MODULE 2

ARRAYS & SEARCHING

EDULINE FOR CSE STUDENTS

ARRAY

- An array is a consecutive set of memory locations. An array is a set of pairs, ie; index & values.
- For each index which is defined, there is a value associated with the index. In mathematical term we call this a correspondence or a mapping.
- Array can be defined as

Structure - ARRAY (value, index)

Declare - CREATE() ⟶ array

RETRIEVE(array, index) ⟶ value

STORE(array, index, value) ⟶ array

For all A is an array, i, j are index and x is value

REPRESENTATION OF ARRAY

❖One Dimensional Array

➢One dimensional array can be represented as follows

A[lower bound : Upper bound]

Eg: A[3:4]

➢The address A[i] can be calculated by

Base address + (i - Lower bound) = $\alpha+(i-L)$

Here; base address is the starting address.

➢Total number of elements can be calculated by

Upper bound – Lower bound+1 = U-L+1

❖Two Dimensional Array

➢It can be represented as

$A[L_1.......U_1 ,\ L_2.......U_2]$

Row = U_1-L_1+1

Column = U_2-L_2+1

- So, Total number of elements= Row*Column

➢The two common ways to represent multidimensional arrays are

1. Row major order
2. Column major order

❖Row Major

As its name implies, row major order stores multidimensional arrays by rows

$A[L_1 \ldots\ldots U_1 , L_2 \ldots\ldots U_2]$

Where U_1 = row representation

U_2 = column representation

U_2 element	U_2 element	U_2 element	…………	
1^{st} row	2^{nd} row	3^{rd} row		U_1 row

Eg:

1	2	3
4	5	6
7	8	9

A [3,3] in C, A[3][3]

U_2 element	U_2 element	…………	
1 2 3	4 5 6		7 8 9
1^{st} row	2^{nd} row		U_1 row

we can find the address of A[i, j] using row major,

Base address + $(i-L_1)U_2 + (j-L_2)$

Where, U_2 represents column

Eg: A[5:7 , 2:4] find the address of A[6,3] , base address α=10.

$$A\,[\,\overset{L_1}{5} : \overset{U_1}{7}\,,\,\overset{L_2}{2} : \overset{U_2}{4}\,]$$

m (row) = $U_1 - L_1 + 1 = 3$

n (column) = $U_2 - L_2 + 1 = 3$

$$\begin{bmatrix} (5,2) & (5,3) & (5,4) \\ 1 & 2 & 3 \\ (6,2) & (6,3) & (6,4) \\ 4 & 5 & 6 \\ (7,2) & (7,3) & (7,4) \\ 7 & 8 & 9 \end{bmatrix}$$

A(6,3)

= **Base address + $(i-L_1)$ column + $(j-L_2)$**

= 10+(6-5)3 + (3-2)

= 10+4 = **14**

α

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

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 7

❖Column Major

As its name implies, Column major order stores multidimensional arrays by columns

U_1 element	U_1 element	U_1 element		
			…………	
1^{st} column	2^{nd} column	3^{rd} column		U_2 column

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 8

Eg:

1	2	3
4	5	6
7	8	9

U1 element	U1 element	…………	
1 4 7	2 5 8	…………	3 6 9
1st column	2nd column		U2 column

we can find the address of A[i, j] using row major,

Base address + $(i-L_1) + (j-L_2)m$

Where, m represents row

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 9

Eg: A[6:9 , 3:6] find the address of A[8,5] , base address α=10.

$$\begin{array}{c} \\ A \end{array}\begin{array}{ccccccc} & L_1 & & U_1 & & L_2 & & U_2 & \\ [& 6 & : & 9 & , & 3 & : & 6 &] \end{array}$$

m (row) = U_1-L_1+1 = 9-6+1 = 4

n (column) = U_2-L_2+1 = 6-3+1= 4

A(8,5)

= 10+(8-6) + (5-3)x 4

= 10+2+8 = **20**

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 10

REPRESENTATION OF POLYNOMIAL USING ARRAY

➢ 1ST METHOD

- One dimensional array is defined and coefficient is added to the array and exponent is indicated by the index value

Eg : $2x^2+3x+1$

a[2]	a[1]	a[0]
2	3	1
$2x^2$	$3x^1$	$1x^0$

Here, array size is fixed. Usually it will be larger than the degree of polynomial.

