**WEEK 1&2**

**AIM:** Array and its various operations with examples

**Description**: A menu-driven program where there are 8 choices given to the user to perform various array operations.

**Variables Used**:

**size**: size of the array

**arr**: name of the array

**choice**: choice value entered by the user

**pos**: positon entered by the user

**elem**: element value entered by the user

**result**: storing the result returned by non-void methods

**Initialiy, arr =** {10, 20, 30, 40, 50, 60}

**size** = 6

**Array operations available to the user as a choice in the menu-driven program:**

**1. Initialization:** initialize the array by taking size and array element as input from the user

Proc Initialize(arr, size):

For i =1 to size do

arr[i]<- input()

EndFor

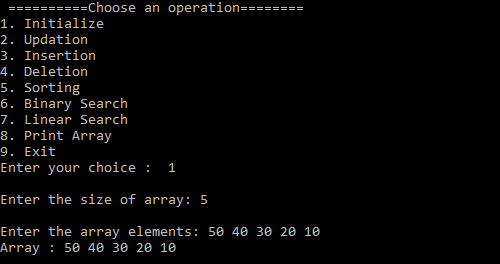
End

Input: 5 // size of array

50 40 30 20 10 // array elements

Output: size = 5

arr = {50, 40, 30, 20, 10}



**2. Updation:** updated the value at a specific position of the array

Time Complexity: O(1)

Proc Updation(arr, size, pos, elem):

If pos<1 and pos>size then

Print “invalid position”

Exit

Else

arr[pos] = elem

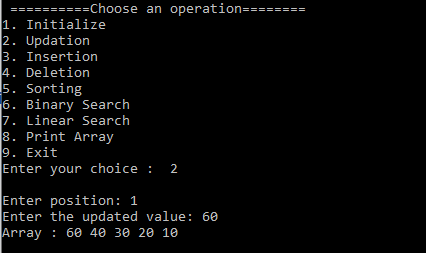
EndIf

End

Input: 1 // position

60 // updated value

Output: arr = {60, 40, 30, 20, 10}



**3. Insertion**: inserts a value at a specific position of the array and shifts the other elements accordingly

Time Complexity: O(n-k) n: size of array

k: the position where insertion happens

Proc Insertion(arr, size, pos, elem)

If pos<1 and pos>size then

Print “invalid position”

Exit

Else

For i = size to pos+1 do

arr[i] <- arr[i-1]

arr[pos] <- elem

size<-size+1

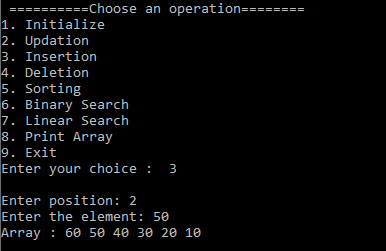
EndIf

End

Input: 2 // position

50 // new element value

Output: arr = {60, 50, 40, 30, 20, 10}

****

**4. Deletion:** deletes element at a specific position of the array and shifts other elements accordingly

Time Complexity: O(n-k) n: size of array

k: the position where deletion happens

Proc Deletion(arr, size, pos)

If pos<1 and pos>size then

Print “invalid position”

Exit

Else

For i = pos to size-1 do

arr[i] <- arr[i+1]

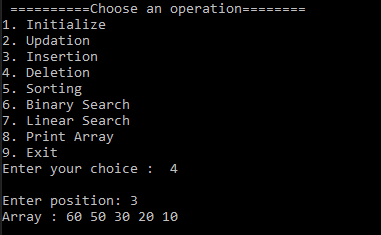
size <- size - 1

EndIf

End

Input: 3 // position where element to be deleted

Output: arr = {60, 50, 30, 20, 10}



**5. Sorting:** sorts the array in ascending order.

Sorting Technique: Insertion Sort (since the array is short in size)

Time Complexity: O(n^2) n: size of array

Proc Sorting(arr, size)

For i = 2 to size do

j <- i-1

key <- arr[i]

While j>=1 and arr[j] > key do

arr[j+1] = arr[j]

j <- j - 1

EndWhile

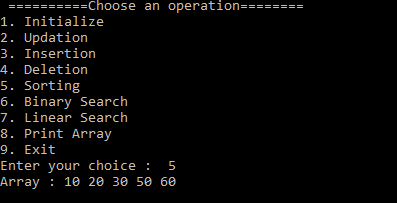
arr[j+1] <- key

EndFor

End

Input: N/A // no external input required

Output: arr = {10, 20, 30, 50, 60}



**6. Binary Search:** searches an element in the array using binary search and returns the position if found otherwise -1

Time Complexity: O(log n) n: size of array

Condition: array must be sorted

Proc Binary\_Search(arr, size, elem)

left <- 1

right <- size

While left <= right do

mid <- (left + right)/2

If arr[mid] = elem then

Return mid

ElseIf arr[mid] > elem then

right <- mid - 1

Else

left <- mid + 1

EndIf

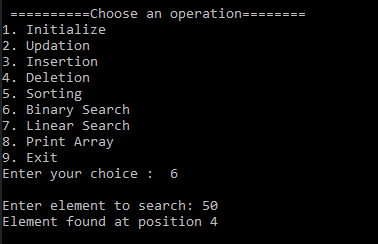
EndWhile

Return -1

End

Input: 50 // element to search

Output: 4



**7. Linear Search:** searches an element in the array in a linear fashion and returns the position of the element if found otherwise returns -1.

Time Complexity: O(n) n: size of array

Proc Linear\_Search(arr, size, elem)

For i = 1 to size do

If arr[i] = elem then

Return i

EndIf

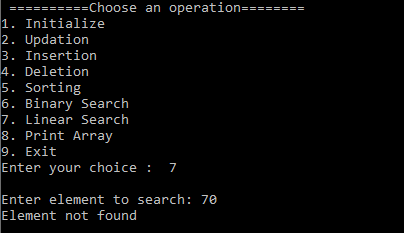
EndFor

Return -1

End

Input: 70 //element to search

Output: -1 // indicated element not found



**8. Print Array:** prints the current state of the array

Time Complexity: O(n) n: size of array

Proc Print\_Array(arr, size)

If size = 0 then

Print “Array is empty”

Exit

For i = 1 to size do

Print arr[i]

EndFor

End

Input: N/A // no external input required

Output: arr = {10, 20, 30, 50, 60}

