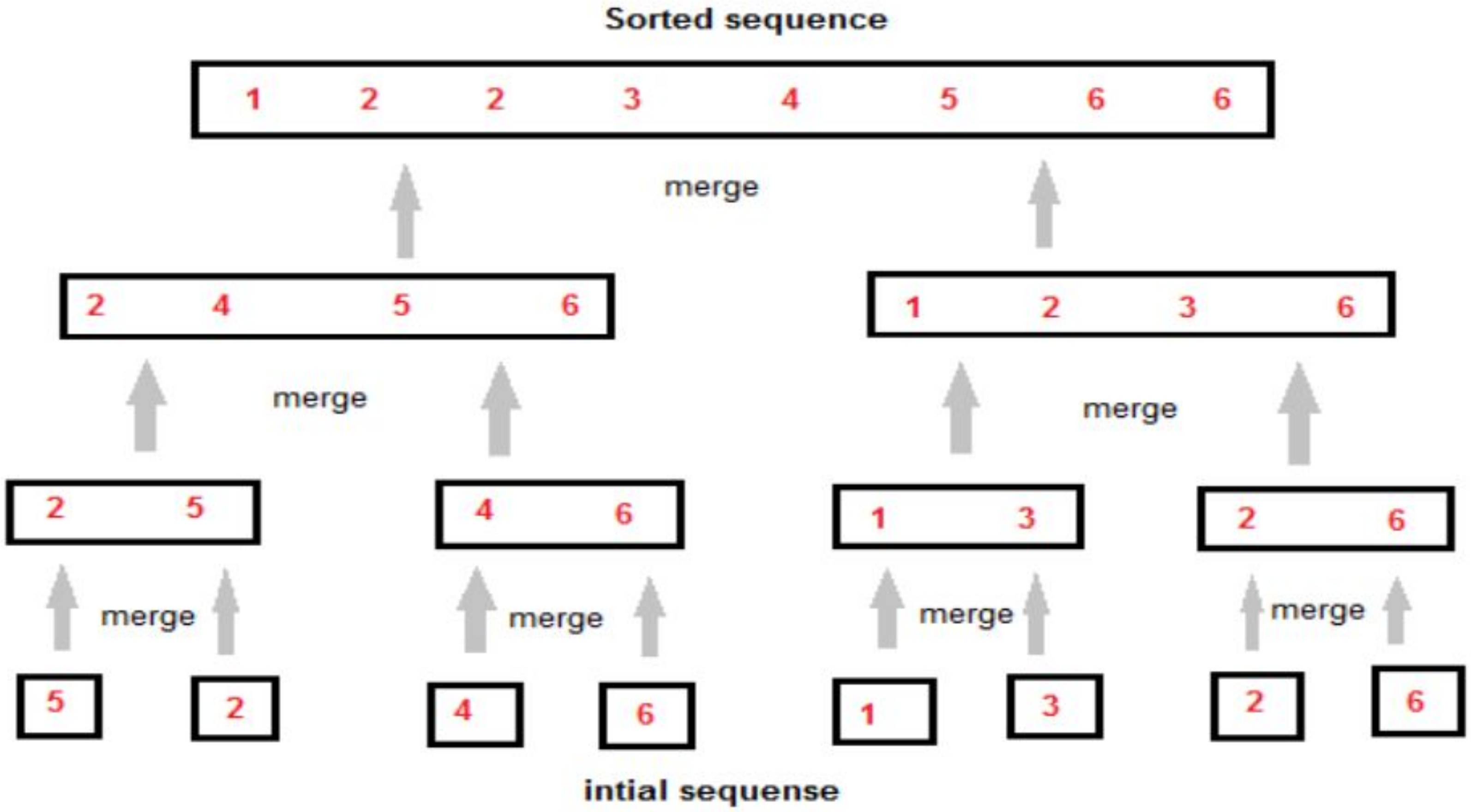


**Sorted elements are: 9 18 27 39 45 72 81 90**

# Merge Sort

Merge sort is a sorting technique based on divide and conquer technique.

**Example:**



Sorted sequence

1 2 2 3 4 5 6 6

merge

2 4 5 6

1 2 3 6

merge

merge

2 5

4 6

1 3

2 6

merge

merge

merge

merge

5 2 4 6 1 3 2 6

intial sequense

# Quick Sort

Quick Sort, as the name suggests, sorts any list very quickly.

