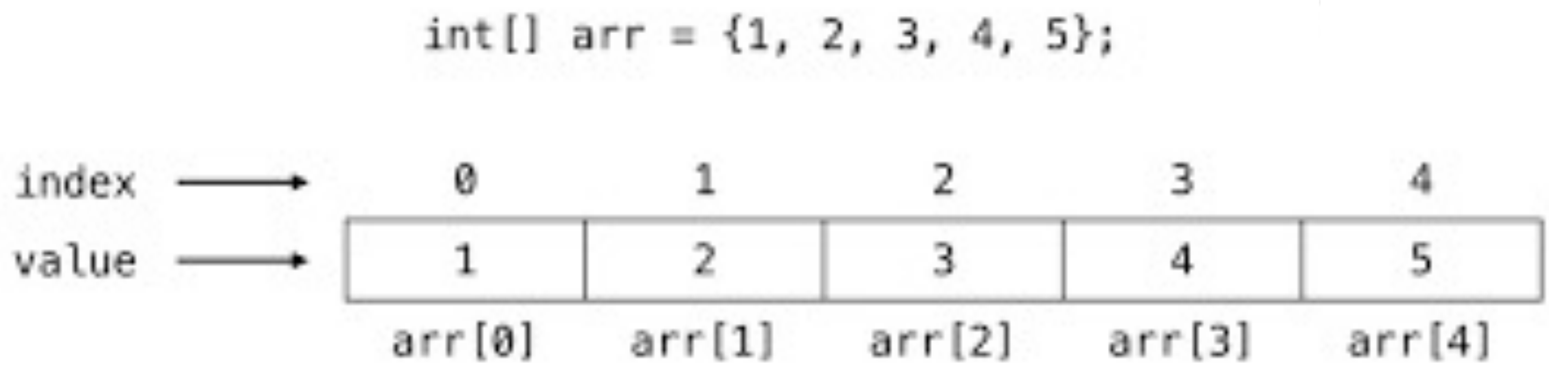
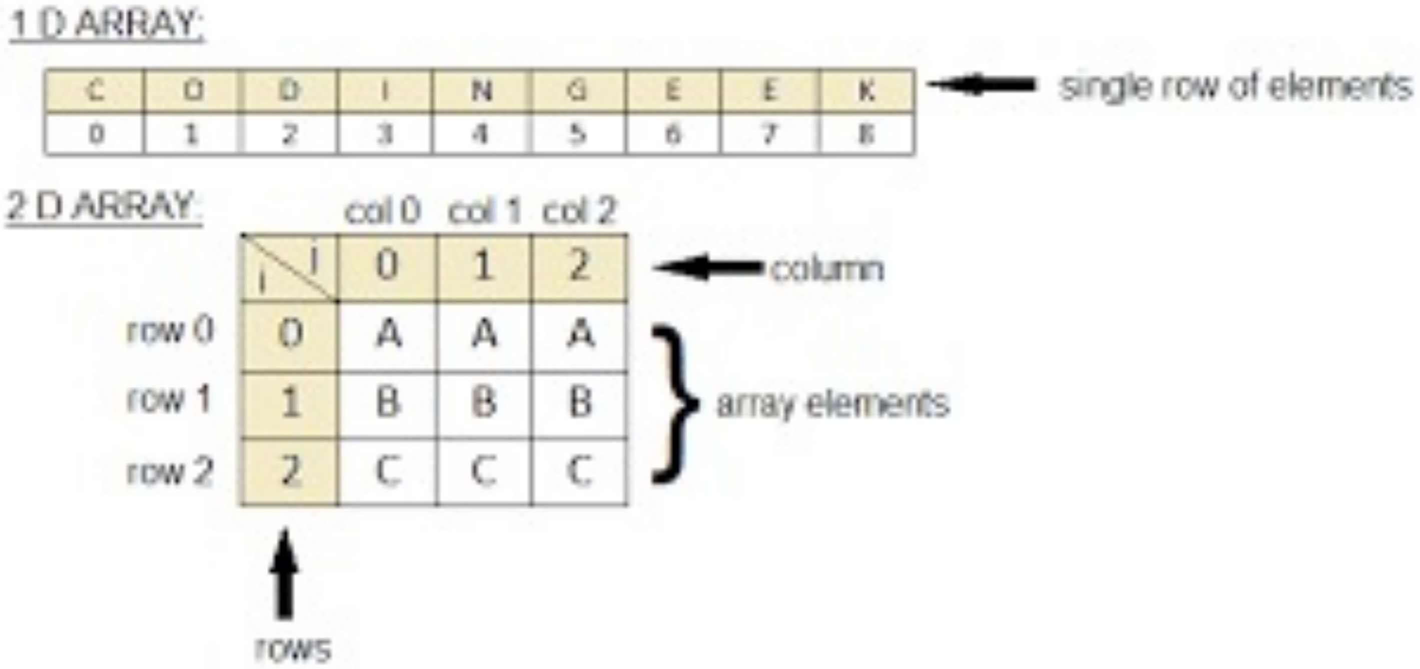
Array



```
int[] arr = {1, 2, 3, 4, 5};
```

