```
struct student
{
  int rollno;
  char name;
  int phoneno;
};
Student s1;
```

assigning value to the rollno in the variable s1 →

```
s1.rollno = 10;
s1.name = 'A';
s1.phoneno = 123456;
```

dot operator

Array of Structure

Name	Rollno	Phoneno	Name	Rollno	Phoneno

1+2+4=9 bytes

S_1 S_2

we apply for for array of structure.
Here structure will become a data type for array.

```
student s[60];
```

Linked list

It is a collection of dissimilar ~~data types or~~ structure.
→ Dynamic memory location. (malloc function)

→ It is a collection of nodes.

node

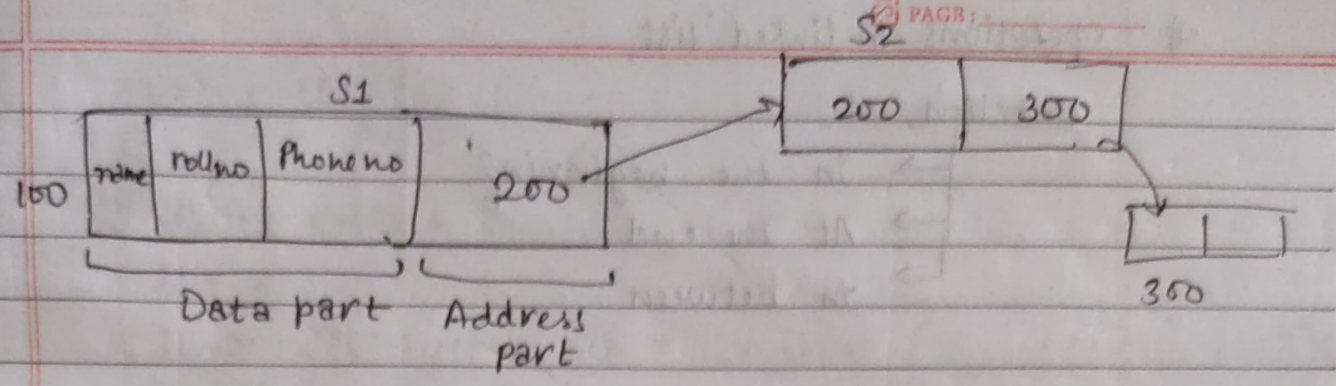
data part	Address part

Address of next student should be stored here.

Address part or linked part

data part

name	rollno	phone no	



Malloc

↳ It is a memory allocator function.

alloc.h ⇒ header file

```
int *x ;      (* represent pointer)
x = (int*) malloc (size of (int));
```

malloc returns an static address.

malloc always returns address of generic data type.

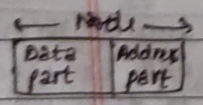
↳ data type ⇒ void *

⇒ $\boxed{x = \text{malloc (size of operator)}}$

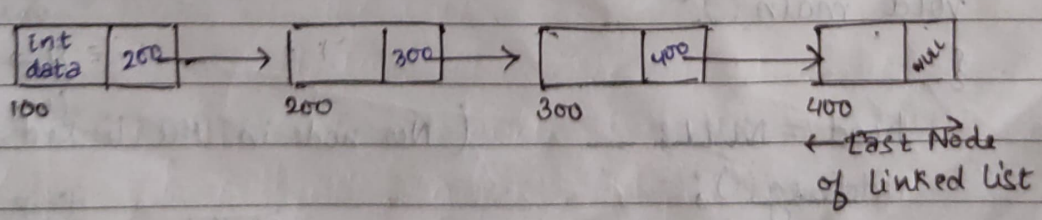
⇒ $\boxed{\text{char *}}$ x ; (data type of x)

```
x = (char *) malloc (size of (char));
```

Monday
11/09/2023



Linked list (collection of Nodes)




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

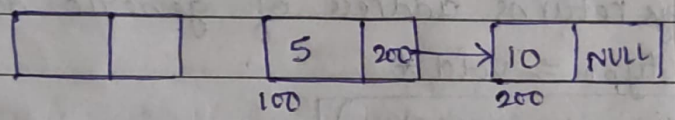
Operations in linked list

Insertion
- → In the beginning
- → At the end
- → In between

Insertion in the beginning

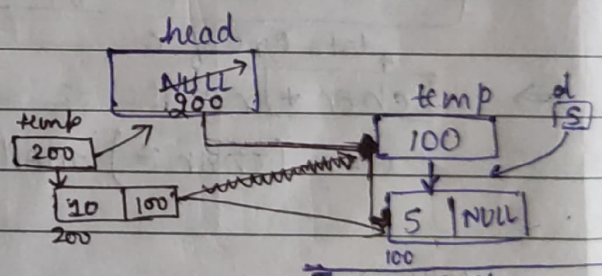
syntax of node structure :-

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



d = local variable

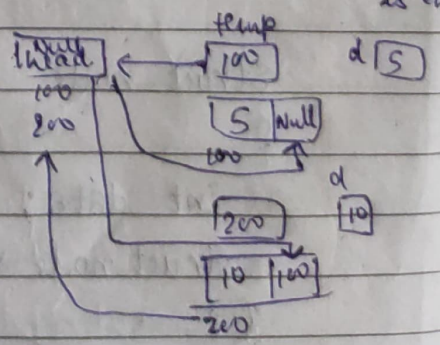
```
struct node
{
  int data;
  struct node * next;
} * head;
// struct node * head;
void main()
{
    head = NULL;
    addbeg();
    addbeg();
```



any of them can be used to define variable

// → comment

(NO node in the linked list i.e L.L. is empty)



In the beginning → head point temp which contains address of 1st node, 2nd node is added but in the beginning i.e. 2nd node created now is the 1st node.

DATE: __/__/____
PAGE: _____

```
Void addbeg( )
{
  Struct node * temp;
  int d;
  Temp = (struct node *) malloc (size of (struct node));
  if ( Temp == NULL)
      printf ("Not enough space");
  else
    {
      printf ("Enter the value for data part: ");
      scanf ("%d", &d);
      temp -> data = d;
      temp -> next = head;
      head = temp;
    }
}
```

Tuesday
12-09-2023

Today's Topic
→ Insertion at the end
→ Display

```
Struct node
{
  int data;
  Struct node * next;
} * head;

void main( )
{
    head = NULL
    addbeg ( );
    add beg( );
    display ( );
    addend( );
}
```

head should always contain address of 1st node
So a local variable is create which take
first address & start printing the data
of Data Part of nodes

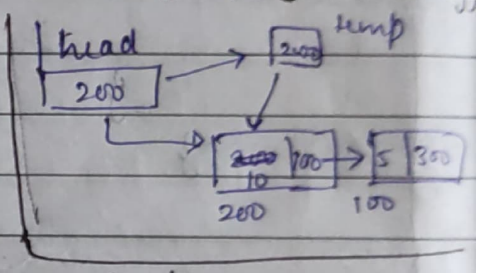
```
void addbeg()
{
   ≡
}
```

} \\previous code of insertion in beginning

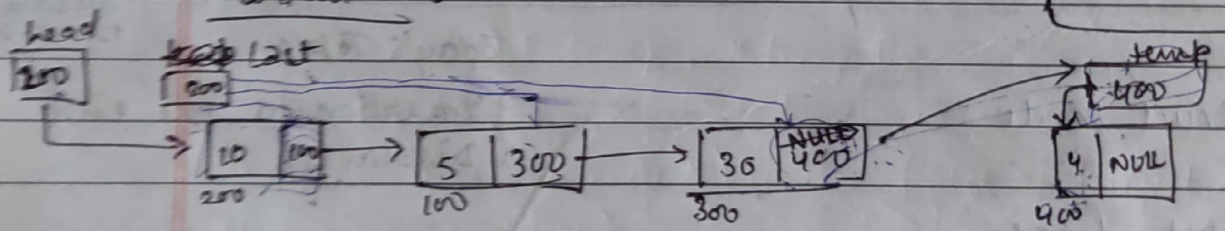
```
void display()
{
  struct node * temp;
   temp = head;
  while (temp != NULL)
  {
     printf ("%d", temp -> data);
     temp = temp -> next;
  }
}
```

data part of temp
next part of temp

(display) of insertion at begg



Insertion at the end of the linked list




```
void addend( )
{
  int d;
  struct node* temp; * last;
  temp = (struct node *) malloc (size of (operator));
  if (temp == NULL)
        printf ("Not enough space");
  else
     { printf ("Enter the value of data part:");
       scanf ("%d", &d);
       temp -> data = d;
```

