

$$A[N+1] = A[N]$$

$O(n)$ = time complexity

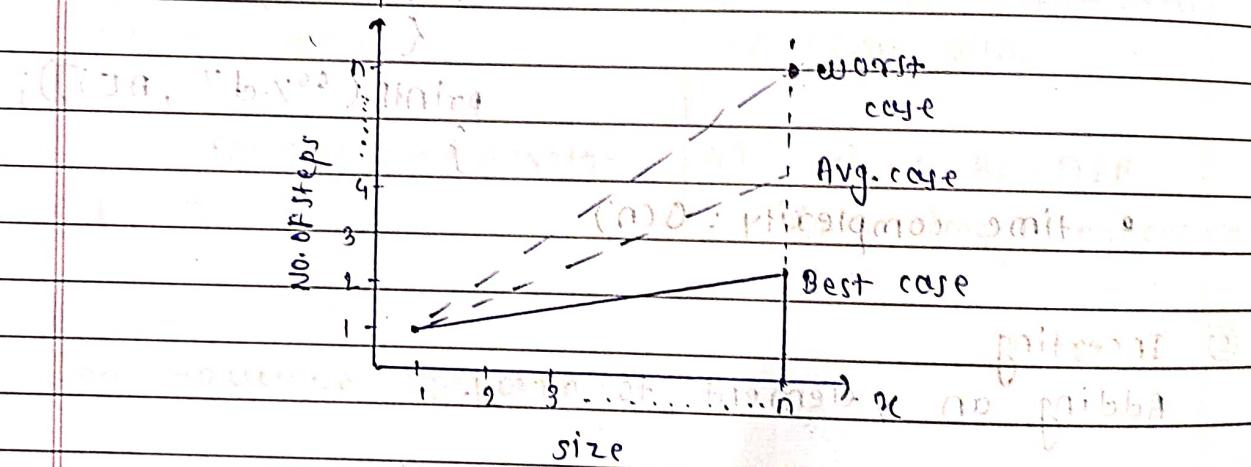
$k++;$ the situation ↑ extra space extra memory
WORST case problem

(3) Insert in between [] [] [] [] [] [] []

10	20	30	40	50	60	70	80	90
----	----	----	----	----	----	----	----	----

[Efficient = less time + less memory]

algorithm



Best case = relating to human and friend

$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < \dots < n^n$ [cases]

$O(1)$ at max to human no friend

avg case $O(n)$ relating to friend

Worst case $O(n^2)$

relating to friend

1 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90

1 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90

Algorithm of Insertion

No. of array

Insert (A, N, K, item)

↓

J ← to be inserted in array

location

(item)

Algorithm:

Step 1 : set J = N

2 : Repeat step 3 & 4 while $J \geq K$ 3 : Set $A[J+1] = A[J]$ 4 : $J = J - 1$ 5 : Set $A[K] = \text{item}$ 6 : Set $N = N + 1$

7 : exit.

for $J = 5$ $A[5] = 50$ $j = 6$ $A[6] = 50$ $j = 7$ $N = 5$, item = 11, $K = 3$ $J = 5, 4, 3, 2$ $J \geq 3$

10	20	80	40	50			→	10	20	11	80	40	50	
1	2	3	4	5	6	7		1	2	3	4	5	6	7

Address of element $\Rightarrow A[i] = b + w$ ($i = b$) $A(1, \dots, 100)$ $b = 1000$

size of element = 4 byte

size of element

final index value

 $\text{loc } A[80] = 1000 + 4(80-1)$ index value of item found

$$= 1000 + 196$$

 $= 1196$

base address

```

void insert (int a[], int);
void main()
{
    int a[10], len, ch, i;
    printf ("Enter no. of elements you want in array");
    scanf ("%d", &len);
    printf ("Enter elements in the array");
    for (i=0; i<len; i++)
    {
        scanf ("%d", &a[i]);
    }
    do
    {
        printf ("Select operation you want to perform");
        printf ("1. insertion");
        printf ("2. deletion");
        printf ("3. display");
        printf ("4. exit");
        scanf ("%d", &ch);
        switch (ch)
        {
            case 1 : Insert (a, len);
            break;
            case 2 : Delete ();
            break;
            case 3 : display ();
            break;
            case 4 : printf ("exit");
            break;
            default : printf ("Invalid choice");
            break;
        }
    } while (ch != 4);
}

