Pointers in C

- holds address of other variable

```
int a = 100;
```

a: 100 (FFFS62 ← adds)

Syntax:-

```
int *p;
```

```
int b = 100;
p = &b;          ← referencing
printf("%d", *p);  ← o/p => 100
                 ← dereferencing
```

```
# include <stdio.h>
# include <conio.h>
main()
{
  int a = 100;
  int *p;
  p = &a;
printf("value of a: %d", a); → 100
printf("value of a: %d", *p); → 100
pf("%p", &a); → 2561
pf("%p", p); → 2561
pf("%p", &p); → 3F
```

a: 100 (2561) p: 2561 (3F)

Scanned with CamScanner

02 Wednesday January 2013

Data Structures

- A data structure is a way to store and organize data in a computer, so that it can be used efficiently.

- We talk about data structure as:
  1) Abstract view
     eg TV
     - turns on/off
     - Receives signals.
     - Audio/video play.

     We do not bother how circuits are embedded & which company has manufactured it.

  2) Implementation.

Abstract Data type :-
Defines data & operations, but no implementations.

Scanned with CamScanner

03 Thursday January 2013

Types classification.

Data structure

- Primitive DS
  - Integer
  - Character
  - float
  - Pointers.
- Non Primitive DS.
  - Linear DS
    - Array
    - Stack
    - Queue
    - LinkedList
  - Non-linear DS.
    - Tree
    - Graph

The List of operations applied on linear data structures.

- Add an element
- Delete an Element
- Traverse/Display
- Sort the list
- Search for element.

Scanned with CamScanner

04 Friday January 2013

1) List (very basic)
- store a given no. of elements of perticular datatype
- write or modify any element at ~~an~~ at a position.
- Read a element at any position.

Q) Are we talking about Arrays?

How to compile in Ubuntu?

compile : gcc filename.c
hello.c
gcc hello.c
if output file is created. a.out

Run ./a.out

Scanned with CamScanner

05 Saturday January 2013

Stack

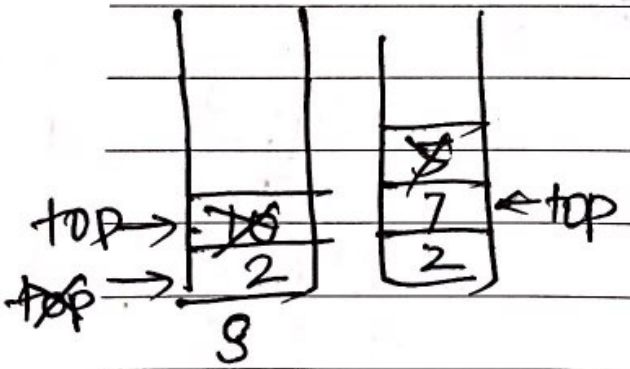
A List with the restriction that insertion & deletion can be performed only from one end, called the TOP.

operations:

1) push(x)
2) pop()
3) Top()
4) IsEmpty()
5) Is Full()

} constant time or O(1) } LIFO

06 Sunday



Push(2)
push(10)
pop()
push(7)
push(5)
Top() → 5
IsEmpty → false
pop().
Top() → 7

App^n :-

(4) Evaluation of Expressions.
(1) - undo in an editor
(3) - Balance Parantheses
[{( )}]
(2) - Reverse a list

07 Monday January 2013

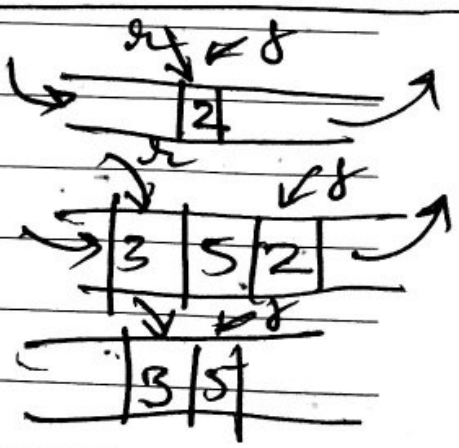
Queues

A list or collection with the restriction that can be performed at one end (rear) & deletion can be performed at the other end (front)

1) Enqueue(x) / push(x)
2) dequeue() / pop()
3) Front()
4) IsEmpty()
5) IsFull() ← optional

} O(1)

Enq(2)
Enq(5)
Enq(3)
dq()
front() → 5
IsEmpty → false



Scanned with CamScanner

08 Tuesday January 2013

Applications:

1) Resource service → printer.
2) Processor works on processes.
   ↳ processes scheduling
3) Simulating wait.

Array Implementation :-

0	1	2	3	4	5	6	7	8	9
	2	4	6	8	10	12			

front rear

enqueue (2)
" (4)
" (6)
" (8)
" (10)
" (12)

dequeue()

Scanned with CamScanner

09 Wednesday January 2013

Stack Implementation

- Using Array :-

```
void push(int x)
{
  if (top == MAX-1
```

```
#include<stdio.h>
#define MAX 101
  int a[MAX];
  int top = -1;

  void push(int x)
{
    if (top == MAX - 1)
  {
    pf ("stack is full");
    return;
    }
    top++;
    a[top] = x;
}
```

} a[++top] = x;

Scanned with CamScanner

```
void top1( )
{
  if (top==-1)
  {
   pf("stack Empty");
  }
  else
   pf("%d", a[top]);
}

void print()
{
  int i;
  pf("stack:\n");
      for (i=0; i<=top; i++)
      {
        pf("%d", a[i]);
      }
}

void pop()
{ if (top==-1)
  {
    pf("No elements);
    return;
  }
  top--;
```

Scanned with CamScanner

```
main()
{
  push(2); print()
  push(5); print()
  push(8); print()
  pop(); print();
  push(12); print();
}
```

Complexity < space / time.

- compare various algorithms & & choose the best one to solve a problem
- for that < space / time complexity
- Time complexity :-
  the amt of time taken by an algorithm to run the instructions.
- Space Complexity :-
  the amt of space or memory taken by an algorithm to run as the instructions.

Scanned with CamScanner

12 Saturday January 2013

- They both depend on various factors like h/w, OS, processors etc but dont consider any of them
- We only consider execution ~~of~~ time of an algo.

push (x) — O(1) : best & Average case
— O(n) worst case.

pop()
Top()
IsEmpty() } O(1)

13 Sunday

Check for Balance Paranthesis

(A+B) ✓ [()()]
[(A-B) ✗
{(a*b) ✗ ⇒ ~~Scan from L → R~~

[(]) ✗

)(✗

[()] ✓

Solution:-

1) Scan from left to right push
2) If opening symbol, add it to a ~~list~~ stack pop
3) If closing symbol, ~~remove~~ last opening symbol in ~~list~~ stack

[( ) ( )]

List : ~~[X X~~ Empty

Evaluation of expressions

2+3 — operand

A−B operator

variable a*2 — constant

(2+3) * 4

operand

* (p+q)*(r+s)

* 4+6*2

(4+6)*2

OR

4+(6*2)

Scanned with CamScanner

Order of operations

1) Parentheses
2) Exponents (R→L).
3) Multiplication & division (L→R)
4) Addition & Subtraction. (L→R)

eg. ① $4+6*2$ $(4+6)*2$

$4+12 = 16$

② $2*6/2 - 3+7$

$\Rightarrow 12/2 - 3+7$

$\Rightarrow 6 - 3+7$

$\Rightarrow 3+7$

$\Rightarrow 10$

③ $[(2*6)/2] - (3+7) = -4$

$(12/2) - 10 =$

$6 - 10 = -4$

Scanned with CamScanner

16 Wednesday January 2013

✓ Prefix (Polish Notation)
✓ Postfix (Reverse Polish Notation)

✓ Infix	Prefix	Postfix
2+3	+23	23+
p-q	-pq	pq-
a+b*c	+a*bc	abc*+

Good for machines

human readable prefix

a+(b*c) ✓	a+(b*c) ✓
a+(*bc)	a+(bc*)
+a(*bc)	a(bc*)+
+a*bc ✓	abc*+ ✓

Evaluation of prefix and postfix expressions :-

Infix to Postfix

[(A*B)+(C*d)]-e

⇒ [(AB*)+(cd*)]-e

[(AB*)(cd*)+]-e

[(AB*)(cd*)+]e-

AB* cd*+e-

① Put parenthises are preference

06 Aditya Choudary

17 Thursday January 2013

Postfix Evaluation

$AB^* \; cd^* + e^-$

$A=2, B=3, C=5, d=4, e=9$

$\Rightarrow (2\ 3^*), (5\ 4^*) + 9^-$

$\Rightarrow [6\ \ 20\ +] \; 9 -$

$\Rightarrow 26\ \ 9 -$

$\Rightarrow 17$

$2\ 3^*\ \ 5\ 4^*\ +\ 9^-$

$\Rightarrow 6\ \ 5\ 4^*\ +\ 9^-$

$\Rightarrow 6\ \ 20\ +\ 9^-$

$\Rightarrow 26\ \ 9 -$

$\Rightarrow 17$

17
9 — 2
26 — 1

26 - 9

*	*	+	+	-	
3 2	4 5 6	20 6	20 6	26	9 26	17

26-9

9 = OP2
26 = OP1

26 - 9 = 17

Infix to prefix

[(A*B) + (C*d)] - e

[(*AB) + (*cd)] - e

[+(*AB)(*cd)] - e

- +(*AB)(*cd) e

-+ *AB *cd e

A=2, B=3, C=5, d=4, e=9

-+ * 2 3 (* 5 4) 9

-+ (* 2 3) 20 9

-+ 6 20 9

- 26 9

17

Scanned with CamScanner

19 Saturday January 2013

prefix Evaluation

- + * 2 8 (* 5 4) 9

op1 op2
26 - 9

*		*	+	-	
5		2	6		
4	20	3	20	26	
9	9	20	9	9	17
		9			

26 - 9

eg ② ABC+*DE/-

A = 5, B = 6, C = 2, D = 12, E = 4

5 6 2 + * 12 4 / -

5 8 * 12 4 / -

40 12 4 / -

40 3 -

20 Sunday = 37

+	*	/	-
2		4	
6	8	12	3
5	5	40	40	37

Scanned with CamScanner

2^3

21 Monday January 2013

Prefix $-+/A\wedge BC*DE*AC$

$A=16, B=2, C=3, D=10, E=4$

$-+/16\wedge 2\ 3*\ 10\ 4*\ 16\ 3$

$-+/16\wedge 2\ 3*\ 10\ 4\ 48$ $\quad 2 = 2\wedge 3$

$-+/16\wedge 2\ 3*\ 40\ 48$ $\quad \wedge, \$, \%, \uparrow$ $\quad 2\ 2\wedge 3$ $\quad 3$

$-+/16\ 8\ 40\ 48$

$-+\ 2\ 40\ 48$ $\quad 16/8$

$-*\ 42\ 48$

$= -6$

$-+/A\wedge BC*DE*AC$

$42-48$

*		*		2	16	/	+	-
	10	3		8		2		
16	4	40		40		40	42	
3	48	48		48		48	48	-6

2^3

Priority of operators

1. exponential, %
2. / *
3. + -

22 Tuesday January 2013

Infix to Postfix using stack

(A + B/C * (D+E) - F).

