C++ Presentation 2

By P.P. Krishnaraj

Parts of a function

main function

{

function prototype declaration Return_type function_name(arguments); eg: int add(int);

function call function_name(actual arguments); eg: add(a);

-------;

}

function declaratory/definition Return_type function_name(formal arguments) eg: int add(int X);

{

------;

return statement

}

Default arguments

C++ allows to assign default values to functions parameter(s) which is useful in case a matching argument is not passed in function call statement.

Default values are specified at the time of function declaration.

Suppose int add(int X,int Y,**int Z=10,int S=20**) // **function definition**

```
{
-----;
}
```

Two arguments are default arguments
Syntactically similar to variable initialization
Z=0;
S=0;

function call

```
add(10,20);           //remaining 2 arguments will take default value
add(10,20,30);        //here S will have default value, Z=30
add(10,20,30,40);
```

```
#include<iosream.h>
#include<conio.h>
int add(int,int,int);
int add(int X=10,int Y=20,int Z=30)        //function definition
{
return(X+Y+Z)
}
/* now we will see how many ways we can call this function */
int main()
{
cout<<add();                               //no arguments supplied, so default value is used 60
cout<<add(100);                            // 100 will be used for X i.e 150
cout<<add(50,50);                          //
cout<<add(100,100,100);                    //
getch();
}
```

Default argument should be given from right to left

```
int add(int X,int Y,int Z=10)          //correct

int add(int X=10,int Y,int Z)          //wrong

int add(int X,int Y=10,int Z)          // wrong

int add(int X,int Y=10,int Z=10)            //correct
```

CONSTANT ARGUMENT

- By constant argument , it is meant that the function cannot modify these arguments. If you pass constant argument to the function, then the function cannot modify the values as the values are constant.(eg π=3.14)

```
add (int X,int Y)                    // function definition
{
X=X+10;                              /* here you can change the value of X
                              since  it's a normal argument*/
}
```

```
int main()
{
add (10);
}
                                        // function call
add (const int X)                                          // function definition
{
X=X+10;                                                    /* this statement is not possible since 10's value is
                                                           copied to X and here value of X cant be modified*/
}
```

Note: you can use constant keyword in argument.

add (const int X)

add (const int &X)

add (const int *X)

FUNCTION OVERLOADING

- In C++ we are given a facility which enable us to make many functions of same name for doing many tasks. This is called as function overloading.

```
int calc(void)
int calc(void)
```

/* this is not possible i.e function name can be same but in parameters lists number, name, size should be different*/

NOTE: Function name is same but parameter list is different

```
int calc(void)
int calc(int a)
int calc(int a,int b)
int calc(float a)              // one argument but type is different
```

Program to illustrate the concept of function overloading

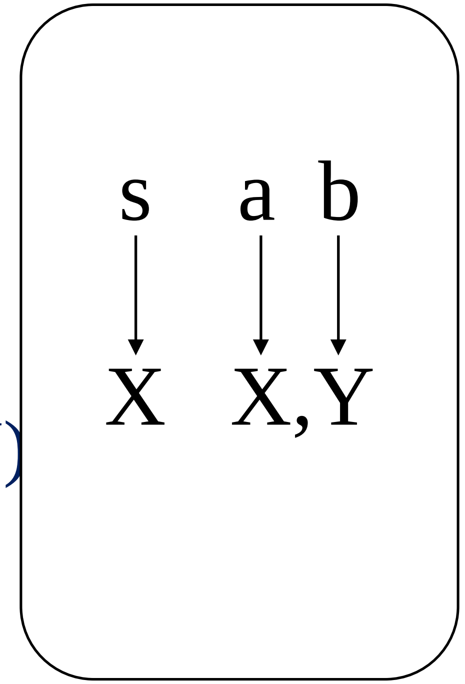
```
#include<iostream.h>
#include<conio.h>
int calc(int);      //function prototyping
int calc(int,int);  //function prototyping

int main()
{
int s,a,b;
cout<<“enter a number”;
cin>>s;
cout<<“square of ”<<s<<“is”<<calc(s);

cout<<“enter 2 numbers”;
cin>>a>>b;
cout<<“addition of 2 number is”<<calc(a,b);
return 0;
}
```

```
int calc(int X)
{
return(X*X);
}

int calc(int X,int Y)
{
return(X+Y);
}
```