Good Write

~~temp → data~~

```
temp → next = NULL;
if (head == NULL)
        head = temp;
else
    {
        last = head;
        while ( last → next != NULL )
            {   last = last → next;
            }
        last → next = temp;
    }
  }
}
```

18/09/2023

* Insertion in between

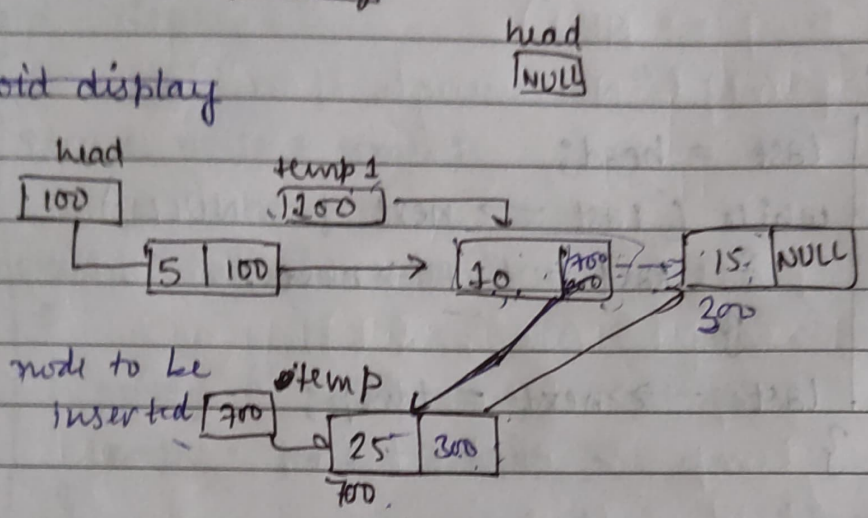
```
struct node
 {
  int data;
  struct data * next;
 } * head;

void main ()
{
  head = NULL;
  addbeg();
  addend();
  display();  addbetween();
              display();
}

void addbeg()
{
}
```

Good Write

```
void addend()
{
}
void display()
{
}
```

Insertion in beginning

~~void display~~




```
void addbetween()
{
struct node * temp, * temp1;
int d, d1;
temp = (struct node *) malloc (size of (struct node));
if (temp == NULL)
    print ("Not enough memory");
else
    {
        print ("Enter the value of data part:");
        scanf ("%d", &d);

temp -> data = d;
printf ("Enter the value after which node to be inserted:")
Scanf ("%d", d1);

temp1 = head;
```

DATE: __/__/____
PAGE: _____

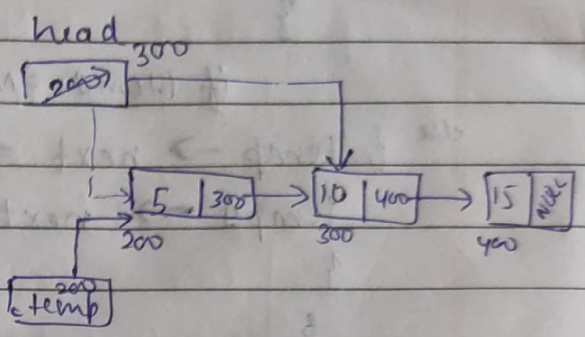
```
while (temp1 -> data != d1)
    {
        temp1 = temp1 -> next;
    }
        if (temp1 == NULL)
                printf (" The Data part after which node new node to be inserted is not present");
else { temp -> next = temp1 -> next;
    temp1 -> next = temp;
    }
    }
}
```

Wednesday
20/9/2023

```
struct node
{
    int data;
    struct node * next;
} * head;

void main()
{
    head = NULL;
    addbeg();
    addend();
    addin between();
    display();
    delbeg();
    display();
}
void addbeg()
{ --- }
void addend()
{ --- }
```

Good Write

```
void addinbetween()
{ - - }

void display()
{ - - }
```

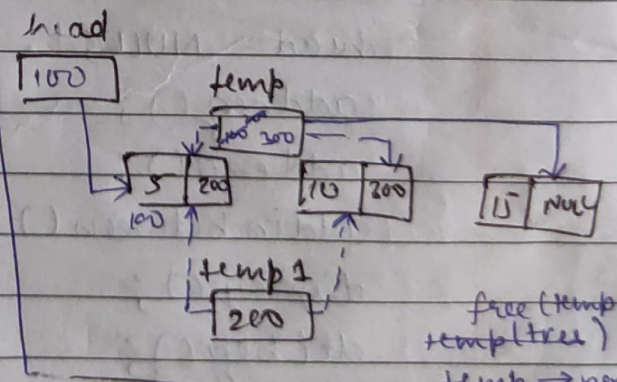


Deletion in beginning

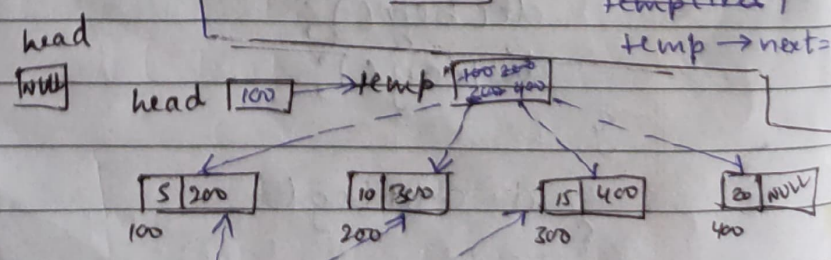
```
void delbeg()
{
    Struct node *temp;
    if (head == NULL)
        printf("linked list does not exist.");

    else
        {
            temp = head;
            head = head -> next;
            free (temp);
        }
}
```

21/09/2023

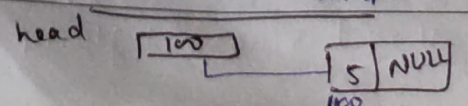
```
struct node
{
  int data;
  struct node * next; }* head;

void main() {
head = NULL;
addbeg();
addend();
add_in_between();
add beg();
display();
del beg();
display();
delend();
display();
getch();
}
```



free (temp)
temp(free)
temp -> next = NULL



temp1 -> next = NULL;

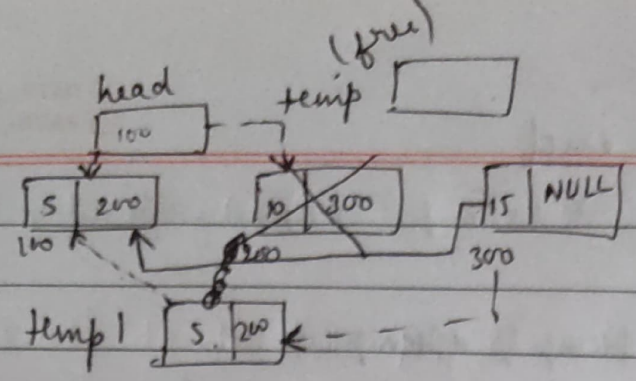


Good Write

Thursday
DATE: 21/09/2023
PAGE:

Deletion in end

```
void delend()
{
    struct node * temp , * temp1 ;
    if ( head == NULL)
        printf (" Linked list does not exist.");

    else
        {
            if (head -> next == NULL)
                {
                    temp = head;
                    free (temp);
                    head = NULL;
                }


            else
                {
                    temp = head ;
                    while ( temp -> next != NULL)
                        {
                            temp1 = temp ;
                            temp = temp -> next ;
                        }
                    temp 1 -> next = NULL ;
                    free(temp);
                }
        }
}
```



Deletion in between

```
void delin between()
{
    int d;
    struct node * temp, * temp1;
    if (head==NULL)
            printf ("Linked list does not exist");

    else
        {
            printf ("Enter the data value of the node to
                    be deleted : ");
            scanf ("%d", &d);
            if ( head -> data == d)
                {
                    temp = head;
                    head = head -> next;
                    free (temp);
                }

            else
                {
                    temp = head;
                    while ( temp -> data != d)
                        {
                            temp1 = temp;
                            temp = temp -> next;
                        }
                    if (temp==NULL)
                            printf ("Required node does not exist.");
```

Good Write

— Node —

DATE: __/__/____
PAGE: _____

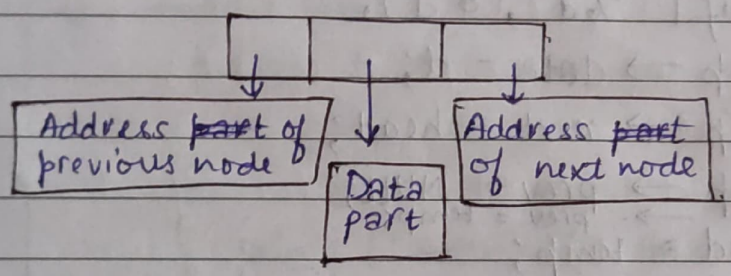
```
else
{
    temp1 → next = temp → next;
    free (temp);
}
}
}
```

Monday
25/09/2023

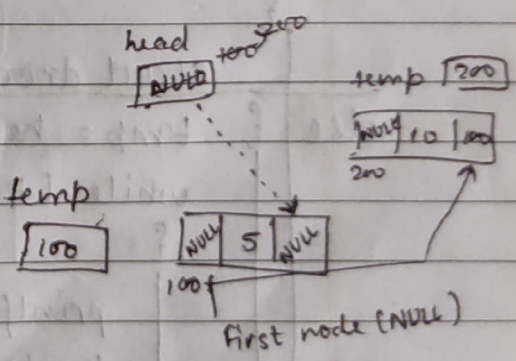
Today's topic : Doubly linked list

Insertion in beg

Node



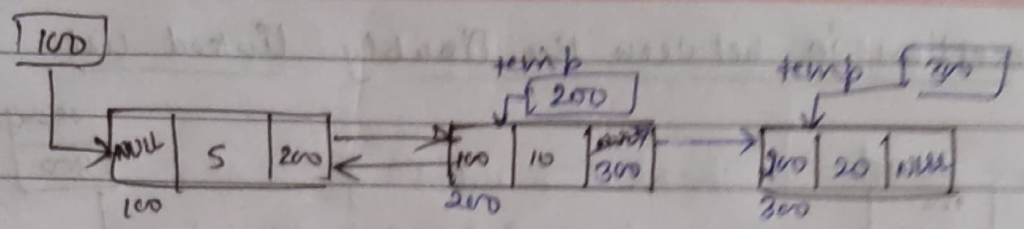

```
struct dnode
{
    int data;
    struct dnode * next;
    struct dnode * prev;
} * head;

void main()
{
    head = NULL;
    daddbeg();
    dadd beg();
}
```



DATE: ___/___/___
PAGE: _____

```
void daddbeg()
{
    struct dnode * temp;
    int d;
    temp= (structdnode *) malloc (sizeof (struct dnode));
    if (temp == NULL)
        printf(" Not enough space.");
    else
        {
            printf(" Enter the value of data part.");
            scanf("%d", &d);
            temp -> data = d;
            temp -> next = head;
            temp -> prev = NULL;
            head -> prev = temp;
            head = temp;
        }
}
```

```
void display()
{
    struct dnode * temp;
    if (head == NULL)
        printf ("Linked list does not exist
else {
    temp = head;
    while (temp != NULL)
    {
        printf("%d", temp -> data);
        temp = temp -> next;
    }
}
}
```

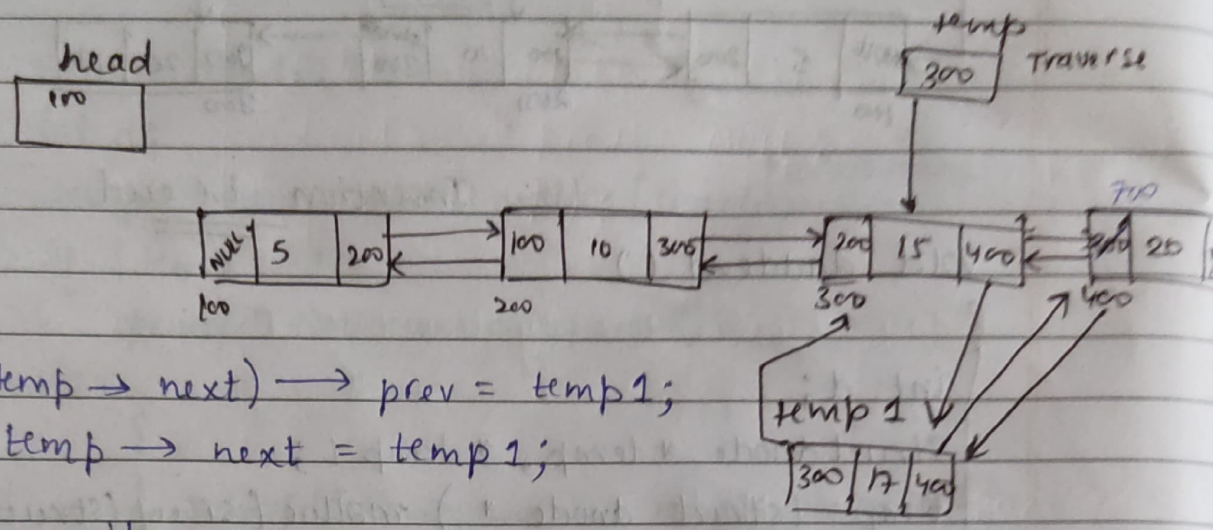


Insertion in end

```
void daddend ()
{
    int d;
    Struct dnode *temp, *temp1;
    temp = (struct dnode *) malloc (sizeof (Struct dnode));
    if (temp = NULL)
            printf (" Not enough space.");
    else
        {
            printf ("Enter the data part: ");
            scanf ("%d", &d);
            temp -> data = d;
            temp -> next = NULL;
            temp1 = head
            if (head == NULL)
            {
                temp -> prev = NULL;
                head = temp;
            }
            else
                {
                    temp1 = head;
                    while (temp -> next != NULL)
                        temp1 = temp1 -> next;
                    temp1 -> next = temp;
                    temp -> prev = temp1;
                }
        }
}
```