Symbol	Stack	Postfix
(	(	
A		A
+	(+	A
B	(+	AB
/	(+/	AB
C	(+/	ABC
*	(+*	ABC/
(	(+*(	
D	(+*(	ABC/D
+	(+*(+	ABC/D
E	(+*(+	ABC/DE
)	(+*(+)	ABC/DE
	(+*	ABC/DE+
-	(- (+*	ABC/DE+*
F	(-	ABC/DE+*F
)	(	ABC/DE+*+F-

↓ GFE^D/*CB*-A+

~~+A-*B.C*/D^EFG~~

23 Wednesday January 2013

Infix to prefix

① Expression

② → reverse

③ → infix to postfix

④ → reverse o/p

① ^ ↑ $ %

② / *

③ + -

A+(B*C-(D/E^F)*G)

(G*(F^E/D)-C*B)+A

Symbol		Stack	Postfix
)	(	(	
G	G	(	G
+	*	(*	G
(	(	(*(	G
B	F	(*(	GF
*	^	(*(^	GF
C	E	(*(^	GFE
-	/	(*(^/	GFE
(	D	(*(^/	GFED
D	)	(*	GFED/^
/	-	(-	GFED/^*
E	C	(-	GFED/^*C
^	*	(-*	GFED/^*C
F	B	(-*	GFED/^*CB
)	)	(	GFED/^*CB*-
*	+	(+	GFED/^*CB*-
G	A	(+	GFED/^*CB*-A
)			GFED/^*CB*-A+

Scanned with CamScanner

Postfix = GFED/^*CB*-A+
Prefix = +A-*BC*^/DEFG

24 Thursday January 2013

(A*B)+C
→ C+(B*A)

Symbol	Stack	Postfix
C		C
+	+	C
(	+(	C
B	+(	CB
*	+(*	CB
A	+(*	CBA
)	+	CBA*+

+*ABC
prefix

(A+B)+C

(AB+)+C

AB+C+

~~+C+BA~~ X

Postfix evaluation

Algorithm to convert infix to postfix

① Add ")" to the end of the infix expression

② Push "(" on the stack.

③ Repeat until each character in the infix notation is scanned.

1.) If a "(" is encountered, push it on the stack

~~2.) Discard the "(" i.e. remove "(" from stack & do not add it to the postfix expression~~

2.) If a ")" is encountered then

ⓐ Repeatedly pop from stack & add it to the postfix expression until "(" is encountered.

ⓑ Discard ")" i.e remove "(" from the stack & do not add to the postfix expression.

Scanned with CamScanner

26 Saturday January 2013

3.> If an operator is encountered then

(a) Repeatedly pop from stack & add each operator to postfix expression which has same precedence or a higher precedence than operator encountered.

(b) push the operator (encountered operator) on to the stack.

end of if.

27 Sunday

(4.> If a ~~number~~ operand is encountered then add it to postfix expression.

(4) Repeatedly pop from the stack & add it to the postfix expression. until the stack is empty.

(5) Stop.

Scanned with CamScanner

Program:-

```
# include<stdio.h>
# includ<ctype.h>
# define MAX 50

typedef struct stack
{
  int data[MAX];
  int top;
} stack;

int precedence(char);
void init(stack *);
int ~~init~~ empty(stack *);
int full(stack *);
int pop(stack *);
void push(stack *, int);
int top(stack *); //value of top element
void infix_to_postfix(char infix[],
                      char postfix[]);
```

Scanned with CamScanner

```
void main()
{
  char infix[30], postfix[30];
 pf("Enter an infix expression \n");
 sf("%s", infix);
 infix_to_postfix(infix, postfix);
 pf("\n Postfix expression: %s\n", postfix);

}.

void infix_to_postfix(char infix[],
                      char postfix[])
{
 Stack S;
 Char x, token;
 int i, j; //i→infix index & j→postfix index
 init(&S);
 j=0;

 for(i=0; infix[i] != '\0' ; i++)
 {  token = infix[i];
   if(isalnum(token))
      postfix[j++] = token;
   else
```

Scanned with CamScanner

```
    if(token == '(')
      push(&S, '(');
else
    if(token == ')')
    while ((x = pop(&S)) != '(')
      postfix[j++] = x;
  else
  {while(precedence(token) <=
          precedence(top(&S)) &&
          !empty(&S))
    {
      x = pop(&S);
      postfix[j++] = x;
    }
    push(&S, token);
  }
}   // end of for
```

Scanned with CamScanner

```
while(!empty(&S))
{
    x=pop(&S);
    postfix[j++]=x;
}
postfix[j]='\0';
} //end of infix_to_postfix

int precedence(char x)
{
  if(x=='(')
    return(0);
  if(x=='+'||x=='-')
      return(1);
  if(x=='*'||x=='/'||x=='%')
    return(2);
  else
//  return(3);
}
```

Scanned with CamScanner

```
void init(stack *s)
{
    s->top=-1;
}

int empty(stack *s)
{
    if(s->top == -1)
        return(1);
    else
        return(0);
}

int full (stack *s)
{
    if(s->top == MAX-1)
        return(1);
    else
        return(0);
}
```

Scanned with CamScanner

02 Saturday February 2013

```
void push(stack *s, int x)
{
 s->top = s->top+1;
 s->data[s->top] = x;
}

int pop(stack *s)
{
   int x;
   x = s->data[s->top];
   s->top = s->top-1;
   return(x)
}
```

03 Sunday

```
int top(stack *p)
{ return(p->data[p->top]);
}
```

Scanned with CamScanner

04 Monday February 2013

infix to postfix evaluation

```
# include <stdio.h>

# define MAX 20

typedef struct stack
{
    int data[MAX];
    int top;
} stack;

void init(stack *);
int empty(stack *);
int full(stack *);
int pop(stack *);
void push(stack *, int);
int evaluate(char x, int op1, op2);

int main()
{
    stack S;
    char x;
    int op1, op2, val;
```

Scanned with CamScanner

```
init(&s)
pf("Enter Expression");

while((x=getchar()!='\n')
{
  if(isdigit(x))
    push(&s, x-48);
  else
  {
    op2 = pop(&s);
    op1 = pop(&s);
    val = evaluate(x, op1, op2);
    push(&s, val);
  }
}

val = pop(&s);
pf("\n Value of expression=%d", val);

  return 0;
}
```

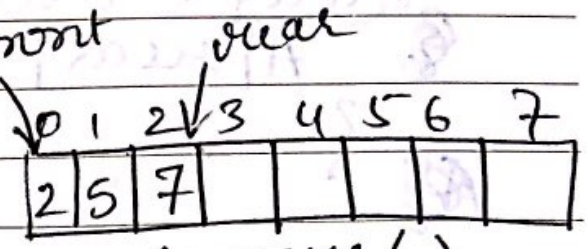
Scanned with CamScanner

```
int evaluate(char x, int op1, int op2)
{
    if(x=='+')
      return(op1+op2);
    if(x=='-')
      return(op1-op2);
    --
    --

}
```

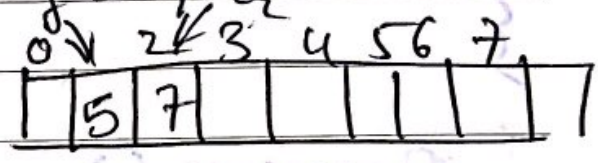
Scanned with CamScanner

Queue Implementating using Array.

```
int A[10]
front ← -1
rear ← -1
```



dequeue()




```
IsEmpty()
{
  if front = -1 && rear = -1
      return true
  else
      return false
}

Enqueue (x)
{
  if rear == size(A) - 1
      print "Q is full"
      return
  else if IsEmpty()
  {
      front ← rear ← 0
      ~~A[rear] ← x~~
  } A[rear] ← x
```

OR

if (rear+1)%N == front

```
else
{
    rear ← (rear+1) % N
    rear ← rear + 1
    A[rear] ← x
}
}
```

current pos = i
Next " = (i+1) % N
previous = (i + N-1) % N

```
Dequeue()
{
  If IsEmpty().
     return()
  else if front == rear
     front = rear = -1
  else
     front = (front+1) % N
     front ← front + 1
}

front()
{
  return A[front]
}
```

Scanned with CamScanner

09 Saturday February 2013

Implementation of Circular Queue

```
# include <stdio.h>
# define MAX 5

int queue_array[MAX];

int rear = -1;
int front = -1;

int main()
{
  int choice
```

10 Sunday

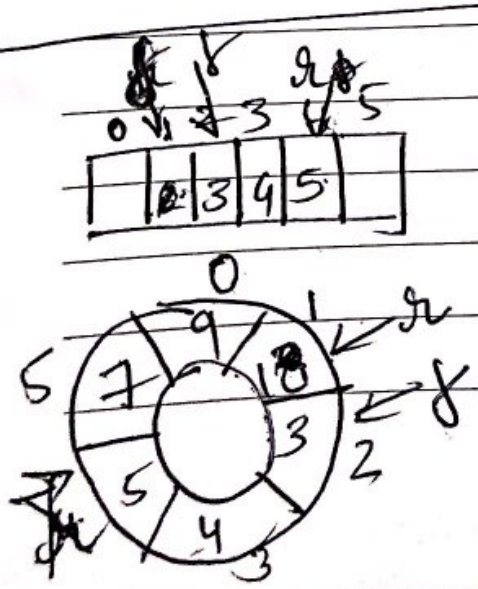
```
insert ()
{ int add_item;
  if((rear+1)%MAX == front)
  pf("Q overflow");
  else
  {
    if(front == -1)
    front = rear = 0;
    pf("insert an element");
    sf("%d, &add_item);
```

rear = MAX-1

```
    rear = (rear+1) % MAX;
    queue_array[rear] = add_item;
  }
}

delete()
{
  int element;
  if(front == -1)
  { pf(" Underflow");
    return(-1);
  }
  else
  {
    element = queue_array[front]
    if(front == rear)
      front = rear = -1;
    else
    {
      front = (front+1) % MAX;
      printf("the deleted element
          is %d", element);
    }
  }
}
```

Scanned with CamScanner

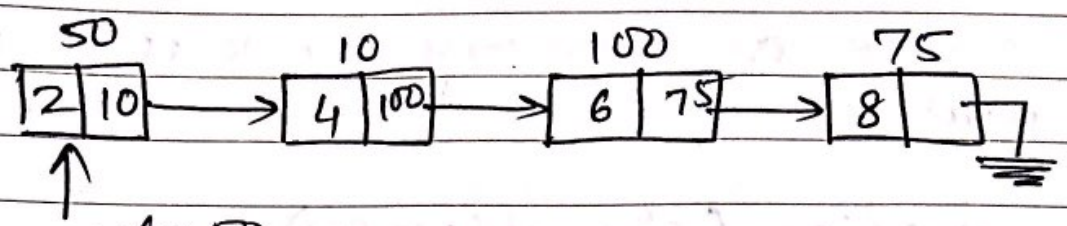
```
display1()
{
    int i, j;
    if (front = -1 && rear = -1)
    {
      pf(Q is underflow);
      return;
    }
    if (front > rear)
    {
      for( i=0; i<=rear; i++)
        pf("%d", queue_array[i]);
      for (j=front; j<=MAX-1; j++)
        pf("%d", queue_array[j]);
      pf("\n rear is at %d\n", queue_array[rear]);
      pf("\n front is at %d\n", queue_array[front]);
    }
```



```
else
{
  for(i=front; i<=rear; i++)
  {
    pf("%d", queue_array[i]);
  }
  pf("rear is at %d", queue_array[rear]);
  pf("front is at %d", queue_array[front]);
}
}
```

Scanned with CamScanner

Linked List



head = 50

```
Struct Node
{
    int data;
Struct Node * next;
};
```

Operations:-

1. insert a node
2. Traverse the list.

pointer to head node

```
Struct Node * head;
```

- head = NULL → [main()]

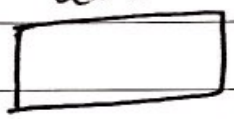
15 Friday February 2013

<u>insertion ()</u>

① create a memory block using malloc

$= \underbrace{\text{malloc (size of(Node))}}_{\text{give me block with size of type Node}}$

200



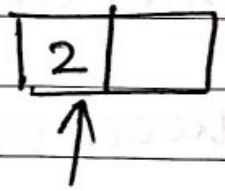
for type cast bcoz malloc return void pointer

```
struct Node* temp = (struct Node*)
        malloc(size of (struct Node));
```

*head=0

~~*temp~~

200



temp = 200

```
temp -> data = 2 ;
temp -> next = NULL;
head = temp;
```

again new node

16 Saturday February 2013

Traverse

```
Node* temp1 = head
while (temp1 -> next != NULL)
{
  temp1 = temp1 -> next;
}

temp1 -> next = temp;
```