➢ 2ND METHOD

- Here a one dimensional array is defined and the coefficient is added to the array and the exponent is represented by the array index.
- It differ from the first method in the size of array. Here size of the array is defined as the largest degree of the polynomial.
- Disadvantage is waste of space.

Eg: $2x^{100}+1$

100	....	0
2	....	1

waste of space

➢3RD METHOD

- Here coefficient and exponent is stored in the array. Two pointers are used to indicate the beginning and end of the polynomial.
- In this method , zero coefficient term is not used. There is no fixed size allocation is needed.

Eg: A(x) = $2x^{100}+1$ B(x)= $x^4+ 10x^3+ 3x^2+1$

	1	2	3	4	5	6	
Coef:	2	1	1	10	3	1	
Exponent:	100	0	4	3	2	0	
	af	al	bf			bl	

Free space

- Here, af and bf indicate the beginning of the polynomial. al and bl indicate the ending of the polynomial.
- To find the last term of the polynomial, we use

el = ef+(n-1)

Where,

el = last term of the polynomial

ef = first term of the polynomial

n = length of the polynomial

Eg: for B(x), el= 3+(4-1) = **6**

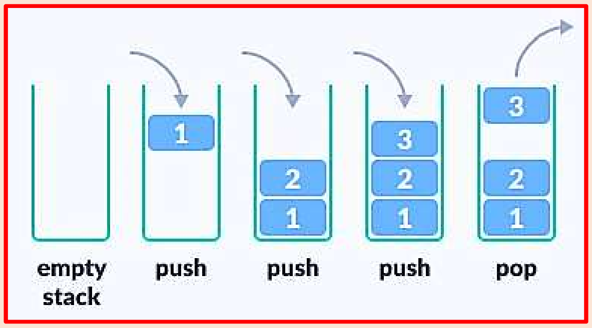
STACK

- Stack is an ordered list in which insertions and deletions are made at one end called the top.
- Insertion called push and deletion called pop.
- A stack is also known as LIFO (Last In First Out) list.

❖Stack using Array

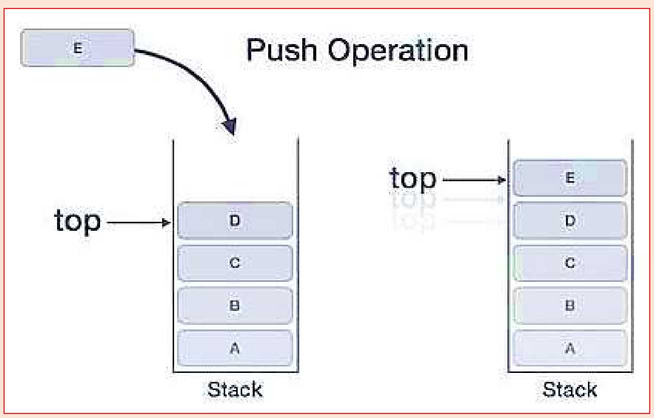
➢2 basic operations

- **PUSH**
- **POP**



➢Algorithm to add an element into a stack (PUSH)

```
Algorithm add (item)
/* add an item to the stack, top is the current top of the stack and n
is the maximum size*/
{
  if (top==n) then
        print ("stack full");
  else
        top=top+1;
        stack[top]=item;
}
```