Good Writ

Insertion in between the Doubly linked list

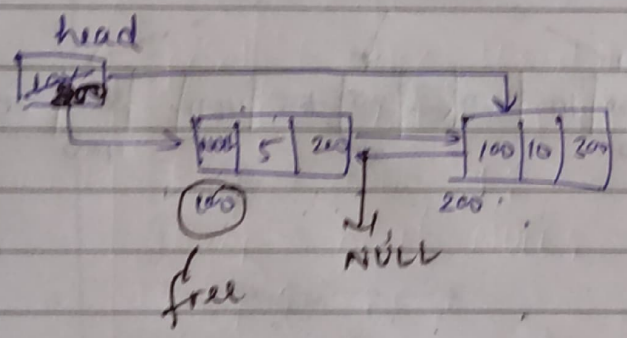


(temp → next) → prev = temp1;
temp → next = temp1;

```
void
~~struct~~ dadd between ()
{ struct dnode * temp, * temp1;
  int d,d1;
  temp1 = (struct dnode *) malloc (sizeof(struct dnode));
  if (temp1 == NULL)
        printf (" Not enough memory. ");
  else
    {
        printf (" Enter the data part: ");   to be ~~searched~~ & value of data part to be sorted
        scanf (" %d", &d, &d1);
        temp1 → data = d;
        if (head = NULL)                     or linked list does not exist
        {
            temp1 → next = NULL
            temp1 → prev = null
        }
        else
        {
            while (temp → data != d1)
                temp = temp → next
                if (temp == NULL)
                    printf (" Node does not exist
```

```
else
    {
        temp1 -> next = temp -> next;
        temp1 -> prev = temp;

        (temp -> next) -> prev = temp1;
        temp -> next = temp1;
    }
  }
}
```

Deletion in beginning

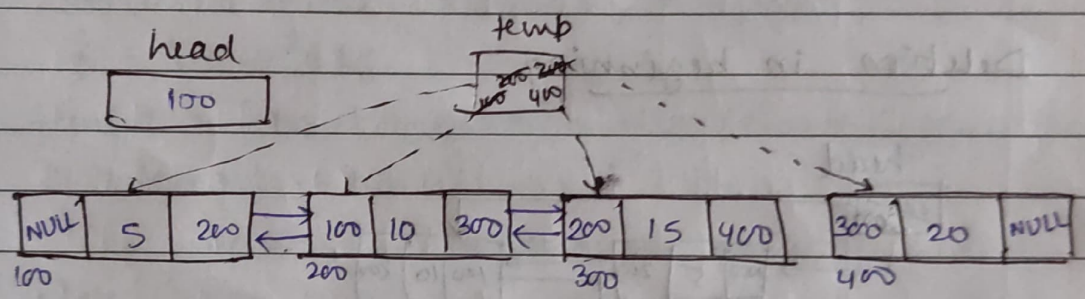



```
void ddelbeg()
{
    struct dnode * temp;
    ~~int d~~;
    if (~~temp~~ head == NULL)
        printf("Doubly link list does not exist.");
    else
        {
            temp = head;
            if (head -> next == NULL)
                {
                    head = NULL
                    free (temp);
                }
```

```
else
{
    head = head -> next;
    head -> prev = NULL;
    free (temp);
}
}
}
```

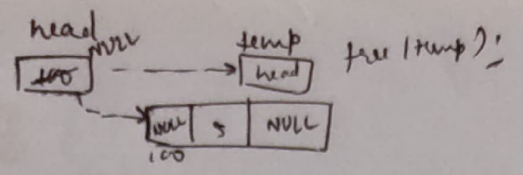
03-10-2023
Tuesday

Doubly linked list Deletion at the end




```
struct dnode
{
    int data;
    struct ~~data~~ dnode * prev;
    struct dnode * next;
} * head;

void main()
{
    head = NULL;
    daddbeg();
    daddend();
    daddinbetween();
    ddelbeg();
```

Good Write



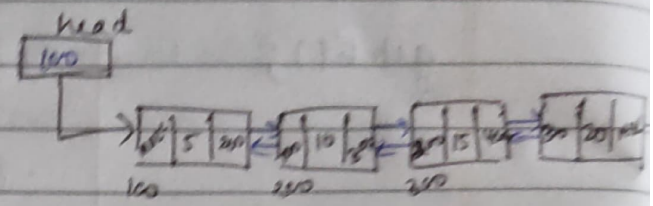

```
ddelend();

getch();
}

void ddelend()
{
  Struct dnode * temp;
   if (head == NULL)
       printf (" Doubly linked list does not exists.");
   else
       {
         temp = head;
         if (head -> next == NULL)
          {
            head = NULL;
            free (temp);
          }
         else
             {
               while (temp -> next != NULL)
                {
                  temp = temp -> next;
                }
               (temp -> prev) -> next = NULL;
               free (temp);
             }
       }
}
```

Thursday
DATE: 5/10/2023
PAGE:

4 Doubly linked list :- Deletion in between

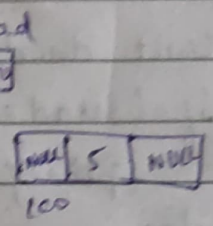
```
Struct dnode
{
  int data;
  struct dnode * prev;
  Struct dnode * next;
} * head;

void main()
{
  head = NULL;
  ~~dnode~~daddbeg();
  dnddend();
  daddinbetween();
  ddelbeg();
  ddelinbetween();

  getch();
}

void ddelinbetween()
{
  struct dnode * temp;
  int d;
  if (head == NULL)
     printf("Doubly linked list does not exist.");

  else
     {
        printf("Enter the value of data part for the node to
                be deleted : ");
```

(i)



(ii) ~~[illegible]~~ head ⇒ ~~head~~ D. linked list does not exist

(iii)



```
scanf("%d", &d);
temp = head;
while (temp -> data != d)
        temp = temp -> next;
        if (temp == NULL)
            print("Node does not exist.");
        else
        {
            if (temp == head)
            {
                (temp -> next) -> prev = NULL;
                head = head->next;
                free(temp);
            }

            else
            {
                (temp -> next) -> prev = temp -> prev;
                (temp -> prev) -> next = temp -> next;
                free(temp);
            }
        }
}
}
```

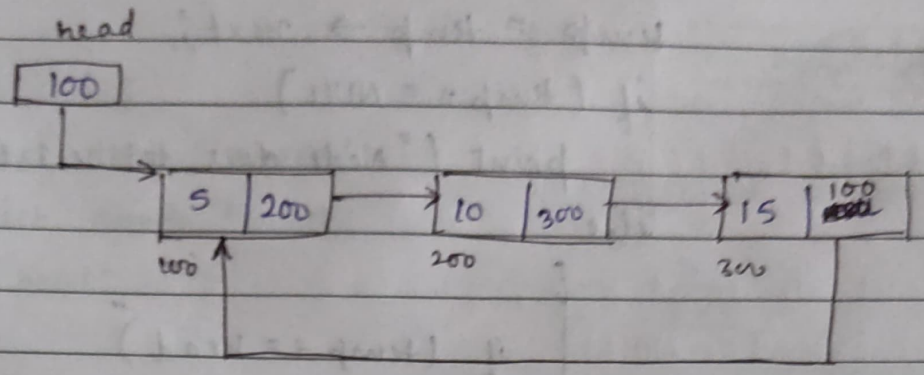
Node does not exist

// 1st node ⇒

Good Write

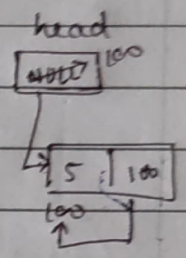
Circular linked list

(ii)




```
struct cnode
{
  int data;
  struct cnode * next;
} * head;
```

(i)




```
void main()
{
  head = NULL;
  caddbeg();
  display();
 getch();
}
```

head → global struct pointer variable

```
void caddbeg()
{
  struct cnode * temp, * temp1;
    int d;

  temp= (struct cnode *) malloc (sizeof (struct cnode));
  if (temp == NULL)
        printf (" Not enough memory.");
```