17 Sunday

Scanned with CamScanner

18 Monday February 2013

insert at the begining of List

```
# include <stdlib.h>
# include <stdio.h>

struct Node
{ int data;
  struct Node* next;
};

struct Node* head;
void insert(int x);
void print();

int main()
{
  head= NULL;
  pf(" How many numbers);
  int n, i, x;
  sf ("%d", &n);
  for( i=0; i<n; i++)
  {
  pf("Enter the number \n");
  sf("%d " &x);
  insert(x);
  print();
  }
```

Scanned with CamScanner

```
void insert(int x)
{
struct Node* temp = (struct Node*)
    malloc(size of (struct Node));

  temp -> data = x;
  temp -> next = head;
  head = temp;
}

void Print()
{
struct Node *temp = head;
printf("List is: ");
while (temp -> next != NULL)
{
  printf("%d", temp -> data);
  temp = temp -> next;
}
}
```

Scanned with CamScanner

20 Wednesday February 2013

insert node at the end of LL

```
void insert(int ~~data~~ x)
{
 struct Node* temp1 = (struct Node*)
        malloc(size of(struct Node));

 temp1 -> data = x;
 temp1 -> next = NULL

  if(head == NULL)
  {
     head = temp1;
     return;
  }

 Struct Node* temp2 = head;
   int i;
 while(temp2->next != NULL)
 {
   temp2 = temp2 -> next;
 }
 temp2 -> next = temp1;
}
```

Scanned with CamScanner

insert node at nth position

```
void insert(int x, int n)
{
 struct Node * temp1 = (struct Node*)
     malloc(sizeof(struct Node));
  temp1 -> data = x;
  temp1 -> next = NULL;
   if (n==1)
   {
      temp1 -> next = head
      head = temp1
      return;
   }

 Node* temp2 = head;
 int i;
 for (i=1; i<n-1; i++)
 {
   temp2 = temp2 -> next;
 }
  temp1 -> next = temp2 -> next
  temp2 -> next = temp1;
}.
```

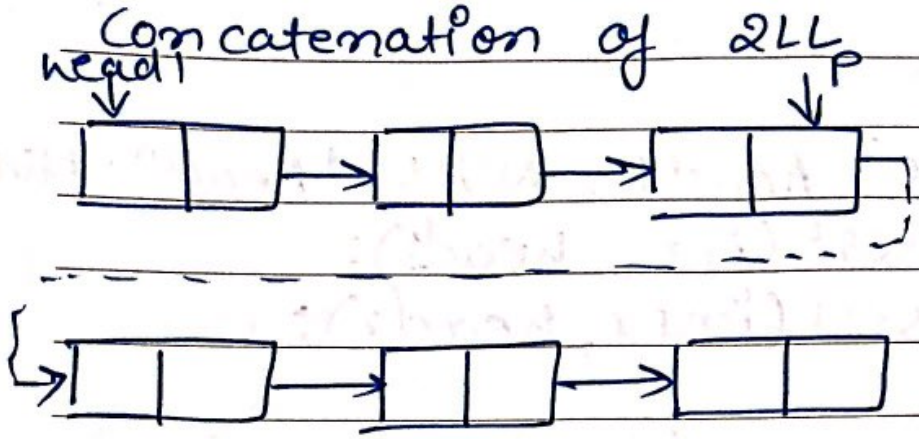
Scanned with CamScanner

Delete a node at n^{th} position

```
void delete(int n)
{
struct Node * temp1 = head;
if (n==1)
{
 head = temp1 -> next;
 free (temp1);
 return;
}
   int i;      n=2
for(i=0; i<n-2; i++)
   temp1 = temp1 -> next;
struct Node* temp2 = temp1 -> next

temp1 -> next = temp2 -> next
free (temp2);
}
```

Scanned with CamScanner

Concatenation of 2LL




```
struct node * concatenate(struct node * head1,
                          struct node * head2)
{
    node *p;
    if (head1 == NULL)
        return(head2);
    if (head2 == NULL)
        return (head1);
    p = head1;
while (p -> next != NULL)
{ p = p -> next;
  p -> next = head;
  return (head 1);

}
```

24 Sunday

Scanned with CamScanner

```
main()
{
Struct Node * head1 = NULL, *head2=NULL;
head1= insert (int x, head1);
head2 = insert(int x, head2);

pf(" First List is : ");
Display (head1);
pf(" Second list is ");
Display(head 2);

head1 = concatenate (head1, head2)
pf("Final list is ");
display(head1);
}
```

Scanned with CamScanner

Searching a data 'x' in LL

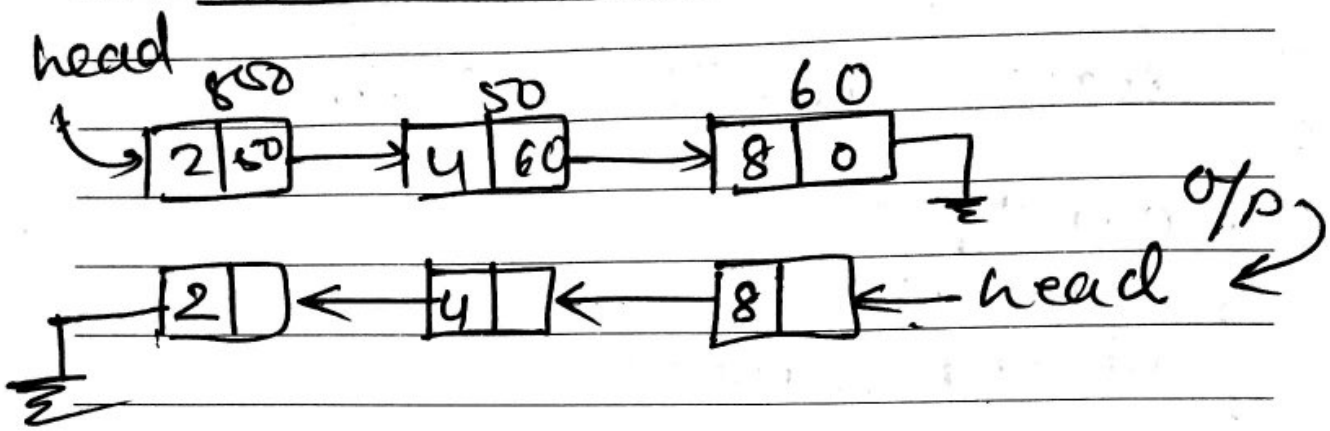
```
int search (node *head, int x)
{
  node *p;
  p = head;
  while (p != NULL)
  {
    if (p->data == x)
      return (1);
      p = p->next;
  }
  return(0);
}
```

OR

```
int search(int num)
{ int flag=0;
  struct node *temp = head;
  while (temp != NULL)
  { if (temp->data == num)
      return (1); // Found
    temp = temp->next;
  }
  if (flag == 0)
  { return(0);
```

Scanned with CamScanner

Reverse a Linked List




```
void Reverse()
{
  struct Node *current, *prev, *next;
   current = head;
   prev = NULL
while (current != NULL)
  {
     next = current -> next;
     current -> next = prev;
      prev = current;
      current = next;
  }
  head = prev;
}
```

Scanned with CamScanner

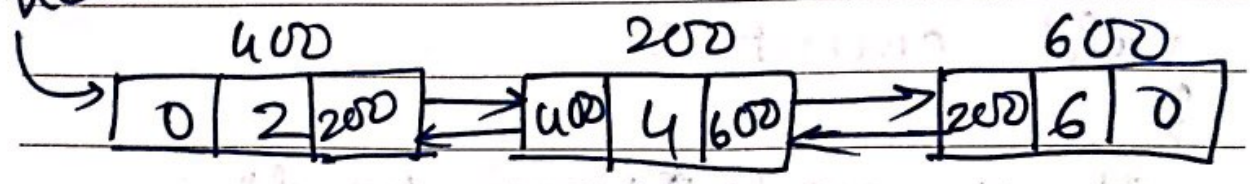
28 Thursday February 2013

* Counting nodes on LL.

```
void count()
{
  struct node *temp=head;
  int length=0;
  temp= start;
  while(temp! =NULL)
  {
    length++;
    ~~length~~ temp = temp->next;
  }
pf("In length is %d", length");
}
```

Scanned with CamScanner

Doubly Linked List.



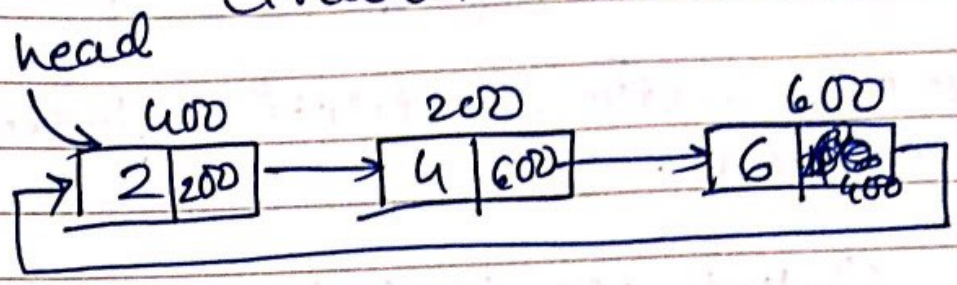
Adv :- Forward as well reverse lookup is possible

Disadv :-

Extra memory for the pointer to the previous node.

Scanned with CamScanner

Circular Linked List.



- Any node can be starting point
- useful in application to repeatedly go around the list.

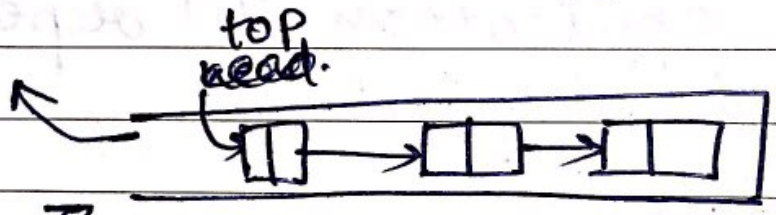
```
void display()
{
  if(head = = NULL)
    pf(" list Empty");
  else
  { x = head;
    while((x -> next ! = head))
    { pf("%d", x -> data);
      x = x -> next;
    }
    pf("%d", x -> data);
  }
}.
```

06 Wednesday March 2013

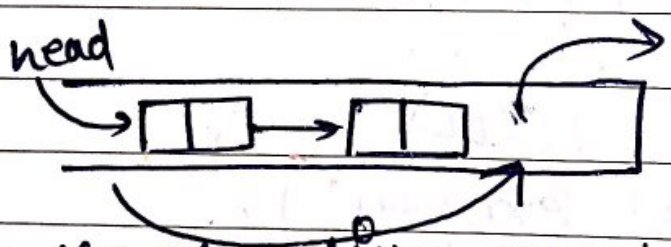
Linked List — Implement stack

Stack :- Insertion & deletion from one end.

with linked we have two options:-
1) insert/delete from head ✓
2) insert/delete from tail ✗



Insertion at head O(1)



Insertion/deletion at tail O(n)

but the operations must take constant time

Adv :- List will never be full

disadv :- More memory required.

Scanned with CamScanner

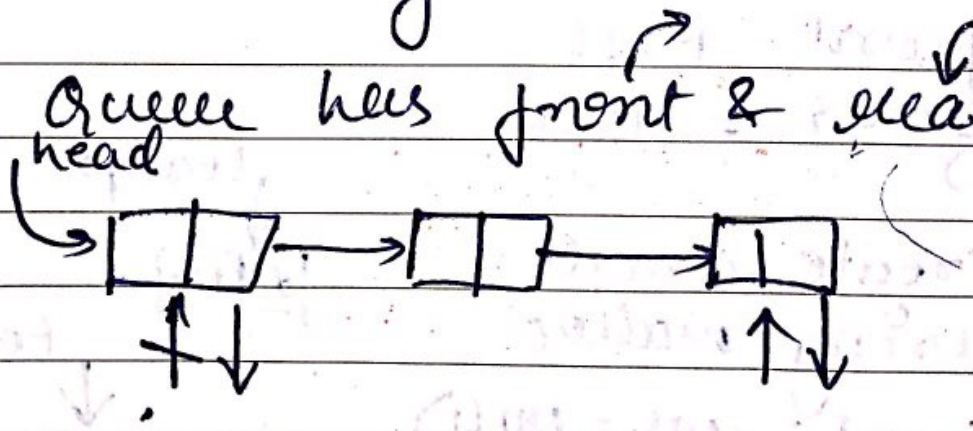
Queue Implementation using LL

- we used circular queue
- disadv :- 1) Array gets filled.
  2) we can ~~to~~ ask an array & copy all the previous elements to the array.