FUNCTION RECURSION

If a function calls itself in a function definition it is called as function recursion

```
int main()
{
add();                          //function call
}


int add(void) //function definition
{
add();                          //function calls itself, This statement is called as recursion
}
```

Program to illustrate the concept of function recursion

```
  #include<iostream.h>
  #include<conio.h>
  int fact(int);              //function prototyping
  int main()
  {
  int no;
  cout<<“enter a number”;
  cin>>no;
  cout<<“factorial of”<<no<<“is”<<fact(no);                //function call i.e value returned in function definition
                                                           will be printed here

  return 0;
  }
int fact(int n);          //function definition
{
if(n==0)
{
return 1;
}
return(n*fact(n-1));
}
```

SORTING

The process of arranging the elements of an array in numerical order from highest to lowest values(descending order) or ascending order is called as sorting .if the array contains strings alphabetical order is needed.

0	1	2	3	4
5	**3**	**8**	**2**	**1**

1	**2**	**3**	**5**	**8**

1.SELECTION SORTING

Step 1: 1st index is selected .i.e i=0
Step 2: searching is done for smallest element of the indices. i.e 1
Step 3: swapping is done b/w 5 and 1.

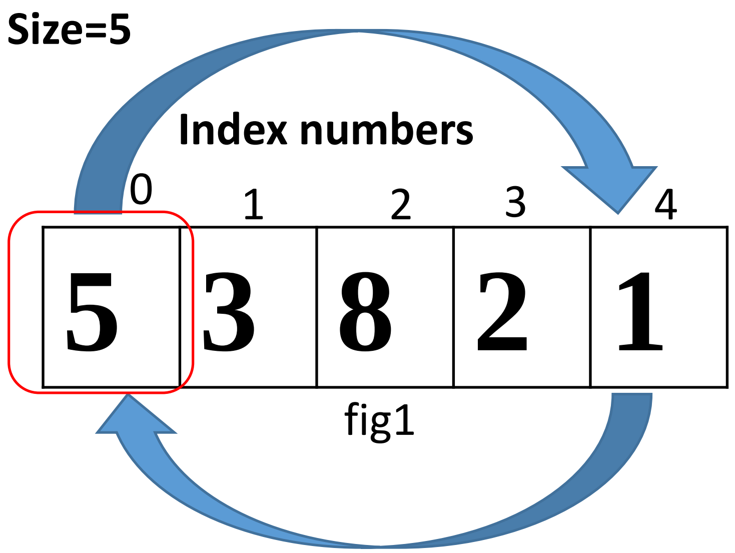


fig1

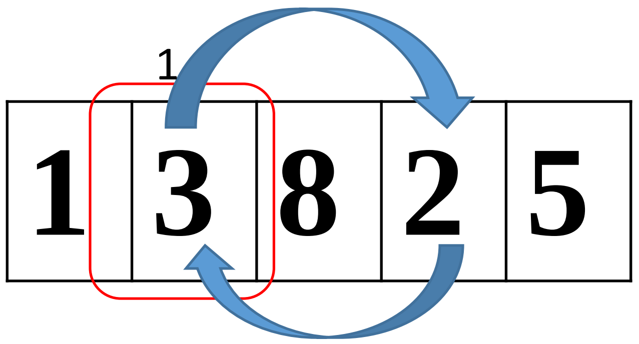


fig2

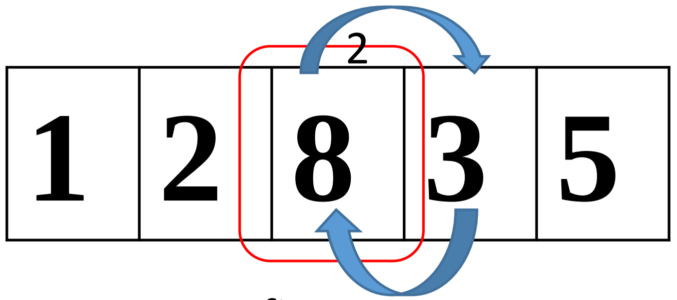


fig3

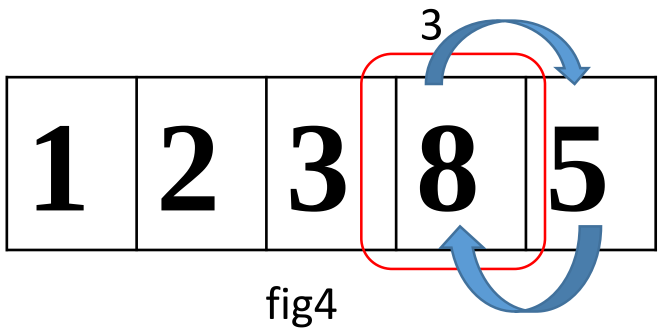


fig4

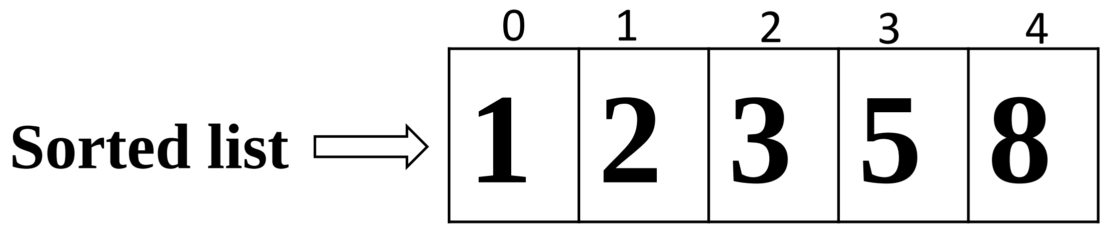
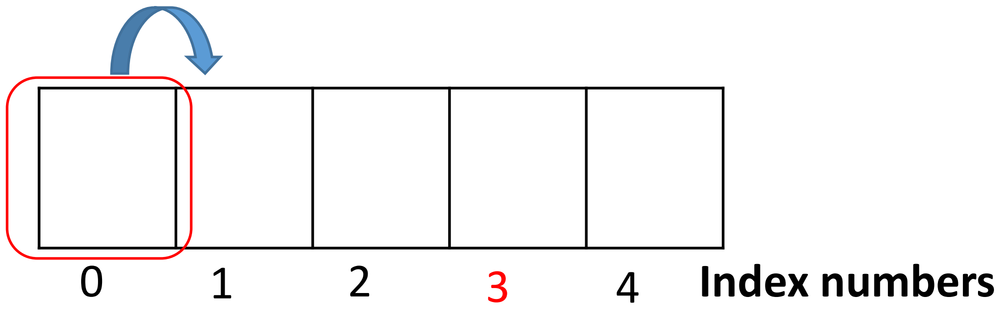


fig5

NOTE: Index number from 0 to 4 Selection done till i=3 (no need for further selection because we will get sorted list after this)

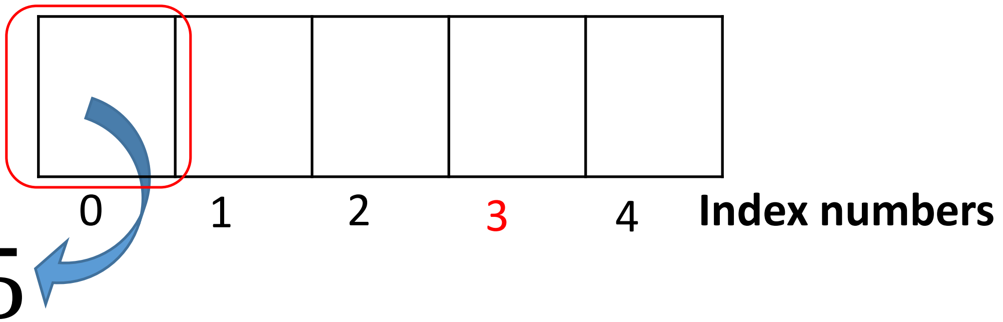
2 for loops needed

```
for(i=0; i<(size-1); i++) //outer for loop
{          <5-1
           <4
           =3
```



NOTE: Index number from 0 to 4 Selection done till i=3 (no need for further selection because we will get sorted list after this)

```
for(j=i+1; j<size; j++) //inner for loop
    =0+1     <5
             =4
{
```