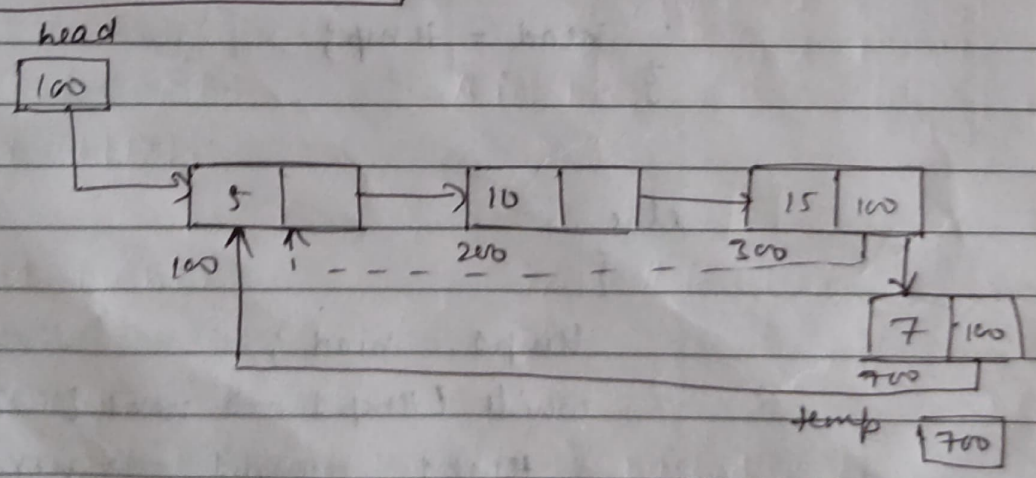
DATE: __/__/____
PAGE: _____

```
else
{
    printf(" Enter the data part:");
    scanf("%d", &d);
    temp -> data = d;
if (head == NULL)
{ temp -> next = temp;
    head = temp;
}
```

```
else
{
    printf(" Enter the data part: ");
    scanf("%d", &d);
    temp -> data = d;

    if (head == NULL)
    {
        temp -> next = temp;
        head = temp;
    }

    else
    {
        temp1 = head;
        while (temp1 -> next != head)
            temp1 = temp1 -> next;
        temp1 -> next = temp;
        temp -> next = head;
        head = temp;
    }
}
}
```

```
void display ()
{
    struct cnode * temp;
    if (head == NULL)
        printf ("circular linked list does not exist.");

    else
    {
        temp = head;
        while (temp -> next != head)
        {
            printf ("%d", temp -> data);
            temp = temp -> next;
        }
        printf ("%d", temp -> data);
    }
}
```

Insertion at the end




```
void caddend ()
{
    struct cnode * temp, * last;
    int d;
temp = (struct cnode *) malloc(sizeof(struct cnode));
```

Good Write

```
if (temp == NULL)
    printf (" Not enough space");

else
    {
        printf (" Enter the value of data part:");
        scanf (" %d", &d);

        temp -> data = d;
        temp -> next = head;
        last = head;

        if (head == NULL)
            {
                temp -> next = temp;
                head = temp;
            }

        else
            {
                while (last -> next != head)
                    last = last -> next;
                last -> next = temp;
            }
    }
}
```

Good Write

$\boxed{a+b}$ ⇒ Infix operation → operands end + operator in between

Postfix = ab+

Prefix = +ab

Monday

DATE: 9 / 10 / 2023

PAGE:

5 Marks → Conversion of Infix to Pre/Postfix

Stack

↳ linear and user defined function

LIFO → last In First Out

global variable ← $\boxed{tos}$ → top of stack

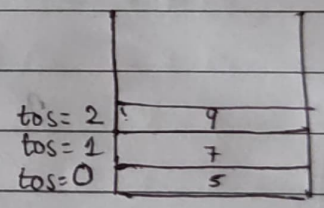
1) Arrays 2) Linked list

Insertion in a stack is called Push operation.
Deletion from the stack is called Pop operation

tos = $\underbrace{-1}_{\text{In case of array}}$ / $\underbrace{NULL}_{\text{In linked list}}$

overflow → No empty space / stack is full

underflow → Stack is empty.



tos++;

tos = -1

push(5)
push(7)
push(9)

pop()
pop()

st[tos] = 5;

st[10]

10 elements

tos--;

Arrays
Insertion

we can give tos = -1 in main function

```
int tos = -1;          // int tos;
int st[10];

void push()            // Insertion
{
    int d;
    printf("Enter the value to be pushed: ");
    scanf("%d", &d);

    if (tos == 9)
        printf("Stack overflow.");
```

Good Write

```
else
    {
        tos++;                          // tos = tos + 1;
        st[tos] = d;
    }
}

void pop()                              // Deletion
{
    int d;

    if (tos == -1)
        printf("Stack underflow");

    else
        {
            d = st[tos];
            tos--;
            printf("Deleted value is %d", d);
        }
}

void display()
{
    int i;
    if (tos == -1)
        printf("Stack does not exist");

    else
        {
            for (i=0; i<= tos; i++)
            printf("%d", st[i]);
        }
}
```

Good Write

```
void main()
{
    push();
    push();
    push();
    pop();
    display();
    getch();
}
```

Assignment : Insertion by linked list in stacks

# Infix = A+B
Prefix = +AB
Postfix = AB+

Infix to Prefix Conversion

5 Marks

Imp

(A + (B+C) - D/E)

Start scanning from right

Character Scanned	Stack (operator)	Prefix Expression (opperands)
)	)	
E	)	E
/	)/	E
D	)/	DE
-	)-	/DE
)	)-)	/DE
C	)-)	C/DE
+	)-)+	C/DE
B	)-)+	BC/DE
C	)-	+BC/DE
*	)-*	+BC/DE

/>E

A	) - *	A + BC / DE
(	empty	$\underbrace{-*A+BC/DE}_{\text{Result}}$

Q (4 $2 * 3 - 3 + 8 / 4 / (1+1))

to the power

$, ^ → highest precidence

/ *

+ -

Character Scanned	Stack	Prefix Expression
)	)	
)	))	
1	))	1
+	))+	1
1	)+	11
(	)	+11
/	)/	+11
4	)/	*4+11
/	)//	4+11
8	)//	84+11
+	)+	//84+11
3	)+	3//84+11
−	)+−	3//84+11
3	)+−	33//84+11
*	)+−*	33//84+11
2	)+−*	233//84+11
$	)+−*$	233//84+11
4	)+−*$	4233//84+11
(	empty	$\boxed{+-*\$4233//84+11}$

Result

Wednesday
DATE: 11/10/2022
PAGE:

Imp

Infix to Post-fix Conversion

Q (A * (B+C) - D/E)

Start scanning from left to right

Character Scanned	Stack	Postfix Expression
(	(	-
A	(	A
*	(*	A
(	(*(	A
B	(*(	AB
+	(*(+	AB
C	(*(+	ABC
)	(*	ABC+
-	(-	ABC+*
D	(-	ABC+*D
/	(-/	ABC+*D
E	(-/	ABC+*DE
)	empty	ABC+*DE/-