Time complexity → O(n)

3) If we are using larger array then there can be unused memory.

Queue has front & rear



But with LL when you ~~insert/~~

- insert/delete at head O(1)
- insert/delete at tail O(n)

Scanned with CamScanner

ur queue with ea. nd.

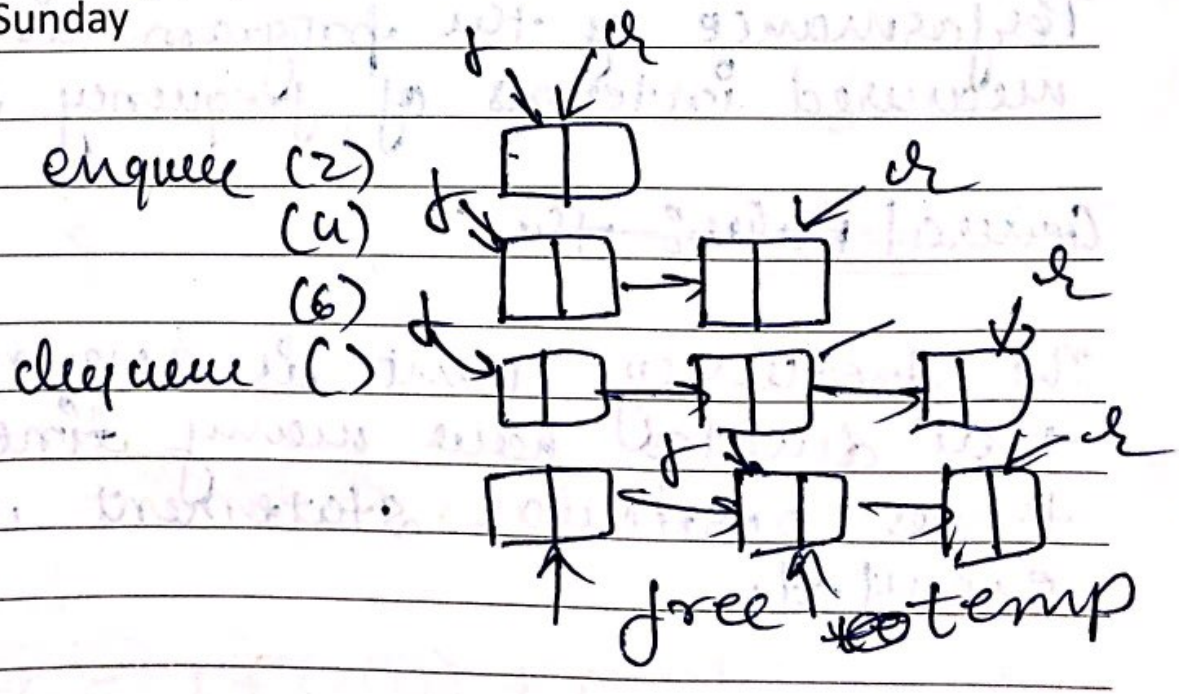
Step 5 :- Insert new node at rear end

set rear → next = temp
rear = temp

step 6 :- Repeat 2 to 5 to enqueue

Step 7 :- To dequeue [temp = front]
a) if (front == NULL) return
b) if (front == rear)
front = rear = NULL
c) else (front = front → next)

& free (temp)



Scanned with CamScanner

- Explain circuit.

11 Monday March 2013

Run time Calculation

- Efficiency of algo can be decided by measuring the performance of algo.
- we can measure the performance of an algorithm by counting two factors:

① Amt. of time required by the algo to execute :- Time Complexity

② Amt of space (memory) required by algorithm to execute :- Space Complexity

Performance of the program is measured in terms of frequency count.

~~General Rules:- the~~

The frequency count is a count that denotes how many ~~times~~ times perticular statement is executed.

Scanned with CamScanner

12 Tuesday March 2013

Rule 1 :- Simple assignment statement

```
void fun()
{ int a;
    a=10;          —①
    printf("%d", a)  —①
}
```

FC = 2

Rule 2 :- For loops

$\frac{n^2+n+3}{2}$

$O(n^2)$

```
void fun()
{
    int a;
    a=0;                        — 1
  for(int i=0 ; i<n ; i++)      — n+1
  {
    a = a+i ;                   — n
  }
  printf("%d", a);              — 1
}
```

FC = $1+(n+1)+n=1$

$= 2n+3$

$O(n) = TC$

Scanned with CamScanner

Rule 3 :- Nested for loops

```
void fun (int a[ ][ ], int b[ ][ ])
{
  int c[3][3];
  for (i=0 ; i<m ; i++)          ---- m+1
  {
    for (j=0 ; j<n ; j++)        m.(n+1)
    {
      c[i][j] = a[i][j] + b[i][j]   m.n
    }
  }
}
```

$FC = (m+1) + [m(n+1)] + m \cdot n$

$= m+1 + mn + m + mn$

$= 2m + 2mn + 1$

```
void add(A, B, n)
{ for (i=0 ; i<n ; i++)
  { for (j=0 ; j<n ; j++)
    { c[i][j] =
```

$2n^2 + 2n + 1$

Scanned with CamScanner

queue with eg. or diagram (sh.... 1..

14 Thursday March 2013

```
for (i=1 ; i<=n ; i++) ....n+1
{
  for (j=1 ; j<=n ; j++)... n (n+1)
  {
    ~~c[i][j] = a[i][j] + b[i][j]~~
    c[i][j] = 0;          .... n.n
    for (k=1 ; i<=n ; k++)..n.n.(n-1)
      c[i][j] = c[i][j] + a[i][k] * b[k][j];
  }                        .... n.n.n
}
```

$PC = (n+1) + [n(n+1)] + n^2 + n^2(n+1) + n^3$

$= n + 1 + n^2 + n + n^2 + n^3 + n^2 + n^3$

$= 2n^3 + 3n^2 + 2n + 1$

Big Oh notation is a most commonly used algorithmic notation.

For above PC all the constant terms are neglected & only the highest order polynomial is always considered

Hence the time complexity is $O(n^3)$

Scanned with CamScanner

Rule 4 :- If-else statement

```
if (a<10)                          --- (1)
Pf("a is less than 10");           --- (1)
else
Pf("a is greater than 10");        --- (1)
```

FC 2 or 2.

eg (1)

```
m = n/2                            --- 1
for(i=0; i+m<n; i++)               --- n/2 + 1
{
    a++;                           --- n/2
    K++;                           --- n/2
}
```

$1 + \frac{n}{2} + 1 + \frac{n}{2} + \frac{n}{2}$

$FC = \frac{3n}{2} + 2$ //

$O(n/2)$

Scanned with CamScanner

Imp

eg for (i=1; i<=n ; i++) ... n+1

practice for (j=1; j<=n ; j++) n·(n+1)

x=x+1 n·n

$FC = n+1 + n^2+n+n^2$

$= 2n^2+2n+1$

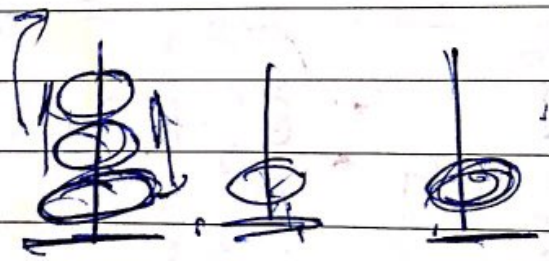
$TC = O(n^2)$

(5) Logarithmic complexity

```
for(i=1; i<=n;)
{
  ---
  ---
  ---
  i = i*2;
}
```

$\log_2 n$

$O(\log n)$

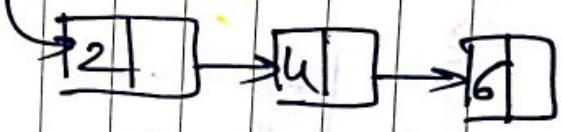


$O(1) < O(\log) < O(n) < O(n \log n) < O(n^2)$

Scanned with CamScanner

18 Monday March 2013

Arrays Vs LL

Factors	Arrays	Linked List
cost of accessing an element:	It takes constant amt. of time array is stored as one contiguous block of memory. If we know the starting address of the block of memory add^s of element with index i $= 200 + i \times 4$ if $i = 4$ $= 200 + 4 \times 4$ $= 200 + 16$ $= 216$ TC $= O(1)$	If data is not stored in contiguous block of memory. Only address that we keep with us is the address of head node. head.  Inserting at:- • head - $O(1)$ • Tail - $O(n)$ • middle → $O(n)$

Scanned with CamScanner

19 Tuesday March 2013

Factors	Arrays	Linked List
Memory Requirement	• Fixed size • unused memory 0 1 2 3 4 5 6 2 4 6 - - - - 7×4 = 28 bytes	• No unused memory • Extra memory for pointer head → 2 → 4 → 6 8 8 8 8×3 = 24
cost of Inserting element	At begining - TC O(n) At end - TC O(1) if array is not full - TC O(n) if array is full At middle - TC O(n/2)	At beg - O(1) at tail - O(n) at middle O(n/2)
~~Cost~~		

Scanned with CamScanner

March 2013

20 Wednesday

Factors	Arrays	Linked List
Ease of use	Easy	Difficult
Cost of deleting	at beg - $O(n)$ at end - $O(1)$ middle - $O(n/2)$	$O(1)$ $O(n)$ $O(n/2)$

Scanned with CamScanner

Algorithm :-

```
Merge(L, R, A)
{
 nL ← Length(L)
 nR ← Length(R)
  i ← j ← k ← 0

while(i < nL && j < nR)
{
 if( L[i] <= R[j])
 {
   A[k] ← L[i]
   k ← k+1
   i ← i+1
 }
else
 { A[k] ← R[j]
   k ← k+1
   j ← j+1
 }
}
```