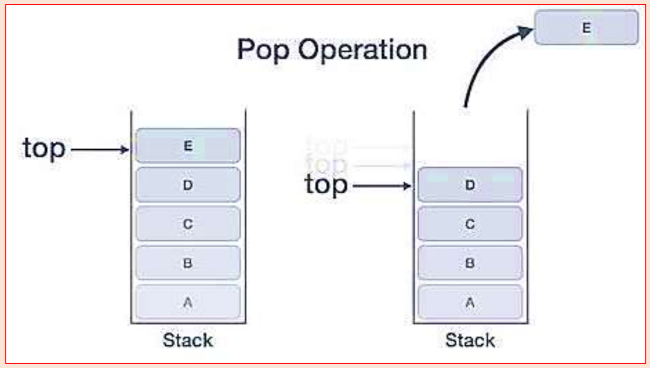


➢Algorithm to delete an element from a stack (POP)

Algorithm delete ()

/* “item” is a variable used to hold the element deleted from the top of the stack*/

```
{
  if (top==0) then
        print (“stack is empty”);
  else
        item=stack[top];
        top=top-1;
}
```



❖Application of stack

- When a function call occurs , the return address is stored in stack
- Number system conversion
- Maintaining undo list for word document application
- Sorting
- Expression evaluation
- Expression conversion
- String reversal
- Used for implementing subroutines in general programming language.

❖Expression Notations

➢Infix Notation

- It places binary operator in between its two operands

Eg: A+B, C-D, E*F

➢Prefix Notation

- Operator symbol is placed before its own operands

Eg: +AB, *EG, -CD

➢Postfix Notation

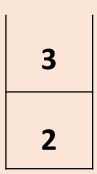
- Operator symbol is placed after its two operands

Eg: AB+, EG*, CD-

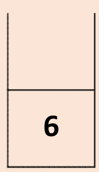
Prepared By Mr.EBIN PM, AP, IESCE EDULINE 19

❖Evaluation of Postfix Expression

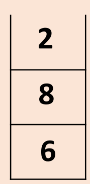
Eg: 2 3 * 8 2 / 2 * + 6 2 * - 4 + #



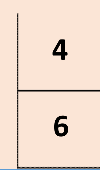
Pop 3 and 2 and perform the operation 3*2 =6 . Push 6 into the Stack.



Now push 8 and 2 in to the stack



Pop 2 & 8 and perform division operation (/) = 8/2= 4



Prepared By Mr.EBIN PM, AP, IESCE EDULINE 20

Push 2 on the top of the stack

2
4
6

Now, pop 2 & 4 , perform multiplication and push result in to the stack

8
6

Now, pop 6 & 8 , perform addition and push result in to the stack

14

Push 6 & 2 on the top of the stack

2
6
14

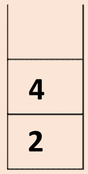
Now, pop 6 & 2 , perform multiplication and push result in to the stack

12
14

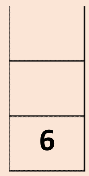
Now, pop 12 & 14 , perform subtraction and push result in to the stack

2

Push 4 on the top of the stack



Now, pop 4 & 2 , perform addition and push result in to the stack



So the answer is **6**

- The time complexity is O(n), where n is the number of tokens in the expression

❖Infix to Postfix Conversion

- 2+3*4 = (2+ (3 * 4)) = 234*+
- a*b+5 = ((a * b) + 5) = ab*5+
- (1+2)*7 = ((1 + 2) * 7) = 12+7*
- a* b/c = ((a * b) / c) = ab*c/

❖Infix to Postfix Conversion Using Stack

➢Rules

Scan the infix expression from left to right. For each symbol,

- If the symbol is an operand, output it immediately
- If the symbol is a left parenthesis, push it on the stack
- If the symbol is a right parenthesis, continually pop the stack and output the operator until the corresponding left parenthesis is popped
- Otherwise pop and output operators from the stack as long as they have a priority higher than or equal to the current operator. But never pop a left parenthesis.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 25

- At the end of the expression, pop and output operators until the stack is empty.