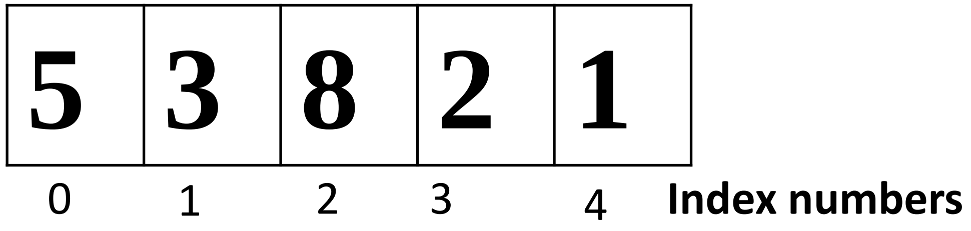
min=5

loc=0

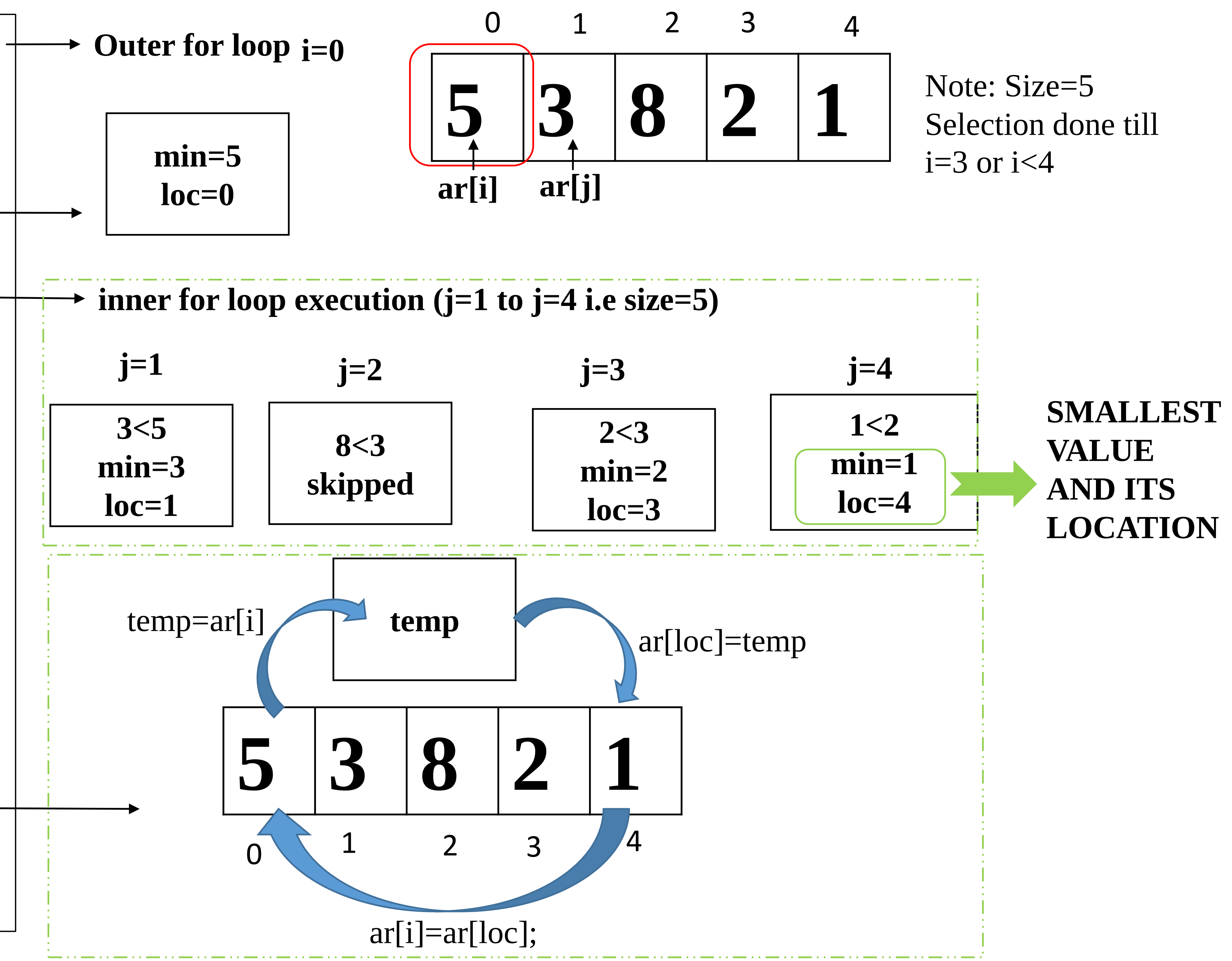
SELECTION SORTING

```
#include<iostream.h>
#include<conio.h>
int main()
{
int ar[20],i,j,size,temp,loc,min;

cout<<"enter size of array";
cin>>size;
cout<<"enter array elements";
for(i=0;i<size;i++)
{
cin>>ar[i];
}
Continued….
```