```

Date _____
Page _____

```

} // insertion function
while (ch != 4); // deletion function

void insertion (int a[], int len)
{
    int i, pos, num;
    printf ("Which element you want to insert");
    scanf ("%d", &num);
    printf ("Element position");
    scanf ("%d", &pos);
    for (i = len-1; i >= pos; i--)
    {
        a[i+1] = a[i];
    }
    a[pos] = num;
    len++;
}

printf ("Array after insertion"); // To print
for (i=0; i< len; i++)
{
    printf ("%d", a[i]);
}

void delete (int a[], int len) // Deletion
{
    int pos, i;
    printf ("Enter position of an element you want to delete");
    scanf ("%d", &pos);
    for (i = pos; i < len-1; i++)
    {
        a[i] = a[i+1];
    }
    len--;
}
```

```

for(i = pos; i < len-1; i++)
    arr[i] = arr[i+1];
}
len = len-1;
printf("y. data");
for(i=0; i < len; i++)
    printf("%d", arr[i]);
}

```

ADT (Abstract Data Type)

- To define methods available for operations
- stack is ADT of push, pop, peek
- ① push()
 - add to front of stack
 - O(1)
- ② pop()
 - remove from front of stack
 - O(1)
- ③ peek()
 - return front element
 - O(1)

Time complexity

	Best case	Worst case	Avg. case
push()	O(1)	O(n)	O(1)
pop()	O(1)	O(n)	O(1)
peek()	O(1)	O(1)	O(1)

ADT :

Abstract collection of data elements and their accessing function where we are not concerned about how the accessing function will be implemented is referred as ADT.

Example : Stack, Queue

Also we are not concerned with space and time complexity at abstract level.

ADT can implement in many different ways

e.g. Array & linked list

the result of operations like insertion, deletion, search, we will get but how it processes behind that we will not conclude.

ADT Stack:

Stack is an ordered list of similar type of elements in which an element may be inserted or deleted only at one end. called as the top.

- It follows the principle LIFO or FILO

Last in First Out

e.g.: Stack of chairs; stack of plates, etc.

- push() and pull() function are used in stack.
- Top always point to the last element of the stack.

Algorithm

① push()

- Step 1: check whether stack is full or not.
- 2: otherwise increment the top by 1.
- 3: Insert the element in to bar.
- stack [top] = element

② pop()

- Step 1: check whether the stack is empty or not.
- 2: otherwise decrement the top by 1.
- 3: Delete the element from bar.

© 1996-2006

294
295

Stack :

Push()

Application of Stack:

① conversion of arithmetic expression is

$a+b \leftarrow$ operand

↑

operator

i) infix expression

ii) prefix expression (Polish Notation)

iii) postfix expression (Reverse Polish Notation)

① () Brackets

② ^ Exponential

③ * / multiplication and division

④ + - Addition & subtraction

⑤ For Infix:

operand₁, operator, operand₂

g. i)

a

+

b

g. ii)

b(x-y)

-

(x+y)

A

-

B

$\Rightarrow -AB \Rightarrow -(x-y)(x+y)$

⑥ For prefix: operator operand₁ operand₂

+ a . b

⑦ For postfix: operand₁ operand₂ operator

ab+

AB-

Xy- pq+ -

Imp

① infix to postfix

algorithm for converting infix to postfix using stack.

i) push '(' onto the stack for)

ii) scan exp left to right to end of exp and
Repeat step 3 to 6.

3) If an operand encountered p. add to post expression.

4) If left parenthesis encountered push to stack.

5) If operator encountered then

a) repeatedly pop from stack

b) add to post exp. For each operator same or
higher priority.

c) add operator to stack.

6) If right parenthesis encountered then,

a) pop from stack and add to post exp until

left parenthesis.

b) Remove left parenthesis.

7) Exit.

has highest priority

$$\text{eq: } (A \wedge B) \rightarrow (C \wedge D)$$

$\overrightarrow{pq} + \overrightarrow{qr} = \overrightarrow{pr}$ (Closure) $\overrightarrow{AB} + \overrightarrow{CD} = \overrightarrow{AD}$

$$\text{eg. 2) } (A * (B + D) / E - F * (G + H / K))$$

symbol stack post exp code

the duration of child participation and involvement.

C * C A

Page 10 of 10 by: bcG*CrashCourse, based on AB 106, 107

$C \times C +$ $\text{ABD} \oplus \text{BDB}$ (P)

• C * ~~is~~ ~~not~~ ~~an~~ ~~algorithm~~ ~~so~~ (ABDT) ~~it~~ (C)

gastroenteritis ABDT+.