  + and – considered low priority

  * and / considered high priority

Eg: A*(B+C)*D

Stack	Output
(empty)	A
*	A
(*	A
(*	AB
+ (*	AB

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 26

Stack	Output
+ (*	ABC
(*	ABC+
* (* POP	ABC+*
(*	**ABC+*D***

QUEUE

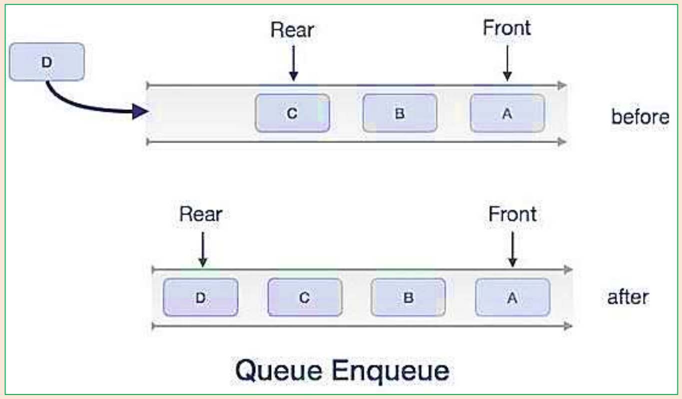
- Queue is an ordered list. The end at which new elements are added is called the rear, and that from which old elements are deleted is called front.
- Queues are also known as **First In First Out(FIFO)** lists.
- Queue can be implemented using either an array or a linked list
- Insertion called "**enqueue**" & deletion is called "**dequeue**"
- Length of the Queue = **rear- front**
- Condition for empty queue = **front==rear**
- If **rear==n**, then the queue is full

❖Array implementation

Algorithm insert (x)

/* insert x to the queue, n is the maximum size*/

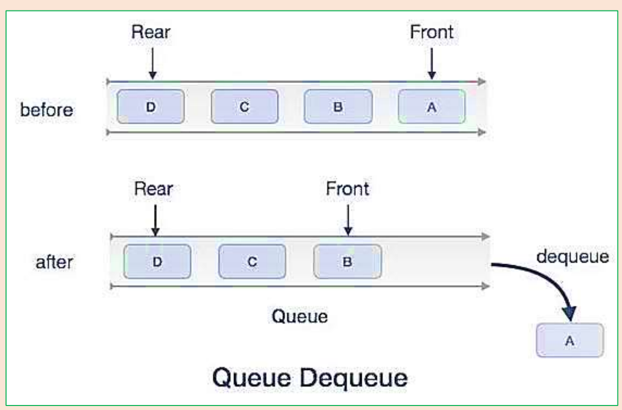
```
{
    if (rear==n) then
        print("Queue is full");
    else
        {
            rear=rear+1;
            Queue[rear]=x;
        }
}
```