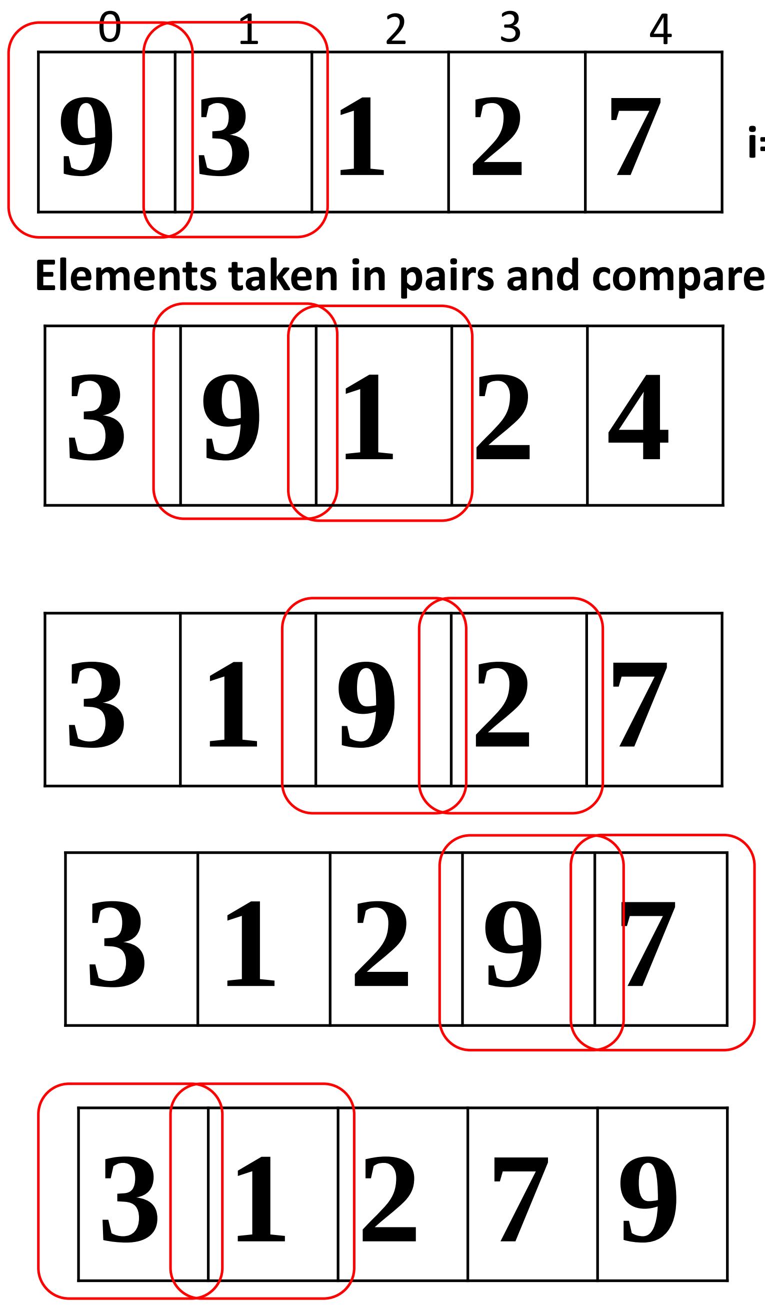
```
for(i=0;i<(size-1);i++)
{
min=ar[i];
loc=i;
   for(j=i+1;j<size;j++)
    {
    if(ar[j]<min)
    {
     min=ar[j];
     loc=j;
     }
     }
temp=ar[i];
ar[i]=ar[loc];
ar[loc]=temp; }
```