C = ~~1010101010~~ ABCDEFGH

Age = 38 months ABDE/10F

$C - F$ $ABD + E/F$

$A'BD + E' / FG$

C - *C + $\text{Zn}(\text{NH}_3)_4^{2+}$ $\text{ABO}_4 + \text{E} \rightarrow \text{FG}$

C-~~ACT~~^{ATG} / ABD + E / FGH
C-~~ATC~~^{ATG} / ABD + E / FGH

C - * C + / ABD + E / * FGHK

Q. $(A+B)*C - D + E/F / (G+H)$ (Infix to postfix)

Symbol	Stack	Postfix expression
C		
A	C	A
B	C*	A B
*	C*	A B*
C	C*	A B C
-	C*-	A B C *
D	C-	A B C * D
+	C-+	A B C * D
E	C-+	A B C * D E
/	C-+/	A B C * D E
F	C-+/	A B C * D E F
/	C-+/ /	A B C * D E F
C	C-+/ / C	A B C * D E F
G	C-+/ / C	A B C * D E F G
H	C-+/ / C +	A B C * D E F G
I	C-+/ / C +	A B C * D E F G H
)	C-+/ / C +	A B C * D E F G H +

conversion of infix to prefix

Algorithm:-

① Reverse the infix expression

$$\text{eg. } (a+b)*c \rightarrow c*(b+a)$$

② Apply infix to postfix algorithm, to obtain postfix expression;

③ Reverse that postfix expression to obtain prefix expression

eg.

$$(a+b)*c \rightarrow c*(b+a)$$

Symbol Stack Postfix exp.

B	C	C
C	C*	C
*	C*	C*
C	C*	C*
b	C*	C*
a	C*	C*
)	C*	C*

$$cbat* \rightarrow *tabc$$

$$(d-c) * (b-a) \rightarrow (a-b) * (c-d)$$

$$*xy \rightarrow yx$$

$$*dc-ba \rightarrow ab-cd$$

Symbol Stack exp.

(

CC

a

CC

a

-

CC-

a-

b

CC-

ab

)

CC-

ab-

*

CC*

ab-

c

CC*

ab-

d

CC*

ab-

)

CC*

ab-

)

CC*

ab-

$$ab-cd-* \rightarrow *-dc=ba$$

stack



② Evaluation of postfix expression

Algorithm

- ① Add round bracket () (closing) at the end of the expression.
- ② Scan the expression from left to right until round bracket encountered ().
- ③ If an operand encountered push it to stack.
- ④ If an operator encountered push it to stack then
 - 1) pop top two operand from stack
 - First pop operand is denoted by op2, second pop operand is denoted by op1
 - 2) evaluate operand 2 to and operand 1.
 - 3) put that answer to stack.
- ⑤ Top of the stack will be the final answer.
- ⑥ Exit.

$$5 \ 6 \ 2 \ + \ * \ 12 \ 4 \ / \ -)$$

stack symbol stack

5

5

6

5 6

2

5 6 2

+

5 8

*

40

12

40 12

4

40 12 4

/

40 8

-

37

)