↳ Result

Q2 (4 $ 2 * 3 - 3 + 8 / 4 / (1+1))

Character Scanned	Stack	Postfix Expression
(	(	
4	(	4
$	($	4
2	($	42
*	(*	42$
3	(*	42$3
-	(-	42$3*
3	(-	42$3*3
+	(+	42$3*3-
8	(+	42$3*3-8

Good Write

/	C+/	42$3*3-8
4	C+/	42$3*3-84
/	C+/	42$3*3-84/
(	C+/(	42$3*3-84/
1	C+/(	42$3*3-84/1
+	C+/(+	42$3*3-84/1
1	C+/(+	42$3*3-84/11
)	C+/	42$3*3-84/11+
)	empty	42$3*3-84/11+/+

(Imp) # Evaluation of Postfix Expression

No brackets for pre / postfix evaluation

562 + * 124/−

Character Scanned	Stack
5	5
6	5, 6
2	5, 6, 2
+	5, 8
*	40
12	40, 12
4	40, 12, 4
/	40, 3
−	$\boxed{37}$

If operator comes, pop top 2 operands

⇒ 5 * 8 = 40

⇒ A → 2, B → 6, B op A, 6 + 2 = 8

⇒ A → 4, B → 12, B op A = 12/4

Q 42$3*3-84/11+/+

Character Scanned	Stack
4	4
2	4, 2
$	16
3	16, 3
*	48

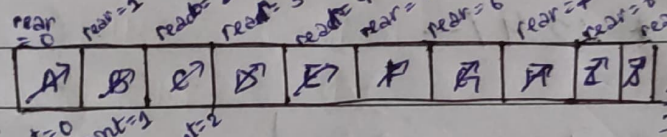
3	48, 3
−	45
8	45, 8
4	45, 8, 4
/	45, 2
1	45, 2, 1
1	45, 2, 1, 1
+	45, 2, 2.
/	45, 1
+	46

Monday
16/10/2023

Queue

→ Insert A, B, C
→ Deletion one element
→ Insert D, E
→ Delete two elements.

Arrays :- → Insert F, G, H, I, J, K



1st ly rear = -1

→ Delete 8 elements

Queue,
FIFO:- First in First Out

rear = rear + 1
queue [rear] = d;

```
int queue [10];
int Front = -1;
int rear = -1;
```

Deletion ←
Insertion ←

```
void addqueue ()
{
  int d;
  if ( rear == 9 )
     printf ( " Queue is full." );
  else
     {
        printf (" Enter the element to be inserted: ");
        scanf (" %d", &d);
        rear = rear + 1;
        queue [rear] = d;
        if (rear == 0)
           front = 0;
}
```

Queue full => Rear == max - 1 && front == 0

DATE: __/__/____
PAGE: _____

```
void delqueue()
{
    int d;
    if (front == -1)
        printf("Queue is empty.");
    else
    {
        d = queue[front];
        front = front + 1;
        printf("Deleted value is %d", d);

        if (front == rear)
        {
            front = -1;
            rear = -1;
        }
    }
}
```

Tuesday
17/10/2023

Circular Queue

Rear = 3 Front = 1

Queue - _, L, M, N, _, _
(1, 2, 3, 4, 5)
front Rear

a) Add O
b) Add P
c) Delete two letters
d) Add Q, R, S
e) Delete one letter.
f) Delete three letters.

Queue = —, L, M, N, —, —.
0 1 2 3 4 5

rear = (Rear+1) then insert

Delete then front+1

		Front	Rear	Queue
a)	Add O	1	4	—, L, M, N, O, —
b)	Add P	1	5	—, L, M, N, O, P
c)	~~Add~~ Delete 2 letters	3	5	—, —, —, N, O, P
d)	Add Q, R, S	3	2	Q, R, S, N, O, P

↳ ~~Queue full~~ applied
condition(?) front > 0 && rear = max - 1
then, value of rear = 0

Condition when queque is fulled
↳ (Rear == max - 1 && front == 0) || (rear = front - 1)

e)	Delete one letter	4	2	Q, R, S, —, O, P
f)	Delete three letters	1	2	—, R, S, —, —

wednesday
18/10/2023

Searching & Sorting

Search → linear ~~Binary~~ search
→ Binary Search

Sorting → Insertion Sort
→ Quick Sort
→ Radix Sort
→ Merge Sort
→ Heap Sort.

array of 10 elements

25	5	17	52	40	23	8	14	36
0	1	2	3	4	5	6	7	8

23
num

```
for (i=0; i<10; i++)
{
    if (num == arr[i])
    {
        printf("Element found at position %d", i+1);
        flag=1;
    }
}
if (flag==0)
```

Good Write

In beginning

```
int num,i;
int flag=0;
int arr[10];
printf("Enter the value of array:");
scanf(for (i=0; i<10; i++)
scanf("%d", &arr[i]);
printf("Enter the number to be searched: ");
scanf("%d", &num);
for(
```

printf("Element not found in the given list");

Binary search

First ~~search~~ sort the array

l=0, h=0

5	8	14	17	23	25	36	40	52
25	5	17	52	40	23	8	14	36
0	1	2	3	4	5	6	7	8

mid = (l + h)/2

If num < mid, change h = mid - 1

num > mid change l = mid + 1

Friday 20/10/2023

① # Insertion Sort

Compare

25	17	13	31	7	12
0	1	2	3	4	5

I^st iteration

17	25	13	31	7	12
0	1	2	3	4	5

2nd iteration

13	17	25	31	7	12
0	1	2	3	4	5

3rd iteration

13	17	25	31	7	12
0	1	2	3	4	5

4th iteration

7	13	17	25	31	12
0	1	2	3	4	5

5th iteration

7	12	13	17	25	31
0	1	2	3	4	5

Good Write

DATE: ___/___/___
PAGE: _____

Important

② # Quick Sort

0	1	2	3	4	5	6	7	8	9
11	2	9	13	57	25	17	1	90	3

Pivot Element → PE i=1 i=2 i=3 j=9

i++ swap

i=3
13>11 stop

3<9

j=9
3<11
stop

if $i<j$, then swap the value of elements at i^{th} and j^{th} position.

0	1	2	3	4	5	6	7	8	9
11	2	9	3	57	25	17	1	90	13

PE i=3 i=4 j=7 j=8 j=9

i++ swap j--

4<j

0	1	2	3	4	5	6	7	8	9
11	2	9	3	1[illegible]	25	17	57	90	13

PE j=4 j=7

i=3 i=5

if $j \leq i$, then swap PE & value at j

Exact position of 11

0	1	2	3	4	5	6	7	8	9
1	2	9	3	11	25	17	57	90	13

PE i=1 j=3 PE i=6 i=7 i=9

j=0

2 arrays

1	2	9	3

PE i=2 j=3

25	17	13	90	57

PE i=6 i=7 i=8 j=9 j=7

1	2	9	3

PE i=2 i++ j=3 j-- swap

13	17	25	90	57

PE i=7 8 9 7 8 j=9

13	17	25	90	57

PE PE → | 13 | 17 | 25 | 57 | 90 |

Good Write

DATE: __/__/____
PAGE: ____

(3)

Radix Sort

Convert into largest no digit 677 → 3 digit
so convert all into 3 digit.