## Algorithm

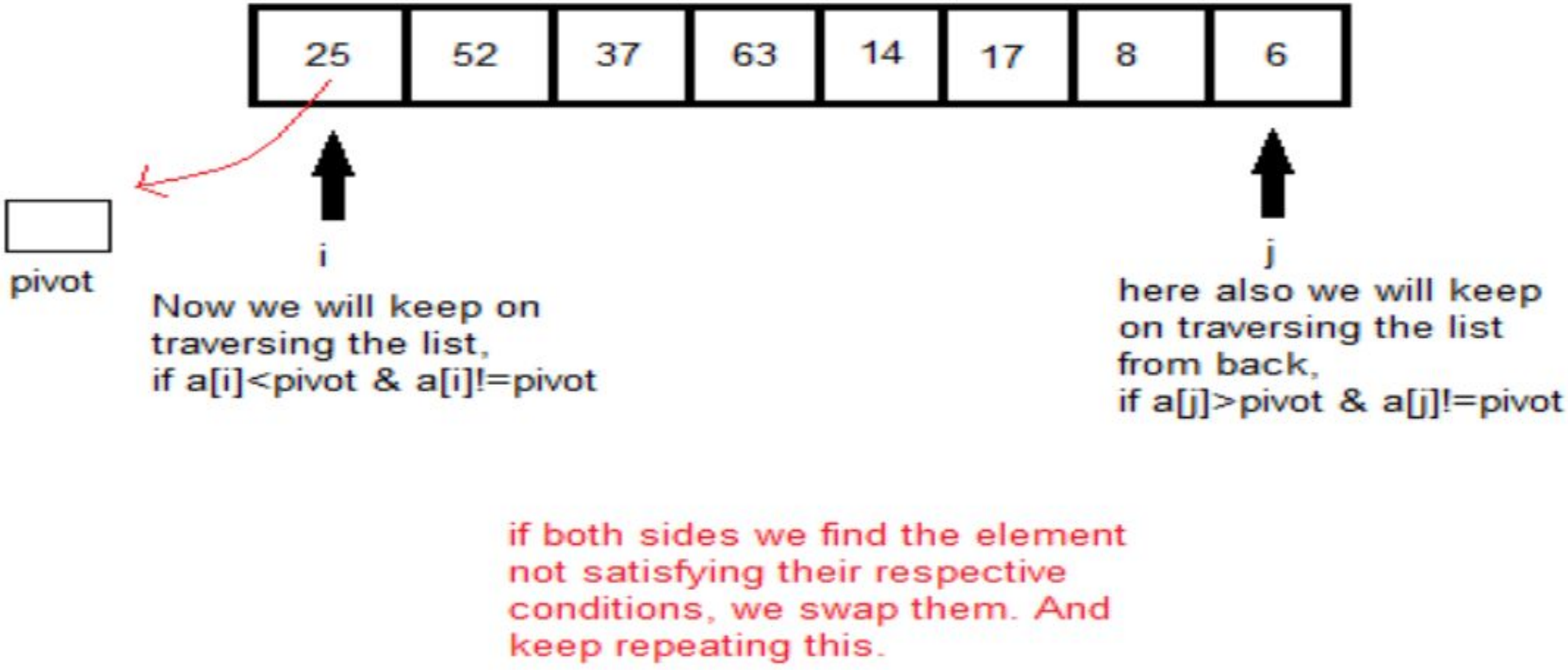
This algorithm divides the list into three main parts :

1) Elements less than the Pivot element
2) Pivot element
3) Elements greater than the pivot element

# Quick Sort

Quick Sort, as the name suggests, sorts any list very quickly.

**Example:**



|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 25 | 52 | 37 | 63 | 14 | 17 | 8 | 6 |

Now we will keep on traversing the list, if a[i]<pivot & a[i]!=pivot

here also we will keep on traversing the list from back, if a[j]>pivot & a[j]!=pivot

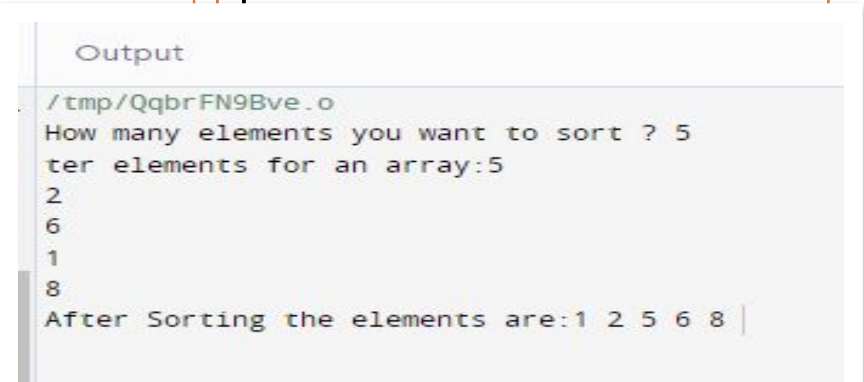
if both sides we find the element not satisfying their respective conditions, we swap them. And keep repeating this.

<u>**DIVIDE AND CONQUER - QUICK SORT**</u>

```
#include<stdio.h>
void quicksort(int[ ],int,int);
void main( )
{
int low, high, pivot, t, n, i, j, a[10];
//clrscr( );
printf("\nHow many elements you want to sort ? ");
scanf("%d",&n);
printf("\Enter elements for an array:");
for(i=0; i<n; i++)
 scanf("%d",&a[i]);
low=0;
high=n-1;
quicksort(a,low,high);
printf("\After Sorting the elements are:");
for(i=0;i<n;i++)
 printf("%d ",a[i]);
```

```
}
void quicksort(int a[ ],int low,int
high)
{
 int pivot,t,i,j;
 if(low<high)
 {
 pivot=a[low];
 i=low+1;
 j=high;
while(1)
 {
    while(pivot>a[i]&&i<=high)
 i++;
 while(pivot<a[j]&&j>=low)
 j--;
 if(i<j) {
```

```
t=a[i];
 a[i]=a[j];
 a[j]=t;
 }
 else
 break;
 }
 a[low]=a[j];
 a[j]=pivot;
 quicksort(a,low,j-1);
 quicksort(a,j+1,high);
}
}
```



Output

```
/tmp/QqbrFN9Bve.o
How many elements you want to sort ? 5
ter elements for an array:5
2
6
1
8
After Sorting the elements are:1 2 5 6 8
```

# Thank you