op2 op1

④ + ③ → 8

op2 op1

5 8 → 40

op2 op1

12 / 4 → 3

op2 op1

40 - 3 → 37

Handwriting		
Symbol	Stack	Written
4	4	four
5	5	five
+	+	plus
*	*	times
-	-	minus
1	1	one
2	2	two
3	3	three
4	4	four
5	5	five
6	6	six
7	7	seven
8	8	eight
9	9	nine
10	10	ten
11	11	eleven
12	12	twelve
13	13	thirteen
14	14	fourteen
15	15	fifteen
16	16	sixteen
17	17	seventeen
18	18	eighteen
19	19	nineteen
20	20	twenty
21	21	twenty-one
22	22	twenty-two
23	23	twenty-three
24	24	twenty-four
25	25	twenty-five
26	26	twenty-six
27	27	twenty-seven
28	28	twenty-eight
29	29	twenty-nine
30	30	thirty
31	31	thirty-one
32	32	thirty-two
33	33	thirty-three
34	34	thirty-four
35	35	thirty-five
36	36	thirty-six
37	37	thirty-seven
38	38	thirty-eight
39	39	thirty-nine
40	40	forty
41	41	forty-one
42	42	forty-two
43	43	forty-three
44	44	forty-four
45	45	forty-five
46	46	forty-six
47	47	forty-seven
48	48	forty-eight
49	49	forty-nine
50	50	fifty
51	51	fifty-one
52	52	fifty-two
53	53	fifty-three
54	54	fifty-four
55	55	fifty-five
56	56	fifty-six
57	57	fifty-seven
58	58	fifty-eight
59	59	fifty-nine
60	60	sixty
61	61	sixty-one
62	62	sixty-two
63	63	sixty-three
64	64	sixty-four
65	65	sixty-five
66	66	sixty-six
67	67	sixty-seven
68	68	sixty-eight
69	69	sixty-nine
70	70	seventy
71	71	seventy-one
72	72	seventy-two
73	73	seventy-three
74	74	seventy-four
75	75	seventy-five
76	76	seventy-six
77	77	seventy-seven
78	78	seventy-eight
79	79	seventy-nine
80	80	eighty
81	81	eighty-one
82	82	eighty-two
83	83	eighty-three
84	84	eighty-four
85	85	eighty-five
86	86	eighty-six
87	87	eighty-seven
88	88	eighty-eight
89	89	eighty-nine
90	90	ninety
91	91	ninety-one
92	92	ninety-two
93	93	ninety-three
94	94	ninety-four
95	95	ninety-five
96	96	ninety-six
97	97	ninety-seven
98	98	ninety-eight
99	99	ninety-nine
100	100	one hundred
101	101	one hundred one
102	102	one hundred two
103	103	one hundred three
104	104	one hundred four
105	105	one hundred five
106	106	one hundred six
107	107	one hundred seven
108	108	one hundred eight
109	109	one hundred nine
110	110	one hundred ten
111	111	one hundred eleven
112	112	one hundred twelve
113	113	one hundred thirteen
114	114	one hundred fourteen
115	115	one hundred fifteen
116	116	one hundred sixteen
117	117	one hundred seventeen
118	118	one hundred eighteen
119	119	one hundred nineteen
120	120	one hundred twenty
121	121	one hundred twenty-one
122	122	one hundred twenty-two
123	123	one hundred twenty-three
124	124	one hundred twenty-four
125	125	one hundred twenty-five
126	126	one hundred twenty-six
127	127	one hundred twenty-seven
128	128	one hundred twenty-eight
129	129	one hundred twenty-nine
130	130	one hundred thirty
131	131	one hundred thirty-one
132	132	one hundred thirty-two
133	133	one hundred thirty-three
134	134	one hundred thirty-four
135	135	one hundred thirty-five
136	136	one hundred thirty-six
137	137	one hundred thirty-seven
138	138	one hundred thirty-eight
139	139	one hundred thirty-nine
140	140	one hundred forty
141	141	one hundred forty-one
142	142	one hundred forty-two
143	143	one hundred forty-three
144	144	one hundred forty-four
145	145	one hundred forty-five
146	146	one hundred forty-six
147	147	one hundred forty-seven
148	148	one hundred forty-eight
149	149	one hundred forty-nine
150	150	one hundred fifty
151	151	one hundred fifty-one
152	152	one hundred fifty-two
153	153	one hundred fifty-three
154	154	one hundred fifty-four
155	155	one hundred fifty-five
156	156	one hundred fifty-six
157	157	one hundred fifty-seven
158	158	one hundred fifty-eight
159	159	one hundred fifty-nine
160	160	one hundred sixty
161	161	one hundred sixty-one
162	162	one hundred sixty-two
163	163	one hundred sixty-three
164	164	one hundred sixty-four
165	165	one hundred sixty-five
166	166	one hundred sixty-six
167	167	one hundred sixty-seven
168	168	one hundred sixty-eight
169	169	one hundred sixty-nine
170	170	one hundred seventy
171	171	one hundred seventy-one
172	172	one hundred seventy-two
173	173	one hundred seventy-three
174	174	one hundred seventy-four
175	175	one hundred seventy-five
176	176	one hundred seventy-six
177	177	one hundred seventy-seven
178	178	one hundred seventy-eight
179	179	one hundred seventy-nine
180	180	one hundred eighty
181	181	one hundred eighty-one
182	182	one hundred eighty-two
183	183	one hundred eighty-three
184	184	one hundred eighty-four
185	185	one hundred eighty-five
186	186	one hundred eighty-six
187	187	one hundred eighty-seven
188	188	one hundred eighty-eight
189	189	one hundred eighty-nine
190	190	one hundred ninety
191	191	one hundred ninety-one
192	192	one hundred ninety-two
193	193	one hundred ninety-three
194	194	one hundred ninety-four
195	195	one hundred ninety-five
196	196	one hundred ninety-six
197	197	one hundred ninety-seven
198	198	one hundred ninety-eight
199	199	one hundred ninety-nine
200	200	two hundred
201	201	two hundred one
202	202	two hundred two
203	203	two hundred three
204	204	two hundred four
205	205	two hundred five
206	206	two hundred six
207	207	two hundred seven
208	208	two hundred eight
209	209	two hundred nine
210	210	two hundred ten
211	211	two hundred eleven
212	212	two hundred twelve
213	213	two hundred thirteen
214	214	two hundred fourteen
215	215	two hundred fifteen
216	216	two hundred sixteen
217	217	two hundred seventeen
218	218	two hundred eighteen
219	219	two hundred nineteen
220	220	two hundred twenty
221	221	two hundred twenty-one
222	222	two hundred twenty-two
223	223	two hundred twenty-three
224	224	two hundred twenty-four
225	225	two hundred twenty-five
226	226	two hundred twenty-six
227	227	two hundred twenty-seven
228	228	two hundred twenty-eight
229	229	two hundred twenty-nine
230	230	two hundred三十
231	231	two hundred thirty-one
232	232	two hundred thirty-two
233	233	two hundred thirty-three
234	234	two hundred thirty-four
235	235	two hundred thirty-five
236	236	two hundred thirty-six
237	237	two hundred thirty-seven
238	238	two hundred thirty-eight
239	239	two hundred thirty-nine
240	240	two hundred forty
241	241	two hundred forty-one
242	242	two hundred forty-two
243	243	two hundred forty-three
244	244	two hundred forty-four
245	245	two hundred forty-five
246	246	two hundred forty-six
247	247	two hundred forty-seven
248	248	two hundred forty-eight
249	249	two hundred forty-nine
250	250	two hundred fifty
251	251	two hundred fifty-one
252	252	two hundred fifty-two
253	253	two hundred fifty-three
254	254	two hundred fifty-four
255	255	two hundred fifty-five
256	256	two hundred fifty-six
257	257	two hundred fifty-seven
258	258	two hundred fifty-eight
259	259	two hundred fifty-nine
260	260	two hundred sixty
261	261	two hundred sixty-one
262	262	two hundred sixty-two
263	263	two hundred sixty-three
264	264	two hundred sixty-four
265	265	two hundred sixty-five
266	266	two hundred sixty-six
267	267	two hundred sixty-seven
268	268	two hundred sixty-eight
269	269	two hundred sixty-nine
270	270	two hundred七十
271	271	two hundred seventy-one
272	272	two hundred seventy-two
273	273	two hundred seventy-three
274	274	two hundred seventy-four
275	275	two hundred seventy-five
276	276	two hundred seventy-six
277	277	two hundred seventy-seven
278	278	two hundred seventy-eight
279	279	two hundred seventy-nine
280	280	two hundred eighty
281	281	two hundred eighty-one
282	282	two hundred eighty-two
283	283	two hundred eighty-three
284	284	two hundred eighty-four
285	285	two hundred eighty-five
286	286	two hundred eighty-six
287	287	two hundred eighty-seven
288	288	two hundred eighty-eight
289	289	two hundred eighty-nine
290	290	two hundred ninety
291	291	two hundred ninety-one
292	292	two hundred ninety-two
293	293	two hundred ninety-three
294	294	two hundred ninety-four
295	295	two hundred ninety-five
296	296	two hundred ninety-six
297	297	two hundred ninety-seven
298	298	two hundred ninety-eight
299	299	two hundred ninety-nine