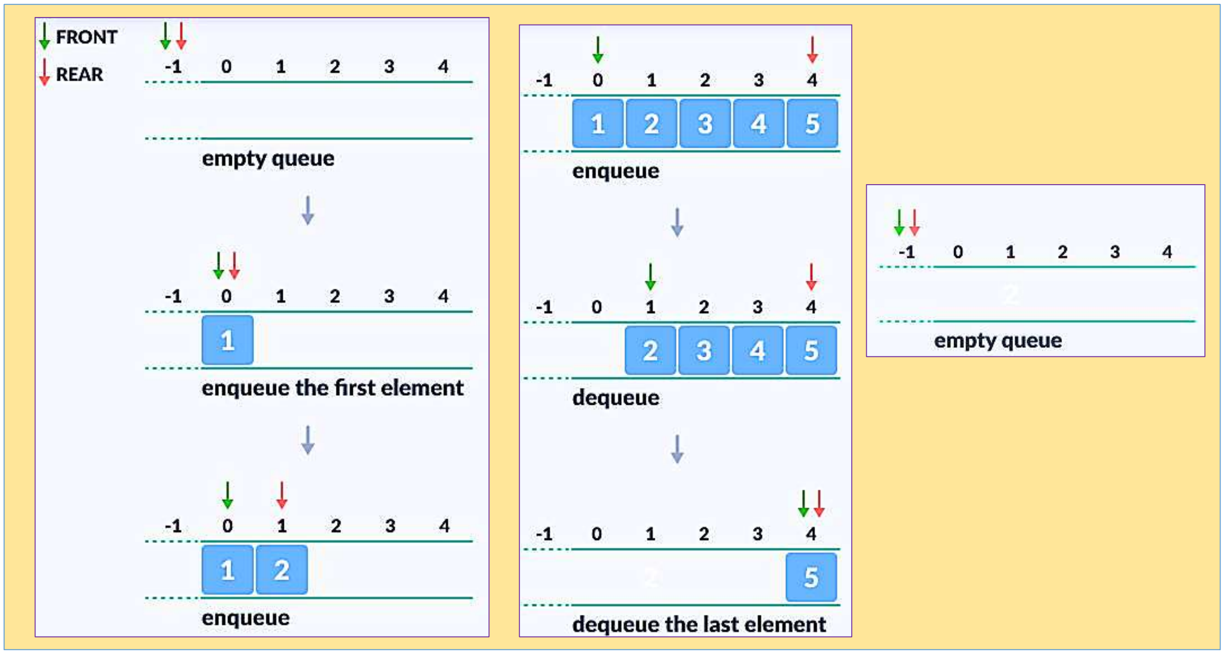


Algorithm delete ()

/* item is a variable used to hold the element deleted from queue*/

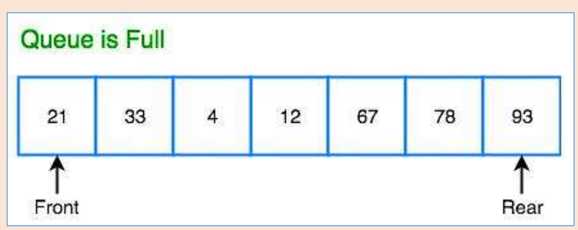
```
{
    if (front==rear) then
        print("Queue is empty");
    else
        {
            front=front+1;
            item = queue[front];
        }
}
```



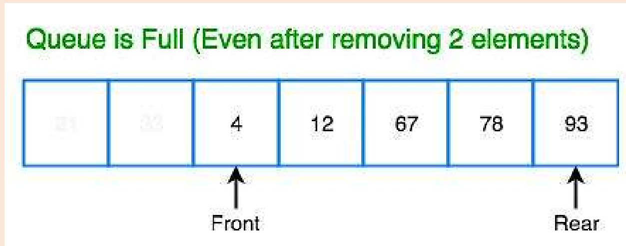


❖Representation of Queue

- In a Linear queue, once the queue is completely full, it's not possible to insert more elements. Even if we dequeue the queue to remove some of the elements, until the queue is reset, no new elements can be inserted.



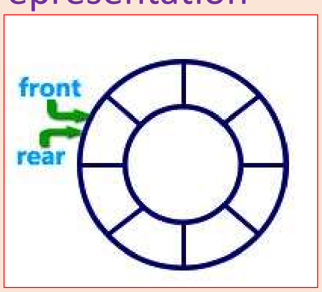
- When we dequeue any element to remove it from the queue, we are actually moving the front of the queue forward, thereby reducing the overall size of the queue. And we cannot insert new elements, because the rear pointer is still at the end of the queue.



- The only way is to reset the linear queue, for a fresh start.
- **Circular Queue** is also a linear data structure, which follows the principle of FIFO(First In First Out), but instead of ending the queue at the last position, it again starts from the first position after the last, hence making the queue behave like a circular data structure.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 33