Start from last

0	1	2	3	4	5	6	7	8	9
432	008	530	090	088	231	011	045	677	199

unit place pr ye digits kitne no. me h

for unit place count →

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

Decrement by 1

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

unit place pr element h usko decrement by 1 krke jo index aya usse kitna h

array k index me jo value h

0	1	2	3	4	5	6	7	8	9
530	090	231	011	432	045	677	008	088	199

0	1	2	3	4	5	6	7	8	9
530	090	231	011	432	045	677	008	088	199

for Tens place

count

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

from last same as unit place

0	1	2	3	4	5	6	7	8	9
0	1	2	4	5	6	6	7	8	10

0	1	2	3	4	5	6	7	8	9
008	011	530	231	432	045	677	088	090	199

for Hundredth place

0	1	2	3	4	5	6	7	8	9
008	011	530	231	432	045	677	088	090	199

count

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

0	1	2	3	4	5	6	7	8	9
5	6	7	7	7	8	9	10	10	10

0	1	2	3	4	5	6	7	8	9
008	011	045	088	090	199	231	432	530	677

DATE: ___/___/___
PAGE: _____

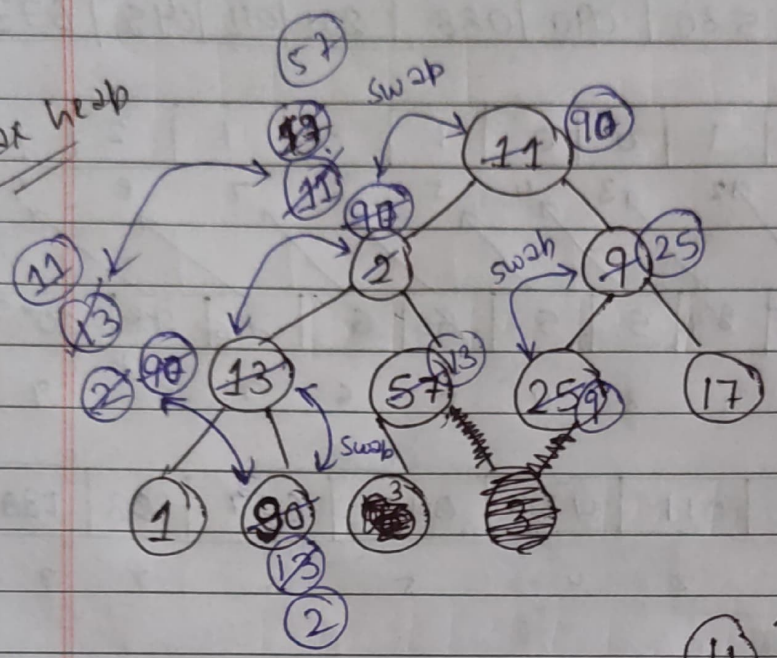
(4)

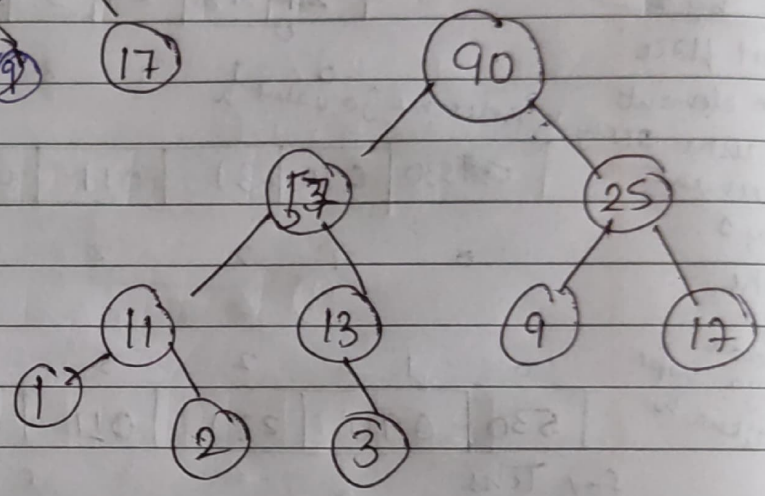
Heap Sort

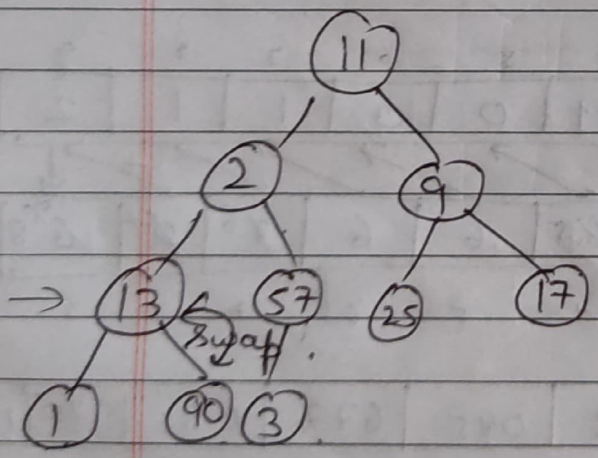
0	1	2	3	4	5	6	7	8	9
11	2	9	13	57	25	17	1	90	3