Application of array

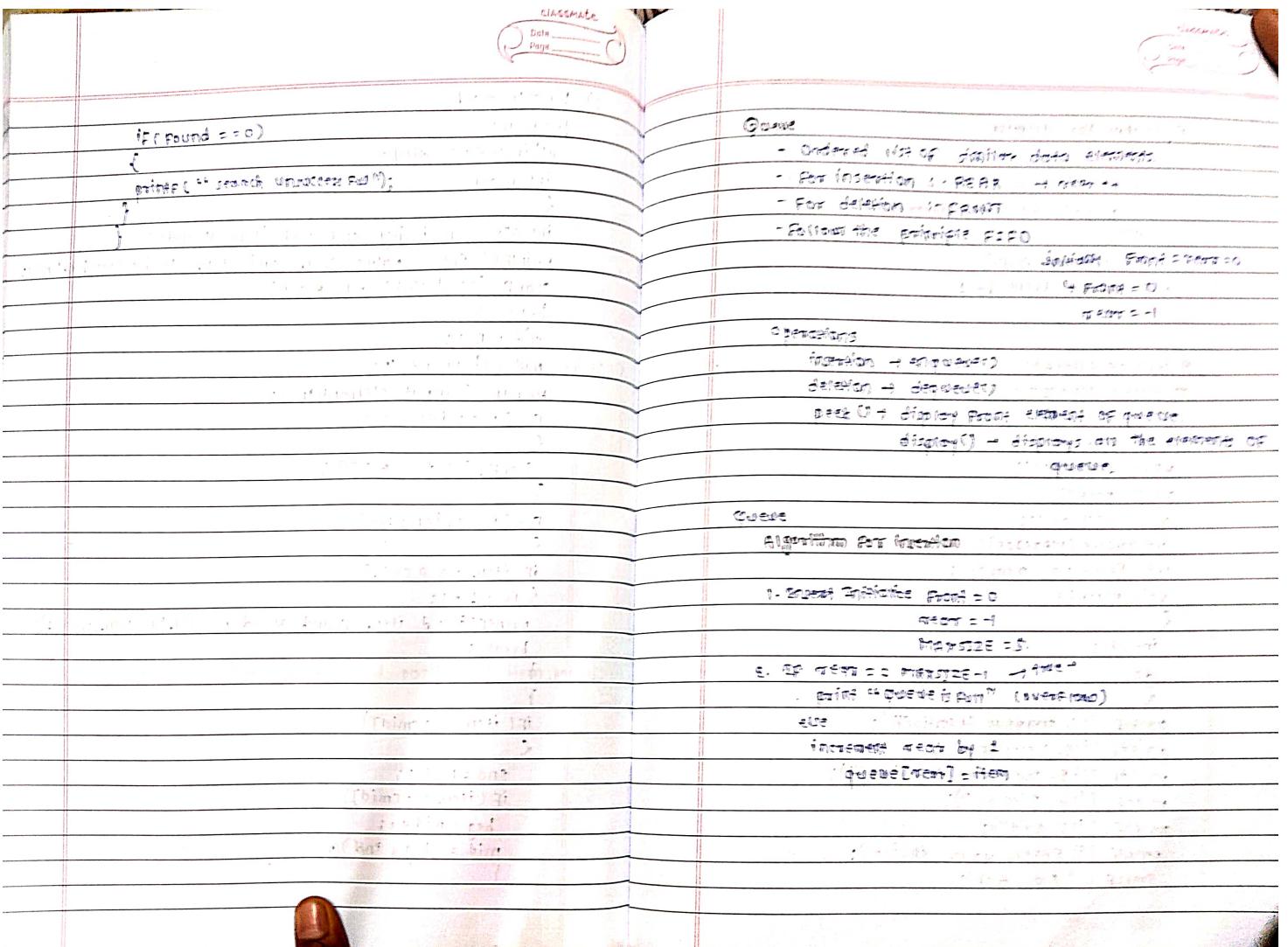
1) Linear search

```
Program:  
#include <stdio.h>  
int main()  
{  
    int a[20], n, item, pos=-1, i, found=0;  
    printf("Enter number of elements and item to be searched\n");  
    scanf("%d %d", &n, &item);  
    printf("Enter elements\n");  
    for(i=0; i<n; i++)  
    {  
        scanf("%d", &a[i]);  
    }  
    for(i=0; i<n; i++)  
    {  
        if(item == a[i])  
        {  
            pos = i;  
            printf("%d item found at %d position\n", item, pos+1);  
            found = 1;  
            break;  
        }  
    }  
    if(found == 0)  
    {  
        printf("Search unsuccessful");  
    }  
}
```