Scanned with CamScanner

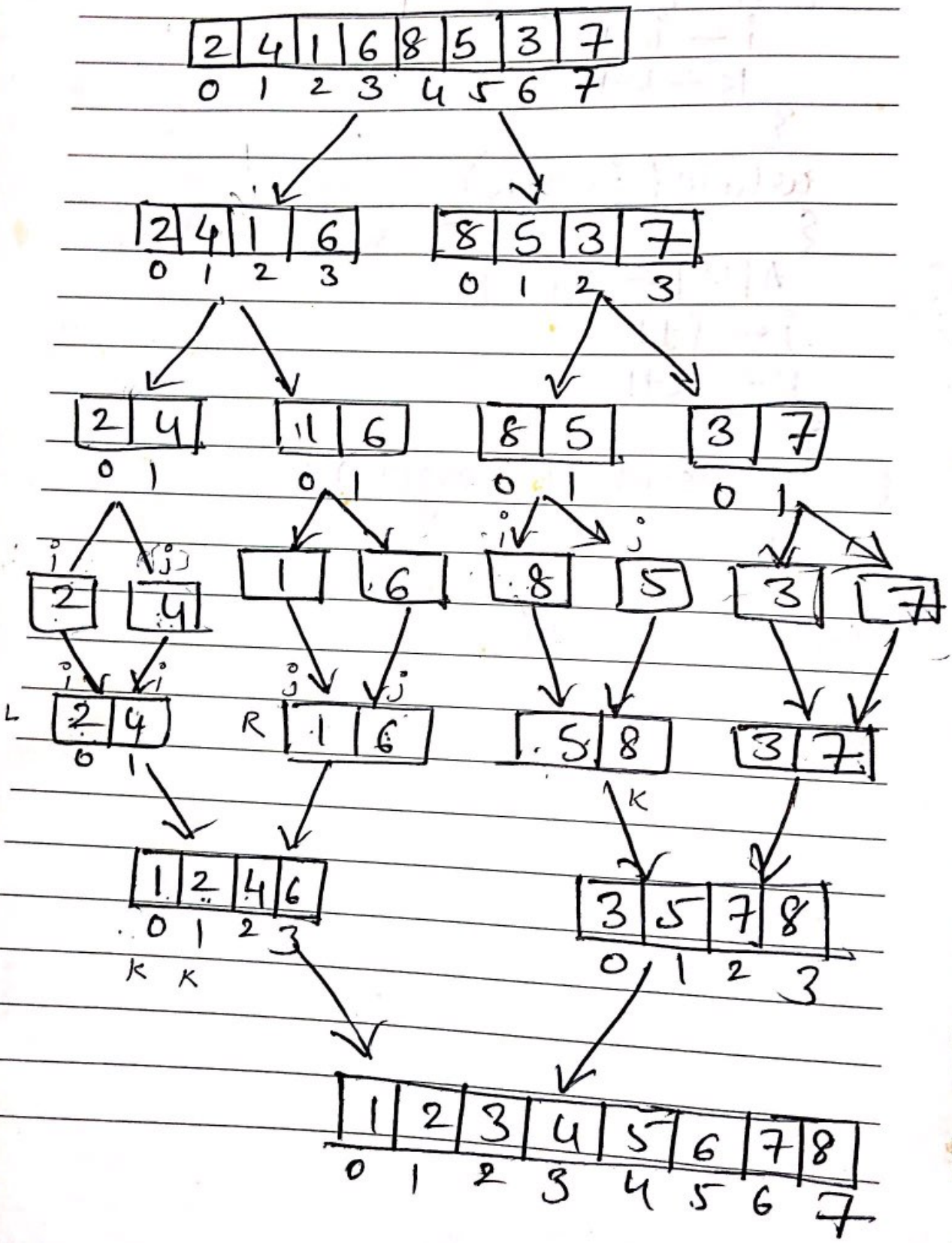
23 Saturday

```
while (i < nL)
{ A[K] ← L[i]
    i ← i+1
    K ← K+1
}
while (j < nR)
{
  A[K] ← R[j]
  j ← j+1
  K ← K+1
}
}     //end of Merge()
```

24 Sunday

Scanned with CamScanner

Merge Sort :-



Scanned with CamScanner

```
MergeSort(A)
{ n ← length(A)
  if (n<2)
    return
  mid ← n/2
  left ← array of size(mid)
  right ← array of size (n-mid)
  for i ← 0 to mid-1
      left[i] ← A[i]
  for i ← mid to n-1)
      right[i-mid] ← A[i]

  MergeSort(left)
  MergeSort(right)
  Merge(left, right, A)

}.
```

```
for i=0 to 3
left[0-3] ← A[0-3]
for i=4 to i=7
right[0-3] ← A[4-7]
```

n = 8
mid = 4
left = 4
right = 4

Scanned with CamScanner

Program:-

```
# Include <stdio.h>
# Include <stdlib.h>

void Mergesort(int *a, int length);
void Merge(int *a, int *L, int *R, int length);

void main()
{
 int a[7] = {99, 36, 55, 2, 1, 27, 92};
 int i=0;
 Mergesort(a, 7)

for(i=0; i<7 ; i++)
{
 pf("%d\t", a[i]);
}

}
```

Scanned with CamScanner

queue with eg. or diagram (sh... l...

March 2013

28 Thursday

```
void Mergesort(int *a, int length)
{
  int mid = 0; int *L, *R; int i=0;
  mid = length/2;

// exit cond^n for recursion
  if (length < 2)
    return;
else
  { L = (int*) malloc(sizeof(int)*mid);
    R = (int*) malloc(sizeof(int)*(length-mid));

// this code is to create left & right child arrays

// this code is to copy contents into childs

for (i=0; i< mid; i++)
{ L[i] = a[i]; }
for (i=mid; i< length; i++)
{ R[i-mid] = a[i]; }

Mergesort(L, mid);
Mergesort(R, length-mid);
Merge(a, L, R, length);  // start merge the children
}
```

Scanned with CamScanner

```
void Merge(int *a, int *L, int *R, int length)
{
int i=0, j=0, k=0;
int mL, mR;
mL = length/2;
mR = length - mL

while (i<mL && j<mR)
{
  if(L[i] < R[j])
  {  a[k] = L[i];
     i++;
  }
  else
  {
    a[k] = R[j];
     j++
  }
  k++;
}
while(i< mL)
{
  a[k] = L[i]
   i++;
}
```

```
while(j< mR)
{
a[k] = R[j];
  j++;
  k++;
}
}
```

Scanned with CamScanner

```
Quicksort(A, start, end)
{
    if (start < end)
    {
      pIndex ← partition (A, start, end)
      Quicksort(A, start, pIndex - 1)
      Quicksort(A, pIndex+1, end)
    }
}
```

pivot

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

PI

pivot

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

PI

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

PI

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

PI

Scanned with CamScanner

```
Partition (A, start, end)
{
  pivot ← A[end]
  pIndex ← start
  for i ← start to end-1
  {
    if (A[i] <= pivot)
    {
      swap (A[i], A[pIndex])
      pIndex ← pIndex+1
    }
  }
  swap (A[pIndex], A[end])
  return pIndex
```

Scanned with CamScanner

06 Saturday April 2013

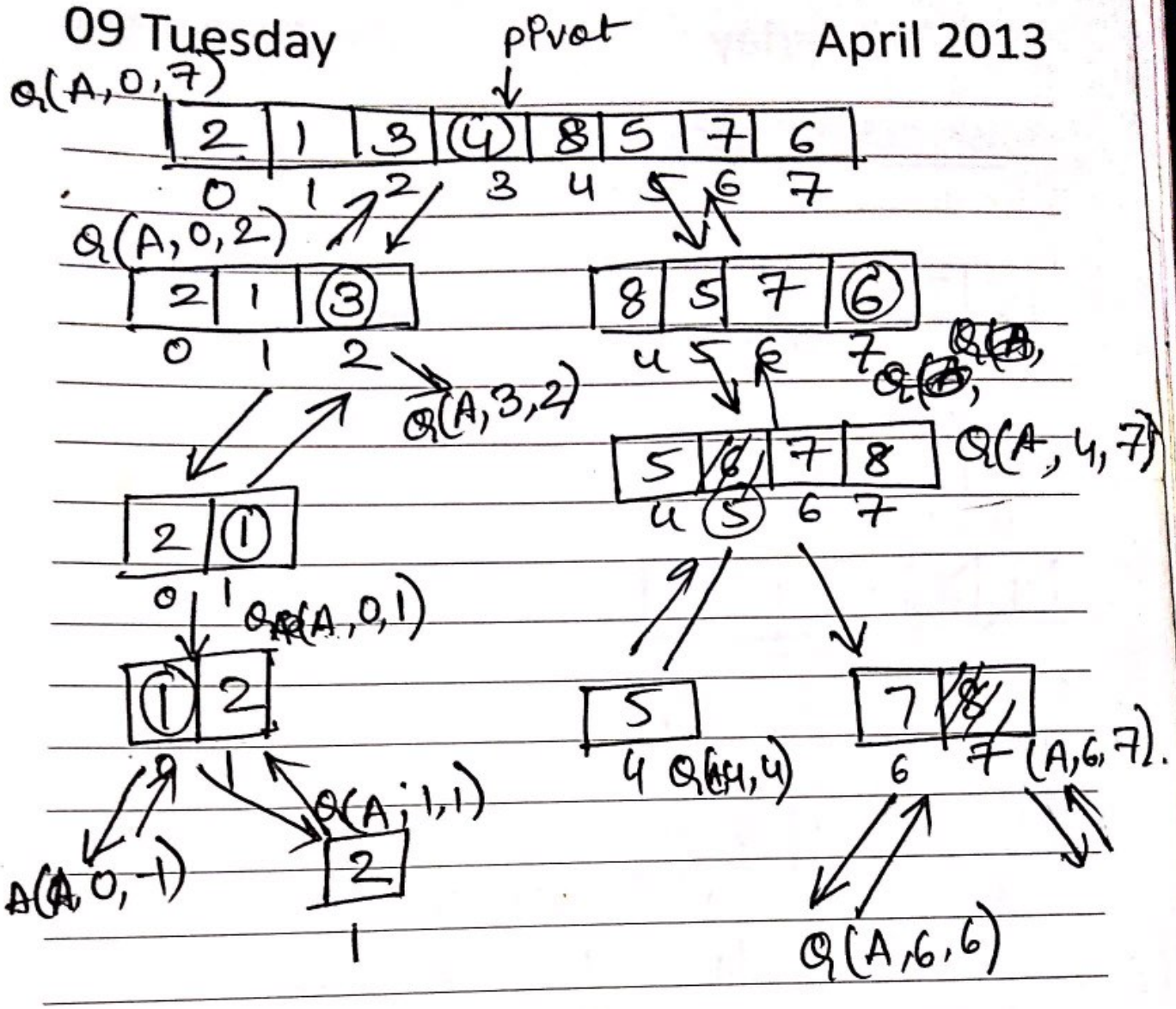
Program :-

```
int partition(int *A[], int start, int end)
{ int pivot = A[end];
  int ~~partition~~ pIndex = start
  for(int i = start ; i<end ; i++)
    {
      if (A[i] <= pivot)
        {
          swap(A[i], A[pIndex]);
          pIndex++;
        }
    }
  swap(A[pIndex], A[end]);
  return pIndex;
}
void QuickSort(int A[], int start, int end)
{ if(start < end)
    {
      int pIndex = partition(A, start, end);
      Quicksort(A, start pIndex - 1);
      Quicksort(A, pIndex+1, end);
    }
}
```

07 Sunday

Scanned with CamScanner

09 Tuesday April 2013



Scanned with CamScanner

Selection Sort

2	7	4	1	5	3
0	1	2	3	4	5

⇒

1	7	4	2	5	3
0	1	2	3	4	5

~~1~~	~~2~~	4	7	5	3

1	2	3	7	5	4

1	2	3	4	5	7

0	1	1	2	3	4
2	7	4	1	5	3
i iMin			iMin		

```
SelectionSort(A,n)
{ for i ← 0 to n-2
{ iMin ← i
  for j ← i+1 to n-1
  {
    if (A[j] < A[iMin])
      iMin ← j
  }
  Swap(A[i], A[iMin])
}
```

Scanned with CamScanner

```
void selection(int A[], int n)
{
for(int i=0 ; i<n-1 ; i++)
{
  int iMin = i
  for( int j = i+1 ; j<n ; j++)
  {
    if(A[j] < A[iMin])
      iMin = j
  }
  int temp = A[i];
  A[i] = A[iMin];
  A[iMin] = temp;
}
}
```

Scanned with CamScanner

Bubble Sort

4 1 7 5 3 7

2 7 4 1 5 3

↑ ↑ ↑ ↑ ↑

~~repeat~~ 1 4 3 5

⇒ 2 4 1 5 3 7

↑ ↑ ↑ ↑

1 2 3 4

⇒ 2 1 4 3 5 7

↑ ↑ ↑ ↑

= 1 2 3 4 5 7

```
BubbleSort(A, n)
{ for k ← 1 to n-1
  {
    for i ← 0 to n-2
    {
      if (A[i] > A[i+1])
      {
      swap(A[i], A[i+1])
      }
    }
  }
}
```

Scanned with CamScanner

13 Saturday April 2013

```
Insertion Sort(A, n)
{
 for i ← 1 to to n-1
 {
  value ← A[i]
  hole ← i
 while (hole > 0 && A[hole - 1] > value)
 {
  A[hole] ← A[hole - 1]
   hole ← hole - 1
 }
 A[hole] ← value
 }
}
```

14 Sunday

Scanned with CamScanner

7	~~2~~	4	1	5	3
0	1	2	3	4	5

value = 2

2 7 | ~~4~~ 1 5 3

value = 4

2 ~~4~~ 7 | 1 5 3

value = 1

1 2 4 7 | ~~8~~ 3

value = 5

1 2 4 5 7 | 3

value = 3.