BUBBLE SORTING

Index numbers

Size=5



i=0/i=1 1st pass

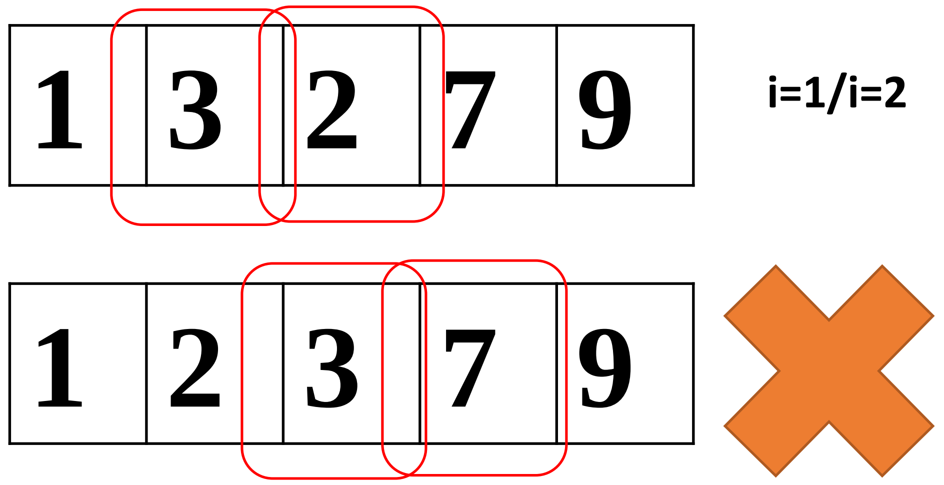
i=1/i=2

i=2/i=3

i=3/i=4

After
1st pass
result

i=0/i=1 2nd pass



i=1/i=2

NOTE: after 1st pass correct elements is in i=4
In ascending/ descending sequence last index is sorted 1st .

- Maximum number of passes=size-1
- If size=5 ; Maximum passes=4
- Minimum can be <4 (previous eg. Passes=2).
- Elements taken in pair and compared.
- Elements sorting get started from last index.