Binary search

Program:

```
#include <stdio.h>  
int main()  
{  
    int a[20], n, i, beg, end, mid, item, found=0;  
    printf("Enter value of n and item to be searched\n");  
    scanf("%d %d", &n, &item);  
    beg = 0;  
    end = n-1;  
    mid = (beg + end)/2;  
    printf("Input elements\n");  
    for(i=0; i<n; i++)  
    {  
        scanf("%d", &a[i]);  
    }  
    for(i=0; i<n; i++)  
    {  
        if(item == a[mid])  
        {  
            found = 1;  
            printf("%d item found at %d position\n", item, mid+1);  
            break;  
        }  
        if(item < a[mid])  
        {  
            end = mid-1;  
            if(item > a[mid])  
            {  
                beg = mid+1;  
                mid = (beg + end)/2;  
            }  
        }  
    }  
}
```



Algorithm for deletion

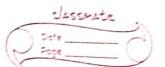
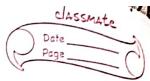
```
1. If (front == rear) then
    print "queue is empty"
else
    item = queue[front]
```

2. Increment front by 1

```
#include <stdio.h> // Standard Input Output
#define MAXSIZE 5 // Maximum size of queue
// Function prototypes
void enqueue();
void dequeue();
void peek();
void display();
int queue[MAXSIZE]; // Array to store elements
int front = 0, rear=-1;
main()
{
    // Function declarations
    printf("1. enqueue\n");
    printf("2. Deletion\n");
    printf("3. Displaying top item\n");
    printf("4. Display\n");
    printf("5. exit\n");
    printf("enter your choice");
    scanf("%d", &ch);
}
```

Switch (ch)

```
case 1 : enqueue();
break;
```



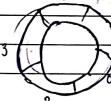
Circular Queue

In normal queue we can insert the element till queue becomes full but once queue becomes full we cannot insert the next element even if there is space in queue. Front of the queue, and thus is the drawback of simple queue (linear queue).

So wastage of memory occurs in simple queue, to utilize this memory a new concept occurs which is circular queue / ring BUFFER / circular Buffer.

In circular queue last position is connected to the first position to make the queue circular.

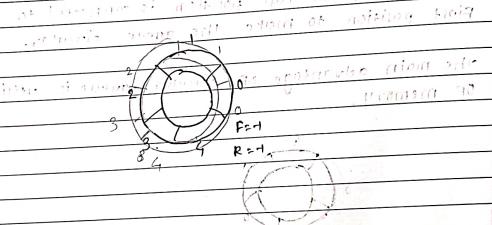
The main advantage of circular queue is utilization of memory.



