DATA STRUCTURE TUTORIAL (U19CS012)
(D-12)

TUTORIAL - 4

Q.1.) Write an Algorithm to

(a) Insert an element in Array

Assumptions : A → Array

N → number of elements in Array

new-ele → new element to insert

max → maximum elements array can store

Error Handling : Check number of elements & maximum size of Array

Assuming 1 based indexing [Simplicity]

(a) ⇒ To insert an element in to an array

Let position be "index"

1) Start
2) if N = max or N > max
   then
      display error
   else
      N = N+1
      for elements i = n-1 to index
         A[i+1] = A[i] // All elements shifted right
      end for
      A[index] = new-ele
3) end if
4) Stop

(b) Insert at Beginning

1.) Begin
2.) If N >= max
then display error
else
N = N+1
for i = N-1 to 1
A[i+1] = A[i]
end for
A[1] = new-ele
end if
3) Stop

(c) Insert an element at End

1.) Start
2.) If N >= max
then display error
else
N = N+1
A[N] = new-ele
end if
3.) Stop

(d) insert an element at position P

1) Start
2) if N >= max
then display error
else
N = N+1
for i = N-1 to i = P
A[i+1] = A[i]
end for
A[P] = new element
end if
3) end

(e) Insert an element before position P

1.) Start
2.) if N >= max
then display error
else
N = N+1
if P<1, display error & return
else
for i = N-1 to P
A[i+1] = A[i]
end for
A[P-1] = new-ele
end if
3.) Stop

1.) (f) Insert an element after Position 'P'

(1) Begin
(2) If N >= max
then
display error

else
N = N+1
for i = N-1 to i = P+1
A[i+1] = A[i]
end for
A[P+1] = new-ele
(3) stop

2.) Write an ~~An~~ Algorithm to delete
Assumptions: A → array, N = no. of elements of array,
Assumed 1 based indexing

(a) Delete an element from Array
let the position be "pos"

1) Start
2) Set i = pos → if C|(pos >= 1 && pos <= N)) → display error
3) REPEAT Steps 4 & 5 while i < N
4) A[i] = ~~A[i]~~ A[i+1]
5) i = i+1
6) N = N-1 // decrease size of array after deletion
7) end

U19CS012 (D12)

(b) delete an element from 1st position

```
1) Begin        if (N<=0) display error
2) set i = 1
3) repeat step 4 & 5 while i<N
4)      A[i] = A[i+1]
5)      i = i+1
6)  N = N-1
7)  End
```

(c) Delete an element from end

```
1)  Begin
2)  if (N<=0)
        display error
    else
        N = N-1
3)  End
```

(d) delete an element at position P

```
1) begin
2) if( !(P>=1 && P<=N) )
        display error // invalid position
   else
        i = P
3)      repeat step 4 & 5 while i<N
4)          A[i] = A[i+1]
5)          i = i+1
6)      N = N-1     // reduce size
7)  end
```

(e) delete an element before position P

```
1) Begin
2) if ( P==1 || P<1 || P>N )
        display error
   else
        i = P-1
3) repeat step 4 & 5 while i<N
4)      A[i] = A[i+1]
5)      i = i+1
6)  N = N-1
7)  Stop
```

(f) delete an element after position P

```
1)  Begin
2)  if ( P==N || P<1 || P>N)
        display error
    else
        i = P+1
3)  repeat step 4 & 5 while i<N
4)      A[i] = A[i+1]
5)      i = i+1
6)  N = N-1
7)  End
```

U19CS012 (D-12)

3) Write an Algorithm to traverse an Array

A → Array N → no. of elements

follows 1 based Indexing

1) Begin
2) set i = 1
3) repeat Step 4 & 5 while i <= N
4) print A[i]
5) i = i+1
6) end

4) Write an Algorithm for Assumption : A → Array in Array N = Number of elements

Key → value to find

LINEAR SEARCH

1) Start
2) Set i = 1, flag = 0
3) FOR i = 1 to N
4) IF A[i] is equal to key
5) print i, flag = 1
6) break the search
7) end for
8) if (flag == 0)
print (element not found)
9) End

O(N) = Time complexity

BINARY SEARCH

Assumption : Array is sorted in Ascending order (left to right) smallest

1) Begin
2) low = 1, high = N
3) Repeat Step 4,5,6,7
WHILE (low < high)
4) mid = low + (high - low)/2 ;
// to prevent overflow
5) IF A[mid] = key
print mid, return
6) ELSE IF A[mid] > key
high = mid - 1
7) ELSE
low = mid + 1
8) print ("No element found")
// control reaches here only if no element found
9) Stop

⇒ O($\log_2(N)$)

SUBMITTED BY :

ROLL NO: U19CS012

(D-12)

Bhagya Rana

vision