index	0	1	2	3	4
value	1	2	3	4	5
	arr[0]	arr[1]	arr[2]	arr[3]	arr[4]



1 D ARRAY:

C	O	D	I	N	G	E	E	K
0	1	2	3	4	5	6	7	8

2 D ARRAY:

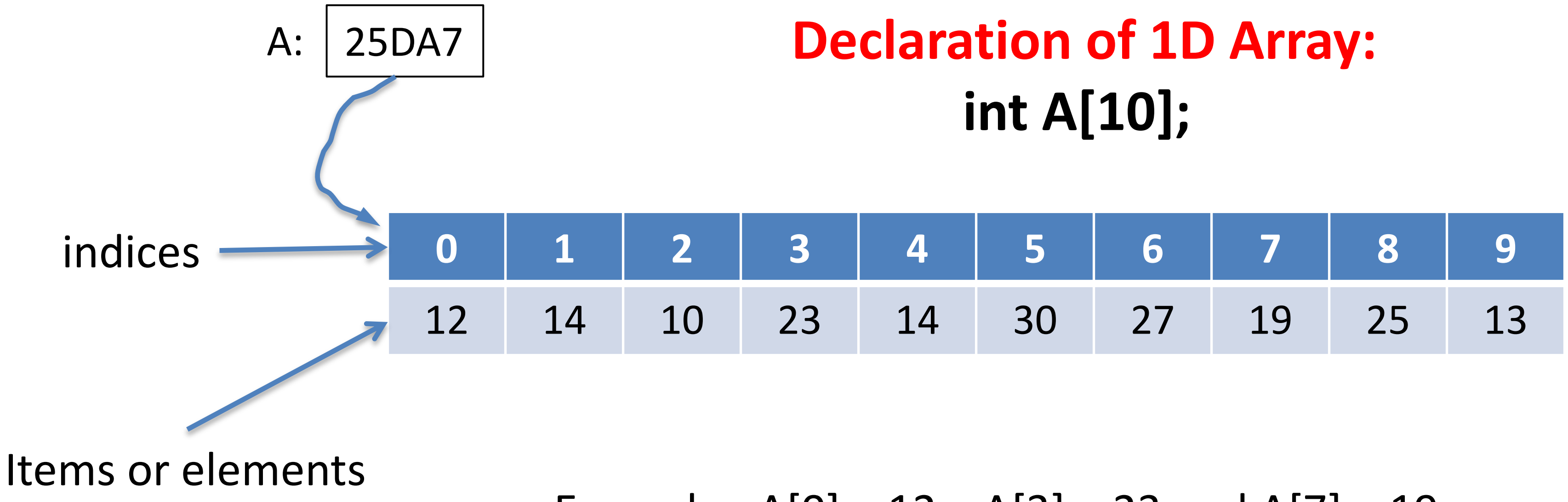
	i \ j	0	1	2
row 0	0	A	A	A
row 1	1	B	B	B
row 2	2	C	C	C

What is Array?

- An array is **a collection of items** stored at **contiguous memory location** and elements can be **accessed randomly using the indices**.

Declaration of 1D Array:

int A[10];



A: 25DA7

indices

0	1	2	3	4	5	6	7	8	9
12	14	10	23	14	30	27	19	25	13

Items or elements

Example: A[0] = 12, A[3] = 23 and A[7] = 19.

A[i]: A is the array name.
i is the index.