Algorithm :

$N = \text{Max size}$
F = front
1) Insertion
Step 1 : If (rear+1) % N = F
 4 (R+1) % N = F
 display "overflow"
2) // If F = -1, R = 1 set F=R=0
 else
 set R = (R+1) % N

Step 3: $Q[R] = \text{item}$
 Step 4: exit. If front and rear are equal, queue is overflow
 2) Deletion
 1. If $F = R$, then queue is empty.
 display "Underflow"



complexity

Simple queue =
enqueue → O(1)
dequeue → O(n)

Time complexity = $O(1)$ enqueue → O(n)

Circular queue =
enqueue → O(1)
dequeue → O(1)
display → O(n)

Time complexity = $O(1)$ enqueue → O(1)

```

#include <stdio.h>
#define MAXSIZE 5
int queue[MAXSIZE];
int front = -1, rear = -1;
void enqueue();
void dequeue();
void peek();
void display();
int queue[MAXSIZE];
void main()
{
  int ch;
  do
  {
    printf("1. Insertion\n");
    printf("2. Deletion\n");
    printf("3. Peek\n");
    printf("4. Display\n");
    printf("5. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &ch);
    switch(ch)
    {
      case 1:
        enqueue();
        break;
      case 2:
        dequeue();
        break;
      case 3:
        peek();
        break;
      case 4:
        display();
        break;
      case 5:
        exit(0);
      default:
        printf("Wrong choice\n");
    }
  } while(ch != 5);
}
  
```

Implementation of Queue using array

Implementation of Queue using linked list

classmate
Date _____
Page _____

classmate
Date _____
Page _____

Linked list

What is linked list?

Ways to maintain a list in memory

(Two ways)

- ① **arrays DS**
- ② **linked list**

Arrays DS

11	21	41	45
----	----	----	----

↓

Arrays have some limitations and disadvantages

linked lists overcome these disadvantages

- **Type of Linked list**
- ① **single linked list**
 - Navigation is formed only.
 - we can't go backward
- ② **Doubly linked list**
 - forward and backward navigation is possible.
- ③ **circular linked list**
 - last element is linked to the first element.

Single linked list :

A single linked list is a list made up of nodes that consists of two parts :

- ① **Data**
- ② **link**

Node = Data + link
+ information

Node →

DATA INFO	LINK
-----------	------

contains the address of actual data

points to the next node of the list

* linked list is made up of node.

Representation of single list

e.g. suppose we want to store a list of numbers: 23, 54, 78, 90

→ address of second node	data	links	address	
1000 → 2000	23 2000	54 3000	78 4000	90 Null

1000 2000 3000 4000

address of data

The data in linked list need not to be in sequential order as it is not an array. It is a linked list.

Q. How to access the first node of the linked list?

With the help of pointer we can access the first node of linked list. Pointing first node to head.

1000	23 2000	54 3000	78 4000	90 Null
------	-----------	-----------	-----------	-----------

Head 1000 2000 3000 4000

→ pointers

access list node

3) Priority Queue

Implementation

① Using array

② Using multiple queue

③ Using linked list

④ Using Heap

Each element in the queue has associated priority and for deque highest priority element is get deleted.

Properties of priority queue:

- Based on First in First out approach.
- Every item has priority associated with it.
- An element with higher priority is deque before an element with lower priority.
- If two elements have the same priority they are served according to their order in the queue.

There are two types:
① Ascending priority
② Descending priority

① Ascending

(for ex) $O(n^2)$

Insertion : It can be normal or special insertion.

Deletion = Highest priority (lowest number)

↳ special deletion $O(n)$

↳ normal deletion $O(1)$.

② Descending

Elements with highest priority get removed first.

Implementation of priority

① By using Array

Priority Queue									
Implementation									
Delete Last Element									
5	1	2	3	4	6	7	8	9	10
2	1	2	3	1	2	1	2	3	4

Ascending priority

Descending priority

2	8	4	5	9	7	10	6	1	3
1	1	2	2	2	2	3	3	2	2

6	5	4	3	2	1	8	7	9	10
3	2	2	2	1	1	1	1	1	1

② By using Multiple queues

→ deleteFirst (FIFO)

P(1)	2	8	4	1	3	5	7	10	6
P(2)	5	9	7	10	6	1	3	2	4

P(2)	5	9	7	10	6	1	3	2	4
P(3)	6	1	3	2	4	5	7	9	10

P(3)	6	1	3	2	4	5	7	9	10
P(4)	1	3	2	4	5	7	9	10	6

Linked list

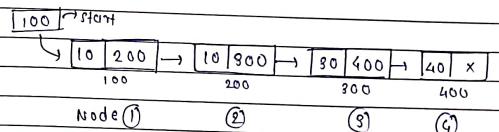
It is also known as one way list. It is linear collection of data element called node, where linear order is given by pointer.