1 2 3 4 5 7

Scanned with CamScanner

Searching Techniques

~~1) Linear Search :-~~

- when we store & organize data, there are various operations that are performed in it.
- Searching is most frequent performed operation
- It ~~is~~ should be efficient & fast.

Types :-

① Linear Search (Sequential search)

In this, search is done one by one from top to bottom.

② Binary search.

- It is done on sorted data.
- find the middle element
- choose the data to be searched.
- If the data is lesser go to the left side else go to right.

Scanned with CamScanner

mid
↓
2 6 7 9 10 12 15

No. to be searched. = 7

$\boxed{7<9}$ → go left & scan each element

~~Bin search(~~
void main()
{
int A[] = {3, 11, 15, 16, 19, 20, 22}
int item = 20;
bin_search (A, 7, item);
}

void bin_search(int L[], int N,
int item)
{
int l=0, u=N-1;
while (l<=u)
{
m=(l+u)/2
if (item==L[m])

Scanned with CamScanner

```
{ pf("Success")
    return;
}
else if (item>L[m])
    l = m+1
else
    u = m-1;
}
pf("Unsuccessfull")
}.
```

<u>Searching : Time complexity</u>

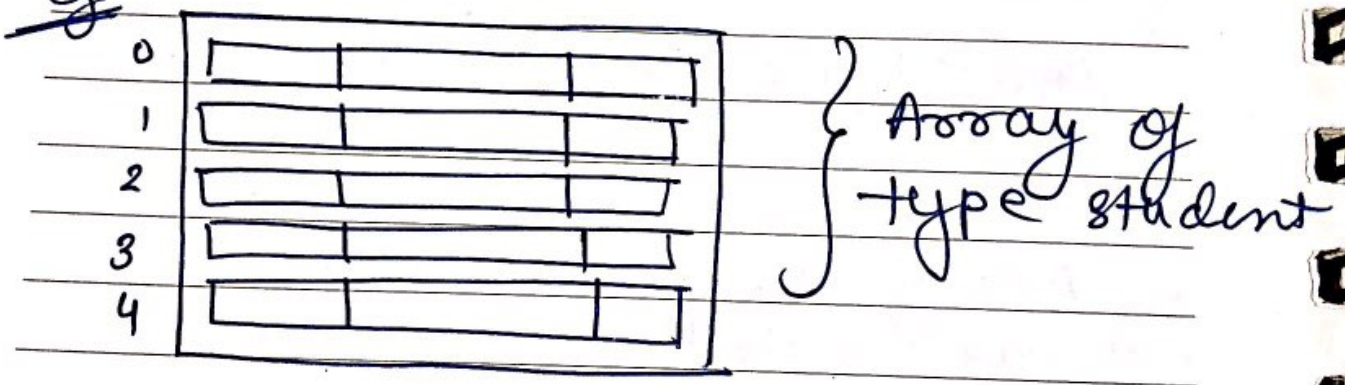
- Linked List - $O(n)$
- Array (unsorted.) - $O(n)$
- Array (sorted) - $O(\log_2 n)$
- BST - $O(\log_2 n)$
- Hash Table - $O(1)$

Scanned with CamScanner

Hash Table — custom array

```
struct student
{ int rollno;
  char name[20];
  int age;
};

student s1;
student s[5];
```

eg



$\boxed{\text{candidate NO} = \text{Index NO}}$

Give the candidate NO & in constant time you can search student info".

Scanned with CamScanner

20 Saturday April 2013

eg 2 complaint System.

Complaint key [0 - - - 9]
m = spaces = 10
n = data = 5
key = bigger Number.

$n \leq m \leq$ key.

In case when keys are large & can't be directly used as index. then we have to map them.

21 Sunday CN (Key) = 1000 to 9999 ✓

CN
1081
1523
3446
7189
5524

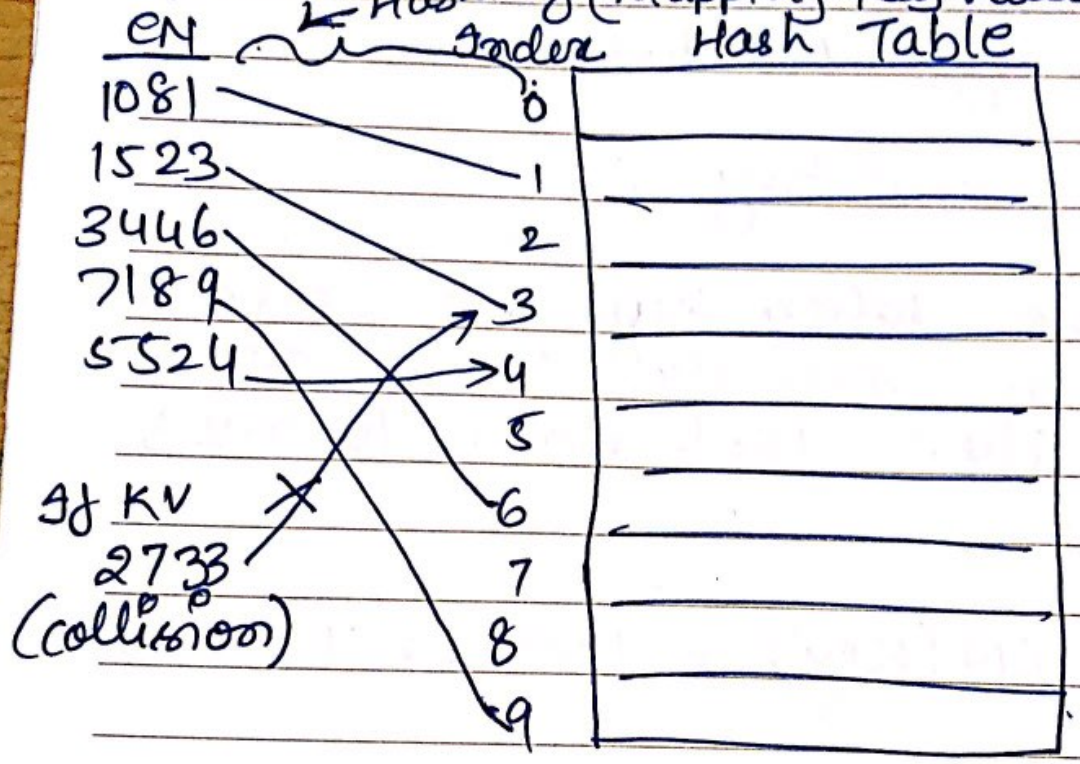
Scanned with CamScanner

22 Monday ⇒ any formula. April 2013

Hash function (CN) ⇒ Index

assume $\boxed{HF = CN \% 10}$

key value ← Hashing (Mapping key value with index)



- To avoid collision we should create Hash function better & efficient

Scanned with CamScanner

1) Hash Table :-
- Mostly it is an array to store dataset. It is the data structure.

2) Hash Function :-
It is funⁿ that can be used to map dataset of arbitrary size to dataset of fixed size which falls into Hash table

3) Hashing :-
large keys are converted into small ones by using hash funⁿ & then the values are stored in data structure called hash tables.

Scanned with CamScanner

22 Wednesday April 2013

Hash function :-

- A hash function usually means a function that compresses, meaning the output is shorter than the input.

- A hash function is any function that can be used to map data of arbitrary size to data of fixed size.

- Parameters of good function :-
  1) Easy to compute
  2) Even distribution
  3) Minimize collision

- Perfect hashing maps each valid input to a different hash value (no collision).

Scanned with CamScanner

Collision Resolution :-

- Hash table is a Data structure where the data elements are stored (inserted), searched, deleted based on the keys generated for each element, which is obtained from a hashing function

- A perfectly implemented hash table would always promise an avg insert/ delete/ retrival time of O(1)

- A situation when the resultant hashes for two or more data elements in the data sets, maps to the same location in the has table, is called callision.

- In such situation two or more data elements would qualify to be stored/ mapped to the same location in the hash table.

Scanned with CamScanner

26 Friday April 2013

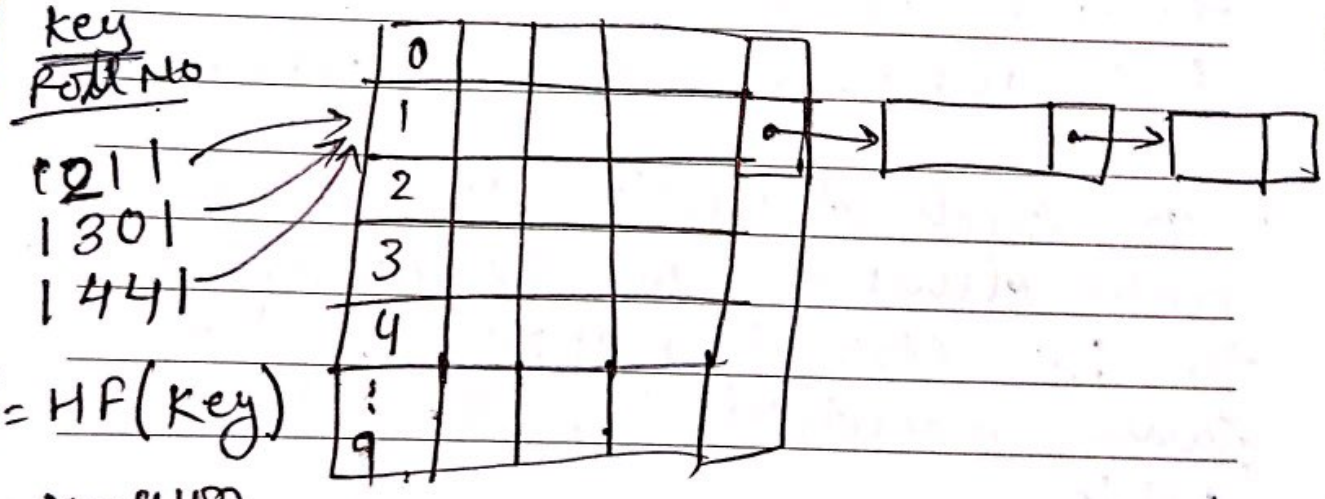
Hash Collision Resolution Techniques :-

1. Open Hashing (chaining)
2. Closed Hashing (Open Addressing)

Open Hashing :-
When collisions are stored outside the table

Closed Hashing :-
when collisions result in storing one of the records at another slot in the table (closed hashing).

Open Hashing



i = HF(Key)

HF = Key % 100

- It defines each slot in the hash table to be the head of linked list.

- All the records that hash to a particular slot are placed on that slot's linked list.