Operations on Array

Different operations on 1D Array:

1. **Taking input** into an array
2. **Printing** elements of an array on screen
3. **Sorting** an array
4. **Searching** an element in a array

   (sorted array & unsorted array)
5. **Inserting** an element into an array

   (sorted array & unsorted array)
6. **Deleting** an element from an array

   (sorted array & unsorted array)

Two variables are mandatory for operation on array: (A, N)

- ➢ Array name, say A
- ➢ Number of elements in an array, say N

Taking input into an Array

When number of elements, N is known:

```
for (i = 0; i < N; ++i){
      printf(“Enter number %d: “, i+1);
      scanf(“%d”, &A[i]);
}
```

When N is unknown, stopping criteria is known:

```
i = 0;
printf(“Enter number %d: “, i+1);
scanf(“%d”, &x);
while ( x != 0){
      A[i] = x;
      ++i;
      printf(“Enter number %d: “, i+1);
      scanf(“%d”, &x);
}
```

There must have some stopping criteria

Printing an Array

While printing an array, N is definitely known

```
printf(“Elements of the list: \n “);
for (i = 0; i < N; ++i)
    printf(“%d”, A[i]);
```

Swapping

How can we put **water to whisky glass** and **whisky to normal glass**?

A[i]: 10

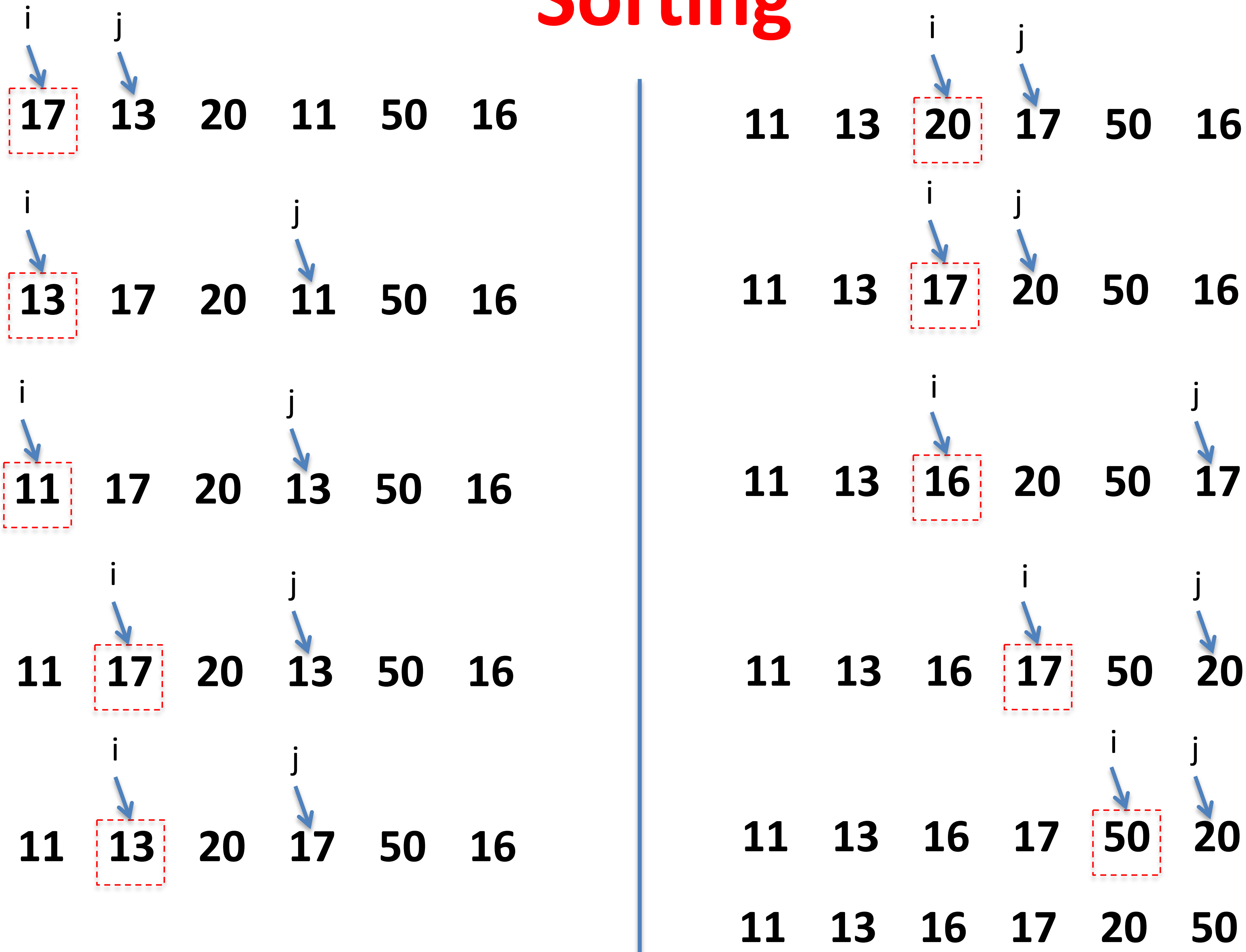
A[j]: 17

temp:

Swapping: give something in exchange of something else

```
temp = A[i];
A[i] = A[j];
A[j] = temp;
```

Sorting



i j

17 13 20 11 50 16

i j

13 17 20 11 50 16

i j

11 17 20 13 50 16

i j

11 17 20 13 50 16

i j

11 13 20 17 50 16

i j

11 13 20 17 50 16

i j

11 13 17 20 50 16

i j

11 13 16 20 50 17

i j

11 13 16 17 50 20

i j

11 13 16 17 50 20

11 13 16 17 20 50

Sorting

N

0	1	2	3	4	5	6	7	8	9
12	14	10	23	14	30				

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

A[i] > A[j]   => Ascending order

A[i] < A[j]   => Descending order

Sorting Program

```
#include <stdio.h>
void main(){
    int A[50], N, i, j, temp;

    printf(“Enter the value of N: “);
    scanf(“%d”, &N);

    for( i = 0;  i < N; ++i){
        printf(“Enter number %d”, i+1);
        scanf(“%d”, &A[i]);
    }

    for ( i = 0; i < N-1; ++i)
        for( j = i+1; j < N; ++j)
            if ( A[i] > A[j] ){
                temp = A[i];
                A[i] = A[j];
                A[j] = temp;
            }

    printf(“The sorted list:\n”);
    for( i = 0; i < N; ++i){
        printf(“%d”, A[i]);
    }
}
```

Searching

When elements are sorted:

N

0	1	2	3	4	5	6	7	8	9
12	14	16	23	24	30				

```
while ( (A[i] < x) && (i < N) ) i++;
```

```
int  SearchSorted( int x ){
        int i = 0;

        while ( (A[i] < x) && (i < N) ) i++;
        if ( x == A[i] ) return i;
        return -1;
}
```

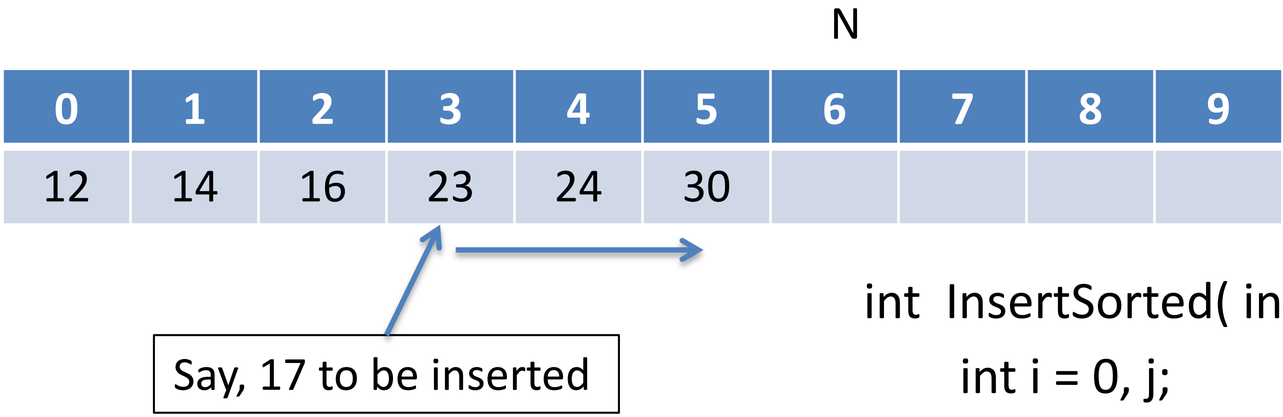
When A and N are global variable.