Each node is divided into two parts:

- ① Info
- ② Link

Info	Link
------	------

Info : Information / data of element
link : Address of next node.



Node ① ② ③ ④

Advantages:

- 1) Dynamic size
- 2) Ease of insertion and deletion.

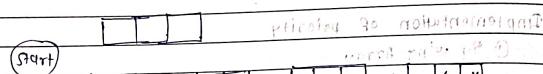
Disadvantage:

- 1) Random access is not allowed.
- 2) Extra memory used at every node.

③ By using linked list

Every node is get divided into three parts

- ① Information
- ② Priority
- ③ Address of next node



→ Deletion occurs from the beginning only.

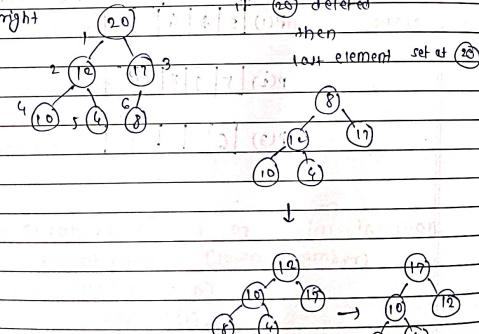
④ By using Heap

Information is stored in heap.

① Max Heap

Information in parent heap is greater than its children heap.

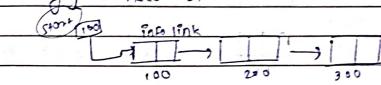
Left child & right child if 20 deleted then last element set at 20



Types of Linked list

- ① Singly (one way list)
- ② doubly
- ③ circular

① Singly linked list



```
struct node {  
    int info;  
    struct node *next;  
};
```

Operations of singly linked list

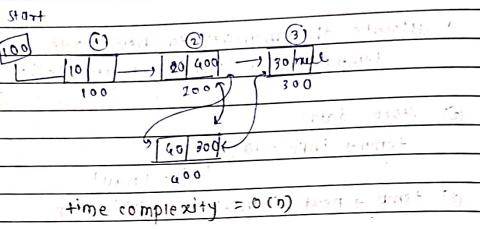
Ex: ① Traversing through list to search

Algorithm:

- 1) set temp = start // struct node *temp;
- 2) Repeat the step ③ and ④ until temp != null
(temp != null) // pointer variable
- 3) Display temp->info; // provides address of node
- 4) temp = temp->next; // move to next node

For empty condition, (temp == null)

time complexity = O(n)



time complexity = $O(n)$

Code (Insertion)

```

void insert()
{
    struct node *temp;
    int info; int pos, i;
    temp = malloc(sizeof(struct node));
    temp->next = NULL;
    newnode = malloc(sizeof(struct node));
    printf("Enter the element you want to insert");
    scanf("%d", &temp->info);
    if (start == NULL)
    {
        start = temp;
    }
    else
    {
        temp->next = start;
        start = temp;
    }
    printf("At which position you want to insert:");
    scanf("%d", &pos);
    struct node
    {
        for(i=2; i<pos; i++)
        {
            if (temp->next == NULL)
            {
                to insert element if (temp->next == NULL)
                at 1st position
            }
        }
    }
}

```

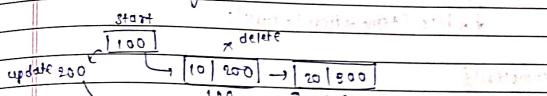
$temp = temp \rightarrow next;$

$newnode \rightarrow next = temp \rightarrow next;$
 $temp \rightarrow next = newnode;$

② Deletion:

complexity $O(1)$

① From begin



Algo-

① point start start \rightarrow next

② Delete from end complexity $O(n)$

Algo.

① traverse the list to the second last node
② change its next point into null.

③ Delete from middle (specified position)

complexity $O(n)$
 $p \rightarrow next = temp \rightarrow next$

Struct node *p, *temp;
while (temp != NULL)

P temp

temp = start
p->temp = temp

```

Searching do {
    if (temp->info == info)
        {
            printf("Item Found!\n");
            break;
        }
    else
        temp = temp->next;
} while (temp != null);

```

complexity

- ① Best case $\rightarrow O(1)$
- ④ Worst case $\rightarrow O(n)$

© 2013 Pearson Education, Inc. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

10.000
10.000
10.000
10.000

Scanning: Indigenous collection staff; editing: RG

卷之三

1937-18-26

Churn Prediction

卷之三

卷之三

《詩經》卷之三

Digitized by srujanika@gmail.com