logic

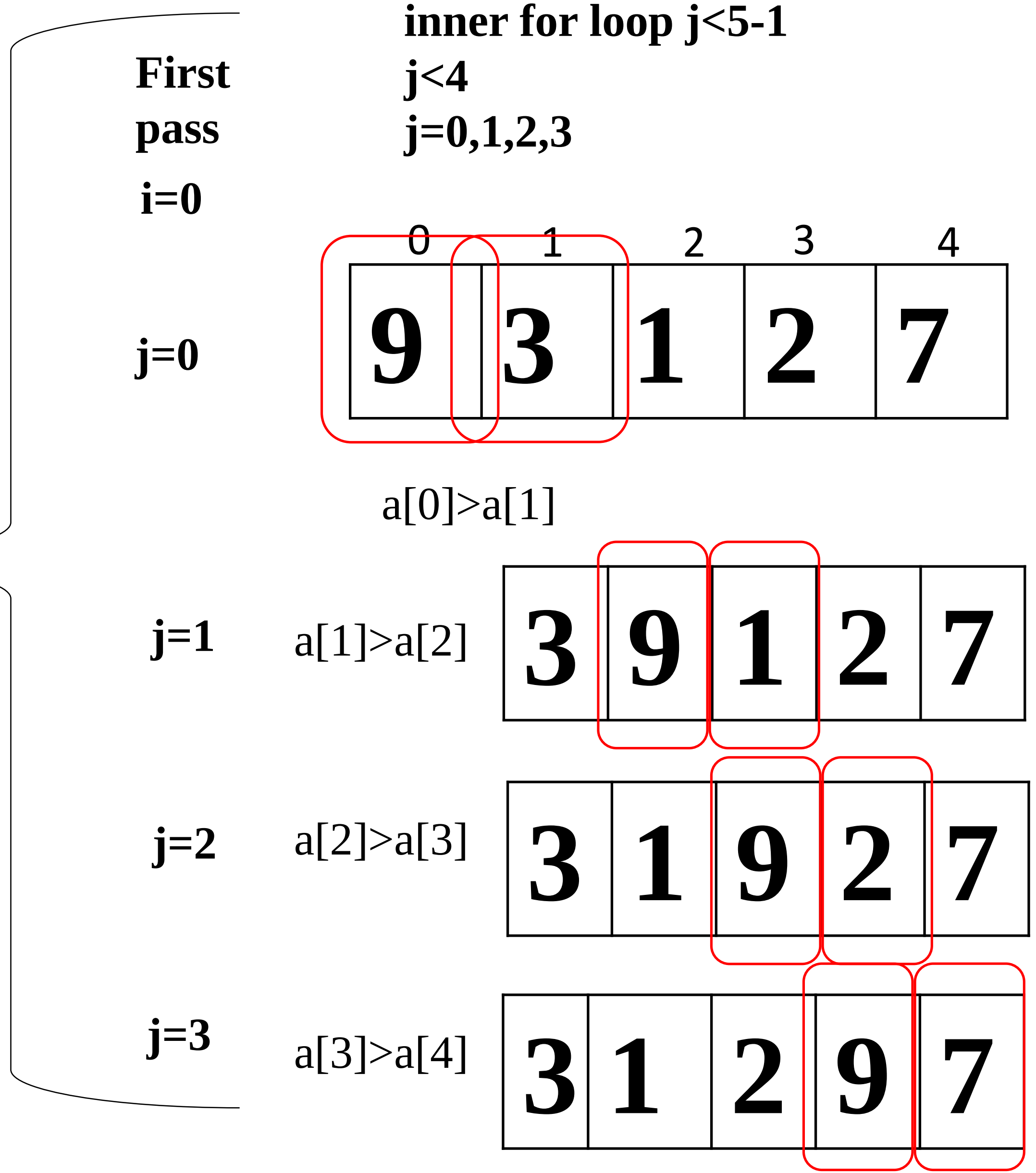
```
for(i=0;i<(size-1);i++)
{
              i<4
/* max no of passes=4 i.e i=0 to i=3*/

for(j=0;j<(size-(i+1);j++)
{ /* 2 elements are taken in pair and compared*/
if a[j]>a[j+1]
{
swap;
}
```

Size=5

no of passes
Outer for loop i=0

inner for loop j<5-1
j<4
j=0,1,2,3



BUBBLE SORTING

```
#include<iostream.h>
#include<conio.h>
int main()
{
int ar[20],i,j,size,temp,temp,swap;

cout<<”enter size of array”;
cin>>size;
cout<<”enter array elements”;
for(i=0;i<size;i++)
{
cin>>ar[i];
}
```

Continued….