When elements are unsorted:

```
while ( ( x != A[i]) && (i < N) ) i++;
```

Inserting

When elements are sorted:



N									
0	1	2	3	4	5	6	7	8	9
12	14	16	23	24	30				

Say, 17 to be inserted

Steps:

1. **Search the location**
2. **Vacant the place**
3. **Insert the element**

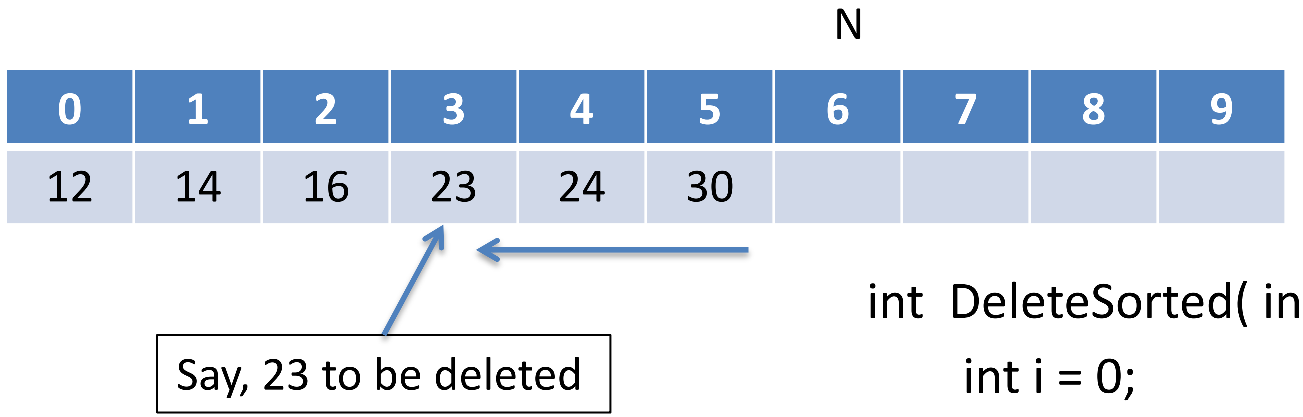
```
int  InsertSorted( int x ){
        int i = 0, j;

        while ( (A[i] < x) && (i < N) ) i++;

        N++;
        for ( j = N-1; j > i; --j)  A[j] = A[j-1];
        A[i] = x;
}
```

When elements are unsorted:

```
int InsertUS( int x){
      A[N++] = x;
}
```

Deleting

When elements are sorted:



N

0	1	2	3	4	5	6	7	8	9
12	14	16	23	24	30				

Say, 23 to be deleted

```
int  DeleteSorted( int x ){
        int i = 0;

        while ( (A[i] < x) && (i < N) ) i++;

        for ( ; i < N - 1; ++i)  A[i] = A[i+1];
        N--;
}
```

Steps:

1. **Search the location**
2. **Shift the data**
3. **Reduce N**

When elements are unsorted:

```
while ( (x != A[i]) && (i < N) ) i++;
```

Extra Operation

Sum up elements :

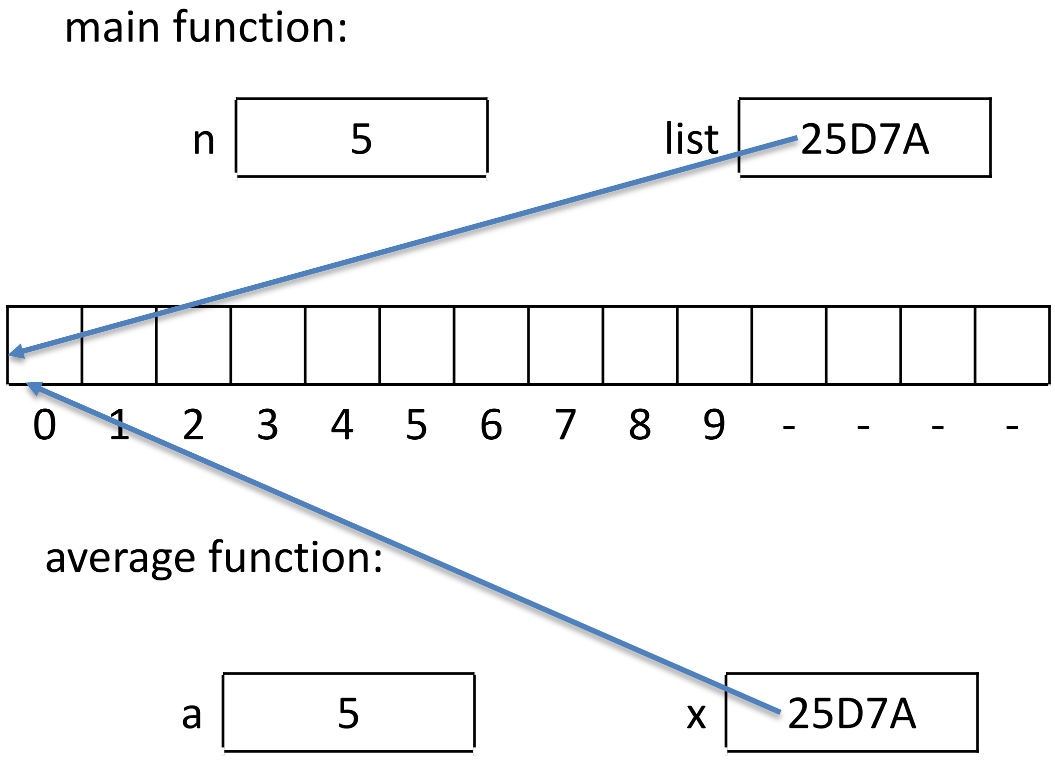
```
Sum = 0;
For ( i = 0; i < N; ++i )
    sum += A[i];
```