(2) Closed Hashing

(1) Linear Probing

$i+1$
$i+2$
$i+3$

28 Sunday

(2) Quadratic Probing

$i+1$
$i+4^{2^2}$
$i+9^{3^2}$

(3) ~~i+5~~ Double Hashing

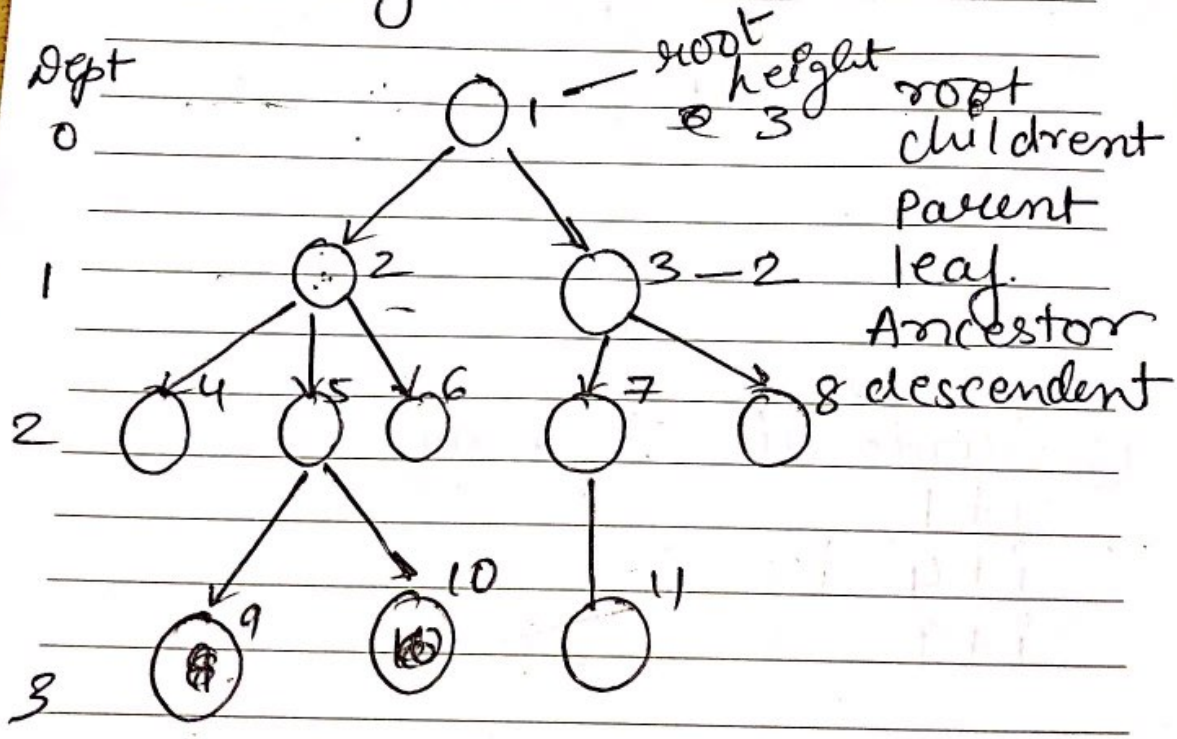
$i+5$
$i+2*5$
$i+3*5$
$i+4*5$

Scanned with CamScanner

Trees

- Non Linear Data structure
- Used to store hirarchical data.
- Definition :-

Collection of nodes linked together to represent a hirarchy.



Scanned with CamScanner

leaf – has no child.

Tree – recursive data structure
- → N nodes
- → N-1 edges.

} valid tree.

Depth & Height

Depth of node x = length of path from root to x
OR
No. of edges from root to x

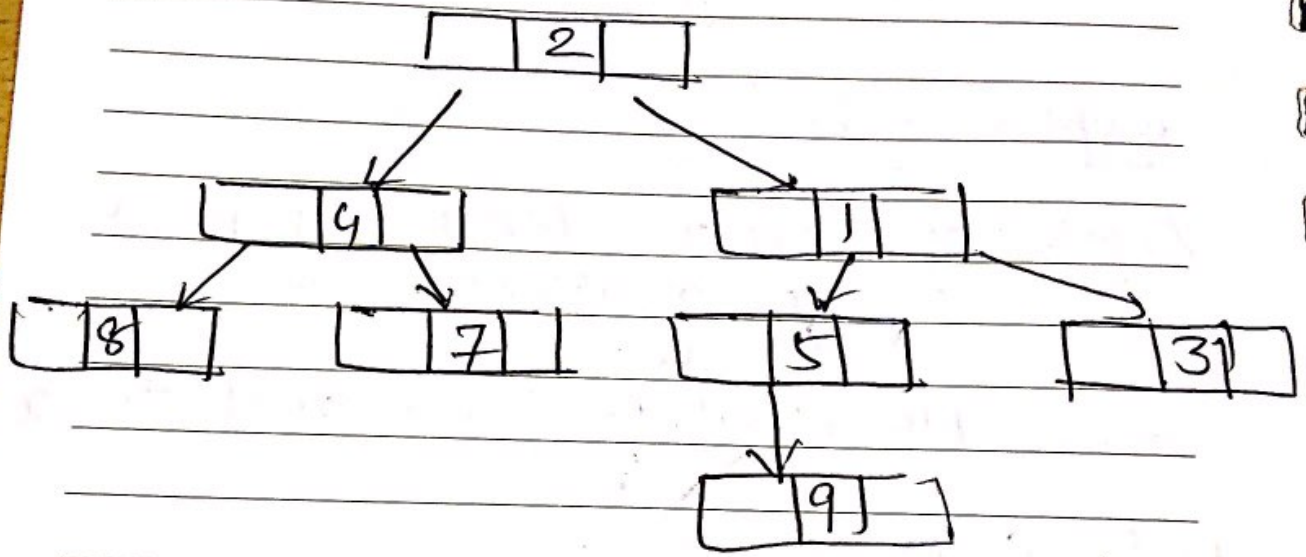
Height of node x =
No. of edges in longest path from x to a leaf.

Height of tree = height of root node

Scanned with CamScanner

01 Wednesday May 2013

- Binary Tree:-
  A tree in which each node can have at most (max) 2 children.




```
Struct Node
{
 int data;
 Node *right;
 Node * left;
};
```

Scanned with CamScanner

02 Thursday May 2

Applications :-

1) Storing naturally hierarchical data →
   eg :- file System

2) Organize data
   → quick search, insertion, deletion.

3) Trie → dictionary
   Dynamic Spell checking.

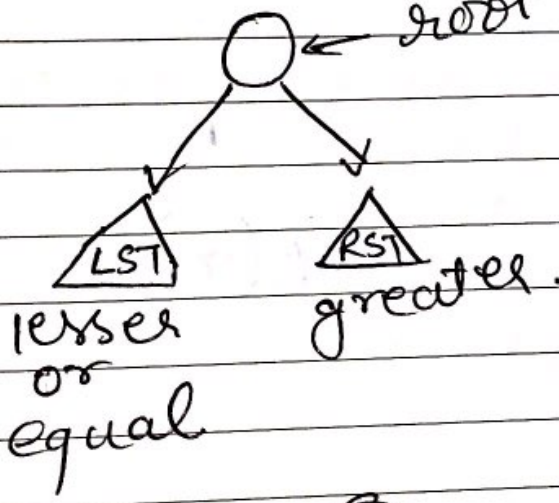
4) Network Routing Algorithm.

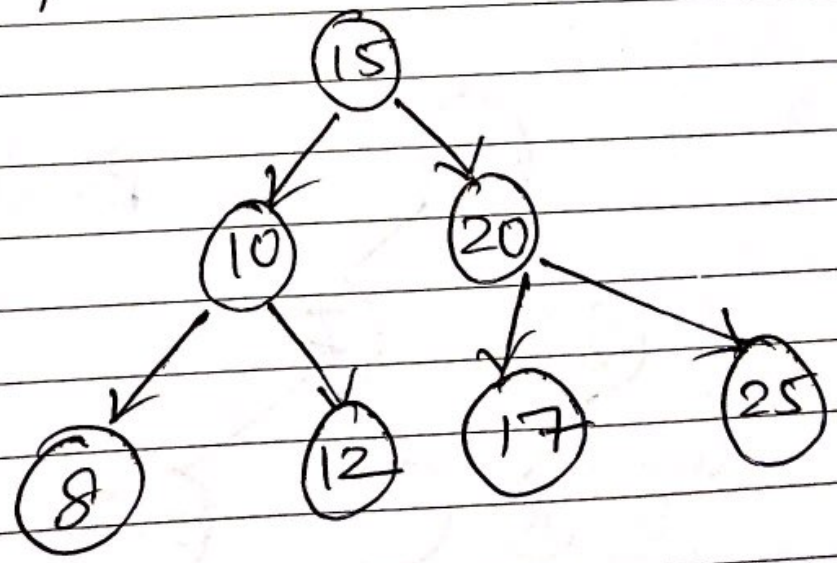
Scanned with CamScanner

07 Tuesday May 2013

Binary Search Tree (BST)

A binary search tree in which for each node, value of all the nodes in left subtree is lesser or equal and values of all the nodes in right subtree is greater.





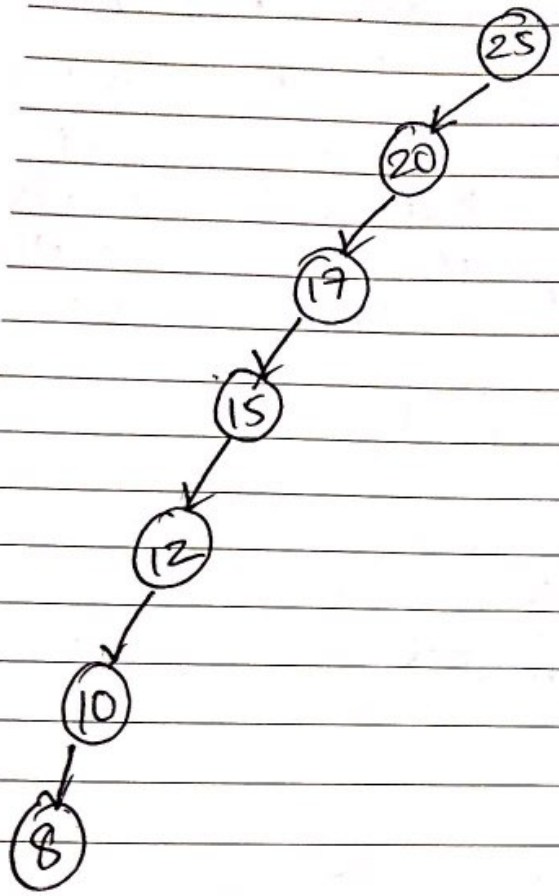
$n \rightarrow n/2 \rightarrow n/4 \rightarrow \cdots \rightarrow 1$

$O(\log_2 n)$

Scanned with CamScanner

08 Wednesday May 2013

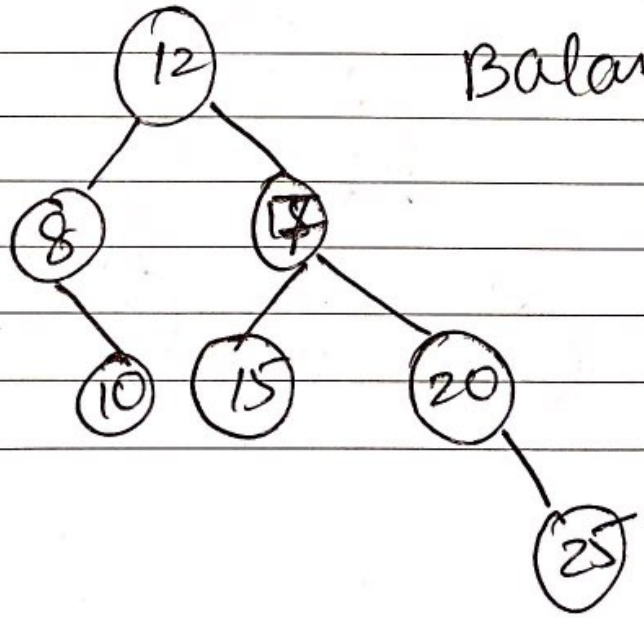
Unbalance BST



$n \rightarrow n-1 \rightarrow n-2 \rightarrow \vdots \rightarrow 1$

$O(n)$

Balanced



Scanned with CamScanner

09 Thursday May 2013

Implementation using Linked List

```
Struct Node
{ int data;
  Struct Node * left;
  Struct Node * right;
};
int main()
{
Struct Node * root = NULL;
root = Insert (root, 15);
root = Insert (root, 10);
root = Insert (root, 20);
root = Insert (root, 25);
}

Struct Node * Insert(struct Node *root,
                     int data)
{ if (root == NULL)
{ root =(struct Node*) malloc (sizeof
                (struct Node*));
  root -> data = data;
  root -> left = root -> right = NULL;
}
```