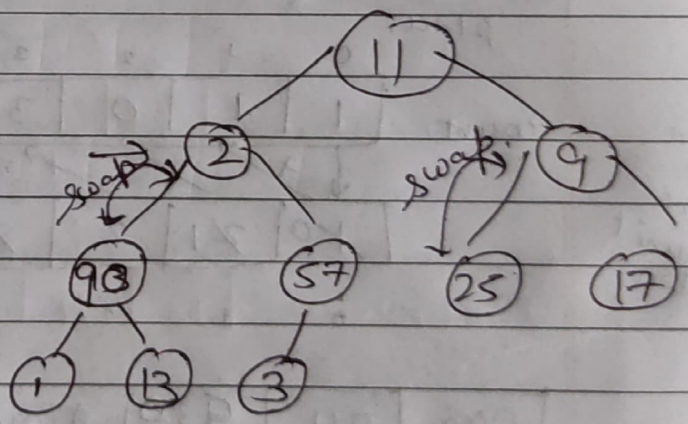
Max heap

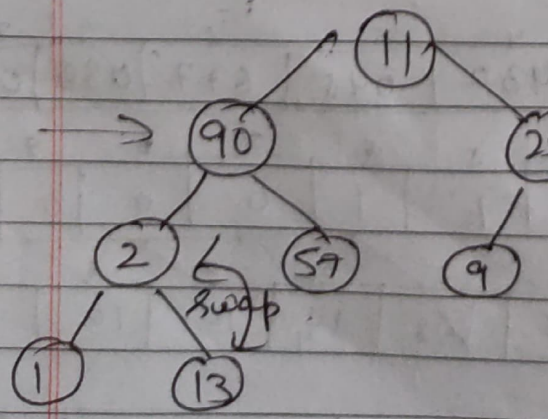
Max Heap
Min Heap

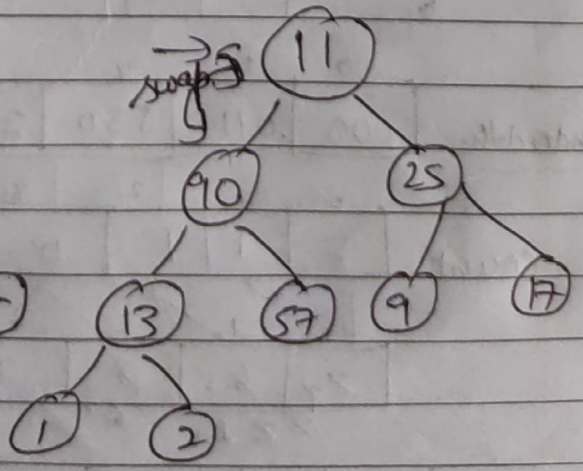




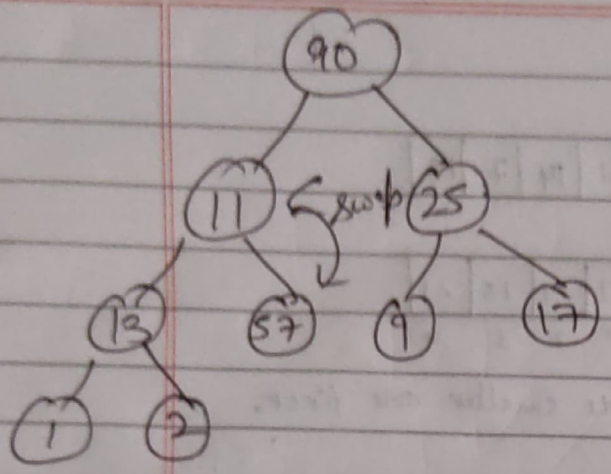






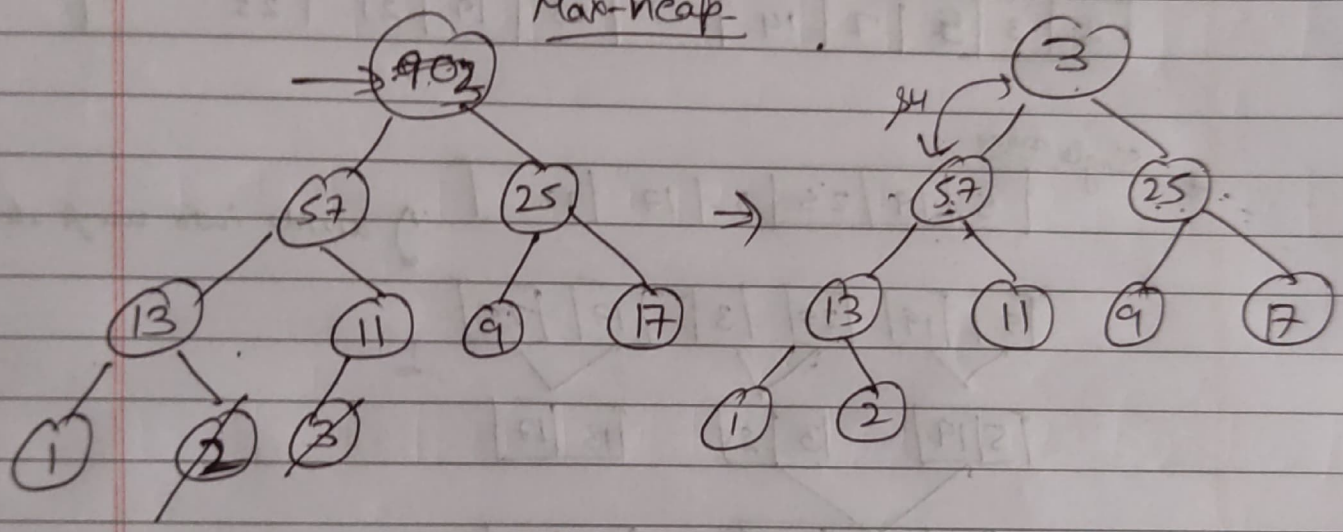


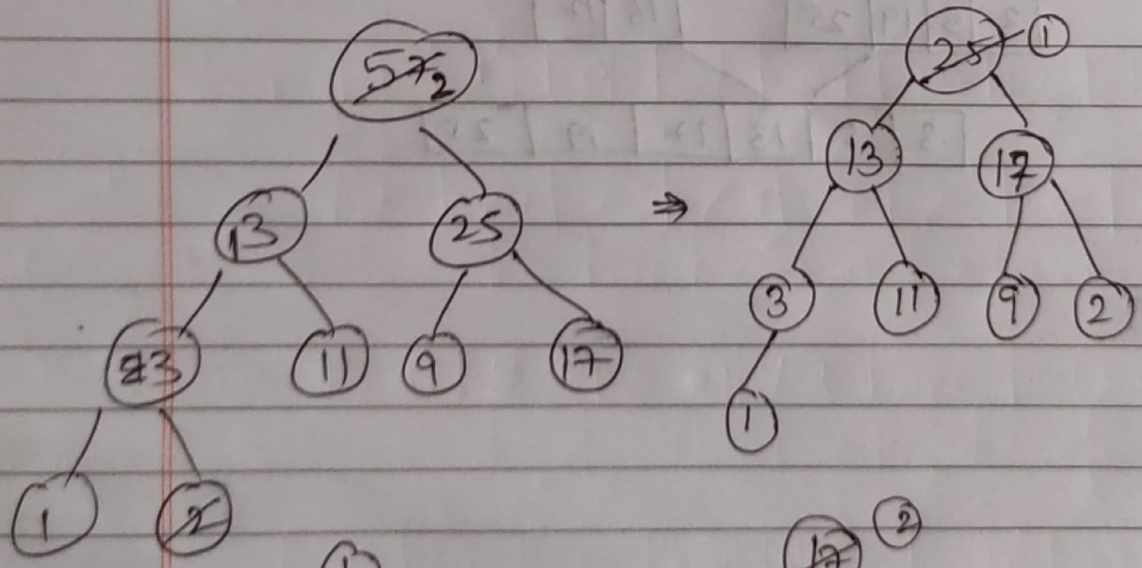
Good Write

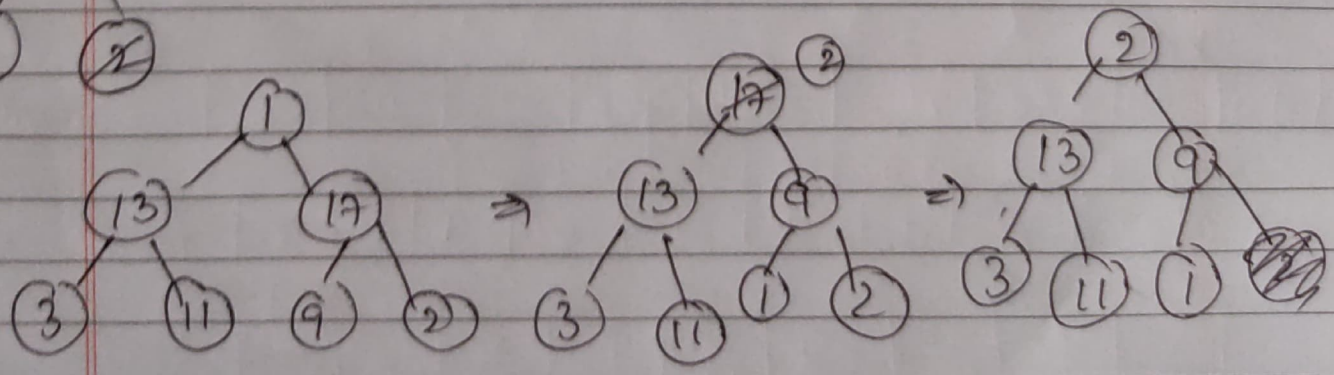


		17	25	57	90

Max-heap







Merge Sort

5	19	25	3	17

8	21	14	7	18

Sort with any merge technique

a

3	5	17	19	25
i=0	1	2	3	4

b

2	7	14	18	21
j=0	1	2	3	4

compare a[i] and b[j] & write smaller one first.
& increment by 1

2	3	5	7	14	17	18	19	21	25

* Single array.

5	19	25	3	17	13

divide into single element

5	19	25	3	17	13

5	19	3	25
13	17		

3	5	19	25	13	17

3	5	13	17	19	25