Product of elements :

```
mult = 1;
For ( i = 0; i < N; ++i )
    mult *= A[i];
```

Passing An Array to a Function

```
Float average (int a, float x[]);

int  main(){
       int n;
       float avg, list[100];

        ----------------------------
        avg = average(n, list);
        ----------------------------
}

Float average (int a, float x[]){
        ----------------------------
        ----------------------------
}
```



main function:

n 5

list 25D7A

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

average function:

a 5

x 25D7A

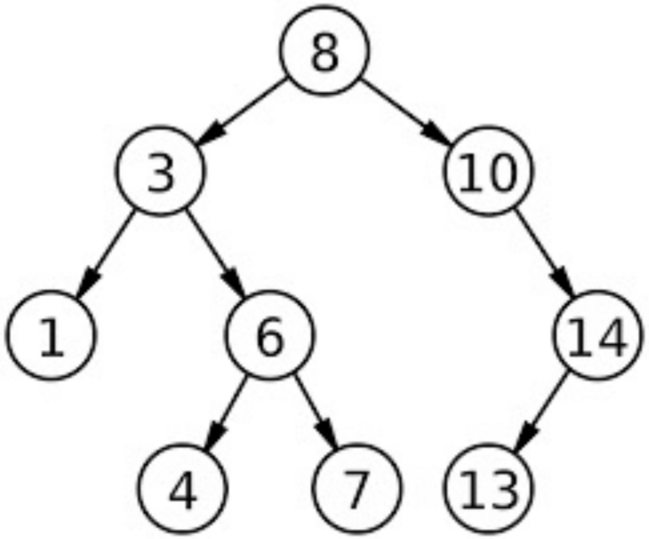
Assignment -1

Write a menu-driven C program for maintaining a 1D integer array. The menu of the program will have the following options.

Choose-
1 for taking input
2 for display
3 for sorting
4 for inserting
5 for appending
6 for deleting
7 for searching
8 for exit
Enter your choice:

Please note that insertion, deletion and searching will consider whether the list is sorted or not to reduce the number of comparison. Again, insertion will be performed in such a way that sorted-list will remain sorted in each step.

Assignment -2



Binary search tree (BST) can be represented by a 1D array using the following rules:

(a) The root is placed at index 1.

(b) If a node is in index i, then its left child is placed at index 2i and the right child is placed at index 2i + 1

(c) -1 indicates no value is available in the tree.

8	**3**	**10**	**1**	**6**	**-1**	**14**	**-1**	**-1**	**4**	**7**	**-1**	**-1**	**13**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Write a menu-driven C program that can –

1. Insert a new value into the tree.
2. Provide the height of a node containing the value inputted from keyboard (assume that height of root is zero).
3. Show listing of values of the tree in (a) in-order; (b) pre-order; and (c) post-order traversal
4. Delete an element from the tree
5. Provide the maximum height of the tree using recursive function.

Assignment -2

Algorithm for BST node deletion:

Case 1: ***Node to be deleted is the leaf:*** Simply remove from the tree.

Case 2: ***Node to be deleted has only one child:*** Copy the child to the node and delete the child

Case 3: ***Node to be deleted has two children:*** Find inorder successor of the node. Copy contents of the inorder successor to the node and delete the inorder successor.