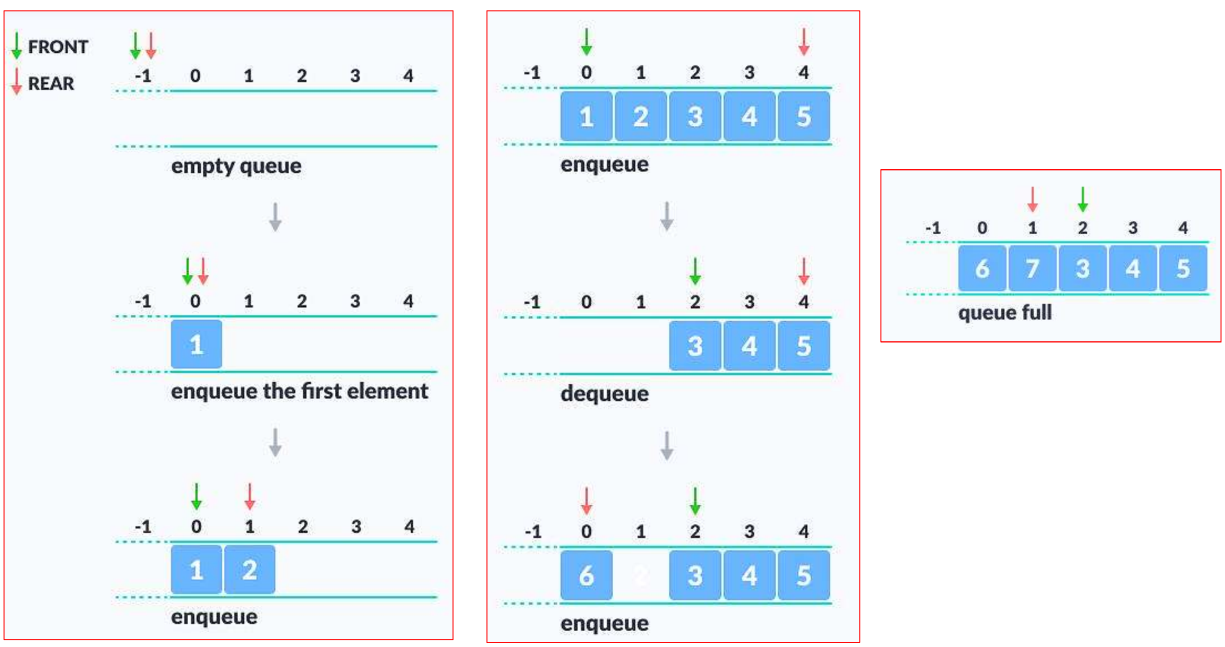
Graphical representation



- In case of a circular queue, front pointer will always point to the front of the queue, and rear pointer will always point to the end of the queue.
- Initially, the front and the rear pointers will be pointing to the same location, this would mean that the queue is empty.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 34

- New data is always added to the location pointed by the rear pointer, and once the data is added, rear pointer is incremented to point to the next available location.
- Only the front pointer is incremented by one position when dequeue is executed.
- As the queue data is only the data between front and rear, hence the data left outside is not a part of the queue anymore, hence removed.
- The front and the rear pointer will get reinitialized to 0 every time they reach the end of the queue.



❖Algorithm to insert (enqueue) an element into circular Queue

```
{
   rear=(rear+1)mod n;
   if (front==rear) then
        print("Queue is full");
   else
        queue[rear]=x;
}
```


❖Algorithm to delete (dequeue) an element from circular Queue

```
{
   if (front==rear) then
        print("Queue is empty");
   else
    {
          front=(front+1)mod n;
           item=queue[front];
     }
}
```

❖Double Ended Queue

➢ It is a linear list in which elements can be added or removed at either end; but not in the middle. There are 2 variations of a double ended queue.

Input restricted – It allows insertion at only one end of the list but allows deletions at both ends of the list

Output restricted – It allows deletion at only one end of the list but allows insertion at both ends of the list.

❖ Priority Queue

- It is a collection of elements such as each element has been assigned a priority.
- The order in which elements are deleted and processed comes from the following rules.

a) An element of higher priority is processed before any element of lower priority

b) Two elements with the same priority are processed according to the order in which they were added to the queue.