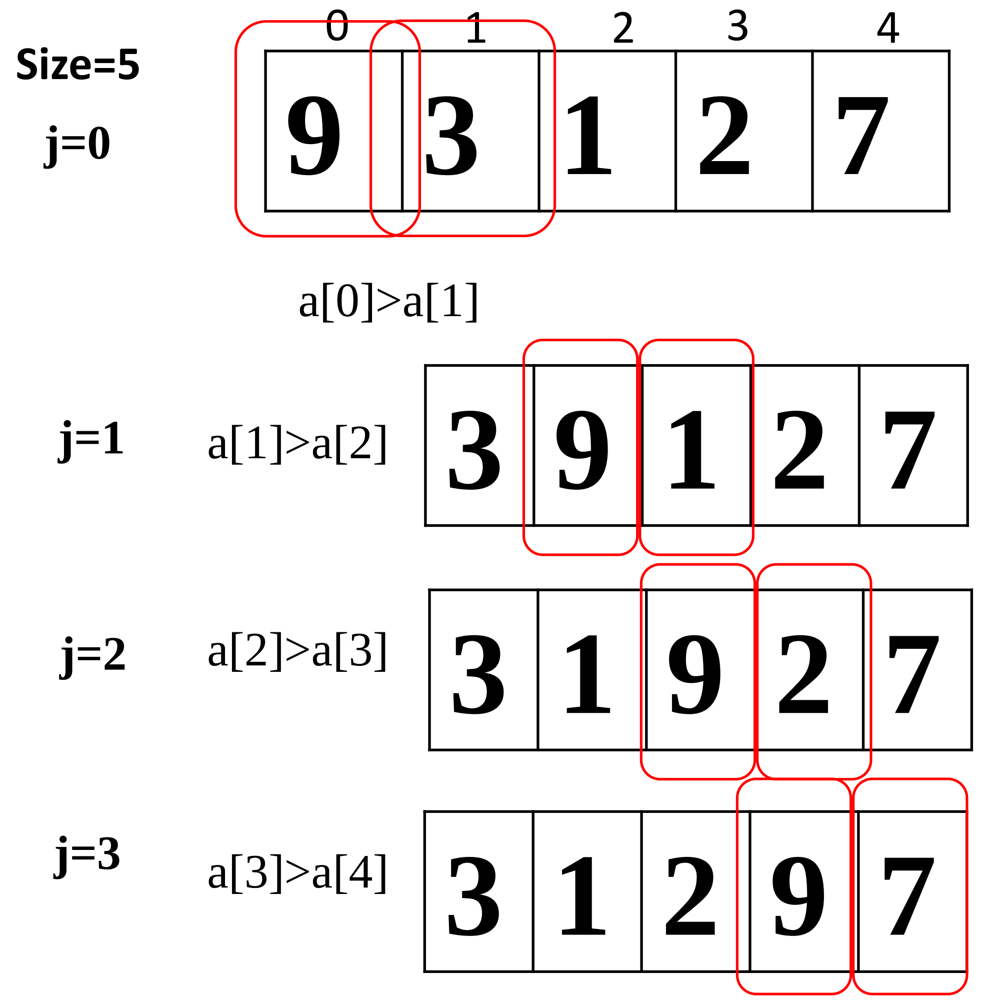
```
for(i=0;i<(size-1);i++)
{
swap=0;

   for(j=0;j<size-(i+1);j++)
    {
    if(a[j]>a[j+1])
    {
     temp=a[j];
     a[j]=a[j+1];
     a[j+1]=temp;
     swap=1;
     }
     }
if (swap==0)
{
break;
}
}
```

i=0,1,2,3
Max no of passes=size-1

COMPARING

SWAPING



After coming from inner for loop, swap value is checked i.e whether swaping is done or not.

INLINE FUNCTIONS

- C++ **inline** function is powerful concept that is commonly used with classes. If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time.
- To inline a function, place the keyword **inline** before the function name and define the function before any calls are made to the function.
- The compiler can ignore the inline qualifier in case defined function is more than a line.
- A function definition in a class definition is an inline function definition, even without the use of the **inline** specifier.

Parts of a function

main function

{

function prototype declaration

Return_type function_name(arguments);   eg: int add(int);

function call

function_name(actual arguments);   eg: add(a);

-------;

}

function declaratory/definition

Return_type   function_name(formal arguments)   eg: int add(int X);

{

------;

return statement

}

```
#include <iostream>
#include<stdio.h>

 inline int Max(int x, int y)
 {
 return (x > y)? x : y;
 }


 int main()
 {
cout << "Max (20,10): " << Max(20,10) << endl;
cout << "Max (0,200): " << Max(0,200) << endl;
cout << "Max (100,1010): " << Max(100,1010) << endl;
 return 0;
}
```

```
Max (20,10): 20
Max (0,200): 200
Max (100,1010): 1010
```

Searching in matrix

```
#include<iostream.h>
#include<conio.h>
void main()
{
clrscr();
int ar[10],n,num,no;
cout<<"Enter size of the array: ";
cin>>n;
cout<<"Enter array element: "<<endl;
for(int i=0;i<n;i++)
{
cout<<"Enter element "<<i<<": ";
cin>>ar[i];
}
cout<<"Elements of array: "<<endl;
for(i=0;i<n;i++)
cout<<"Element at index number "<<i<<" is: "<<ar[i]<<endl;
cout<<"Enter number to search: "; //searching
cin>>num;
for(i=0;i<n;i++)
{
if(num==ar[i])
{
cout<<"Found at index number: "<<i;
no=0;
break;
}
else
{
no=1;
continue;
}
}
if(no==1)
cout<<"Sorry your search returned NO results. “;
getch();
}
```