- Priority queue can be implemented in two ways

    1. Using Array 2. Using Linked List

➢Array Representation of priority queue

- Use separate queue for each level of priority (or for each priority number)

➢Linked list Representation of priority queue

a) Each node in the list will contain three items of information. An information field INFO, a priority number PRN and a link number LINK

b) A node x precedes a node y in the list

- When x has higher priority than y or
- When both have the same priority, but x was added to the list before y. This means that the order in the one way list corresponds to the order of the priority queue.

❖Application of Queue

1) In a multiuser system, task form a queue waiting to be executed one after another.
2) In computer networks, the packets that arrive from various lines are kept in a queue
3) For performing breadth First Search (BFS) of a graph, queue is used
4) Queues are generally used for ordering events on first in first out basis.

❖LINEAR SEARCH

```
Algorithm Linear Search (a, n)
/* a is the array and n is the maximum size of the array*/
  {
     k=1; loc=0;
     while (loc= =0 and k≤ n)
       {
          if (item = =a[k]) then
              loc=k;
             k =k+1;
       }
     if (loc= =0)
          print("Item is not found");
     else
         print("Item is in location - loc");
   }
```

3	2	0	1	4
a[1]	a[2]	a[3]	a[4]	a[5]

n = 5 , item = 1

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 43

Best case time complexity = $O(1)$

Worst case time complexity = $O(n)$

Average case time complexity =

$P=1/n$

$c(n) = 1 \times 1/n + 2\times 1/n + \ldots\ldots.$

$= n(n+1)/2 \times 1/n$

$= \mathbf{(n+1)/2}$

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 44

❖BINARY SEARCH

- Average TC and worst case TC can be reduced using binary search. The operation is performed on small list , and the list must be a sorted one.
- First we find the mid value of sorted list

  mid value = (L+U)/2

- Let x is the searching element. If x < mid value, we consider only the left portion of the list and we next consider that list. Then find the mid value of that list.
- Here , each time complexity can be reduced and list is contracted.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 45

```
Algorithm Binary search ()
{
    // l is the lower index and u is the upper index
    // n is the number of records
    // k is the searching element
    // k_m is the key value of mid position
    l=0;
    u=n-1;
    done = false;
while (l≤u and done=false) do
  {
     m= ⌊ l+u/2⌋;
     if (k > k_m) then
           l=m+1;
      elseif (k = = k_m) then
          {
             l=m;
             done=true;
          }
      else
          {
              u=m-1;
          }

   } // end of while
  if (not done)
  i=0;
  } // end of algorithm
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 46

- If the searching element is less than the mid value, we consider lower bound only, otherwise upper bound.
- The best case is O(1), because the searching element may be the mid value.
- The total list is reduced when divide each time. That is search space is reduced.

1st bisect = $n/2$

2nd bisect $n/2^2$ ……

i^{th} bisection, search space = $n/2^i$

Finally the search space become 1

- If $n/2^i$=1, then I = **$O(\log_2 n)$**
- This is the worst and average time complexity .
- Binary searching is efficient than linear searching

Proof

$\frac{n}{2^i} = 1$

If $\boxed{\frac{\log a}{\log b} = \log_b a}$ (Rule)

$\therefore n = 1 * 2^i$ take log each side

Then

$\log n = \log(1 * 2^i)$

$i = \log_2 n$

$\log n = \log 1 + \log 2^i$

$\log n = 0 + i \log 2$

$i = \frac{\log n}{\log 2}$

❖Algorithm for Matrix Multiplication

```
Algorithm multiplication (a,b,c,m,n,p,q)
{
// a and b are 2 matrices, c is resultant matrix
// m and n are size of matrix a
// p and q are size of matrix b
if (n≠p)
  {
    print("multiplication not possible");
    exit;
  }
else
  {
    for (i=1 to m) do
    for (j=1 to q) do
      {
        c[i][j]=0;
        for (k=1 to n) do
            c[i][j]= c[i][j]+a[i][k]*b[k][j];
      }
  } // end of else
} // end of algorithm
```

TC = O(mnq)

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 49