Scanned with CamScanner

```
else if (data <= root -> data)
  root -> left = Insert(root -> left, data);
else
  root -> right = Insert(root -> right, data);

return root;
}

int search(struct Node *root, int data)
{
  if (root == NULL)
  return 0;
else if (root -> data == data)
  return 1;
else if (data <= root -> data)
  return search(root -> left, data);
else
  return search (root -> right, data);
}
```

Scanned with CamScanner

11 Saturday May 2013

Binary Tree Traversal :-

→ Breadth first-

F, D, J, B, E, G, K, A, C, I, H

→ Depth first:-

<root> <left> <right> — preorder

<left> <root> <right> — inorder

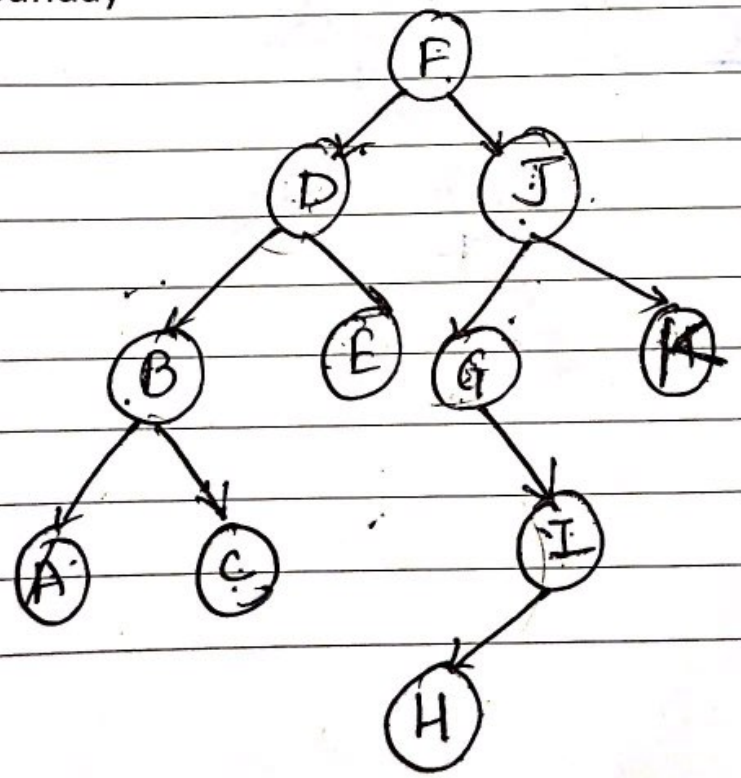
<left> <right> <root> — postorder

preorder:- F, D, B, A, C, E, J, G, I, H, K

inorder:- A, B, C, D, E, F, G, H, I, J, K,

Postorder:- A, C, B, E, D, H, I, G, K, J, F

12 Sunday



L-0

L-1

L-2

L-3

L-4

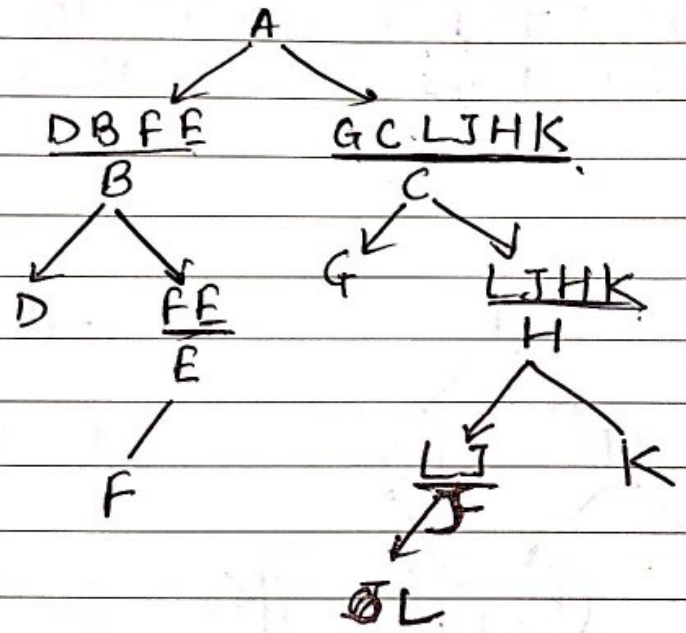
Scanned with CamScanner

13 Monday May 2013

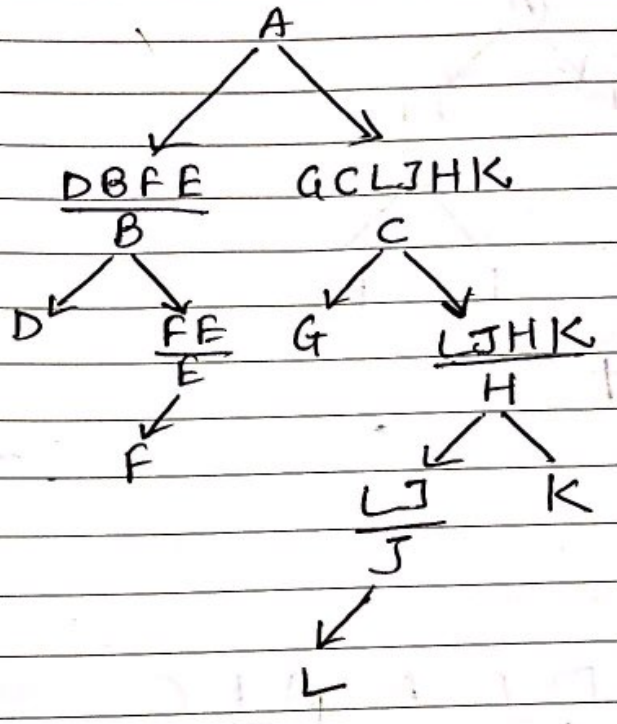
Construct Binary Search Tree using preorder, Inorder & postorder.

Preorder :- Root L R
Inorder :- L Root Right
Postorder :- L R Root

① Preorder :- A B D E F C G H J L K
Inorder :- D B F E A G C L J H K



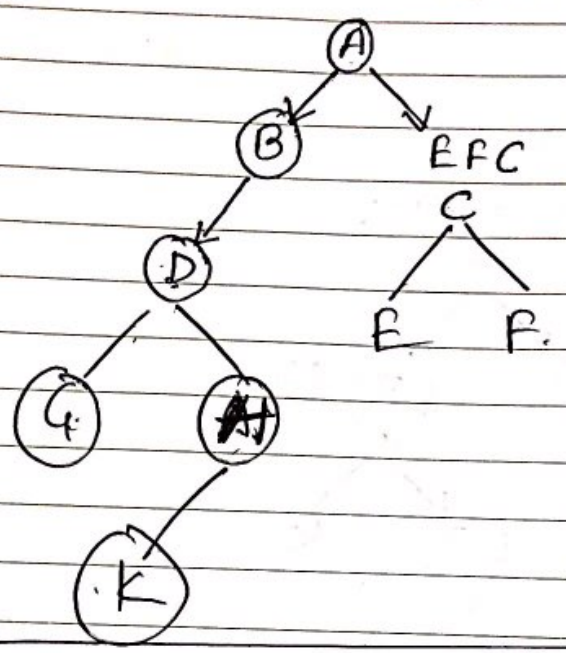
② Postorder : D F E B G L J K H C A
Inorder : D B F E A G C L J H K



Scanned with CamScanner

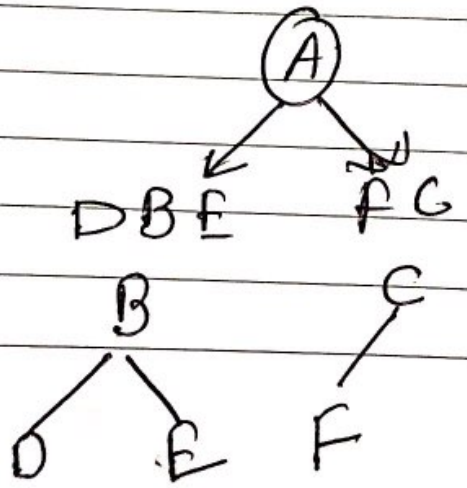
15 Wednesday May 2013

(3) Preorder :- A B D G H K C E F
Postorder :- ~~G H~~ G K H D B E F C A } Tricky



Inorder :- D B E A F C

Preorder :- A B D E C F



Scanned with CamScanner

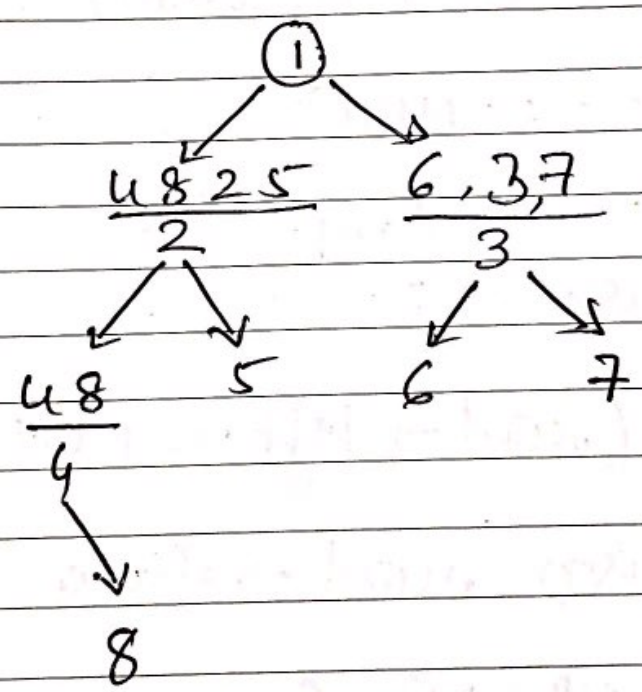
eg

① Preorder : ~~ABCD~~ A B D E F C G H J K

Inorder : D B F E A G C L J H K

Inorder :- 4, 8, 2, 5, 1, 6, 3, 7

postorder :- 8, 4, 5, 2, 6, 7, 3, 1



Scanned with CamScanner

17 Friday

May 2013

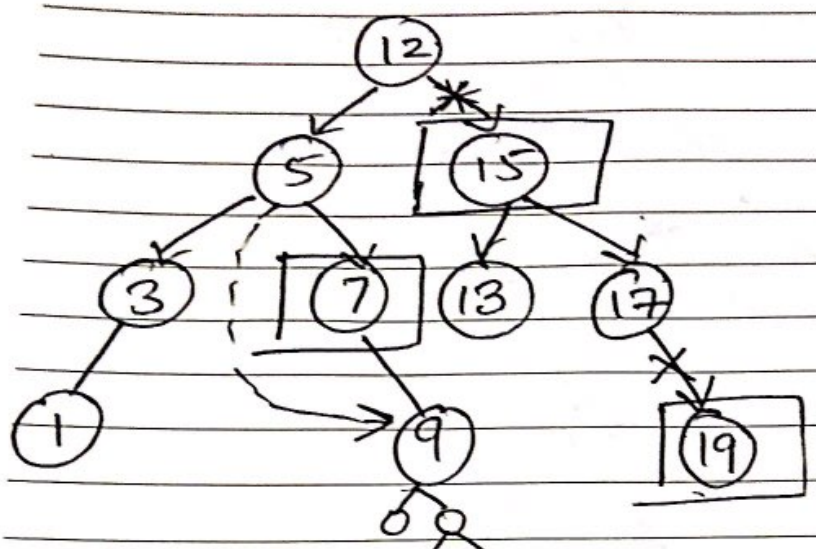
Find min & max element in BST

```
struct BstNode
{ int data;
  BstNode * left;
  BstNode * right;
};

int FindMin(BstNode* root)
{
  if (root == NULL)
  {
    pf("Tree Empty");
    return -1;
  }
else if (root -> left == NULL)
  {
    return root -> data
  }
return FindMin(root -> left);
}
```

Scanned with CamScanner

18 Saturday May 20_

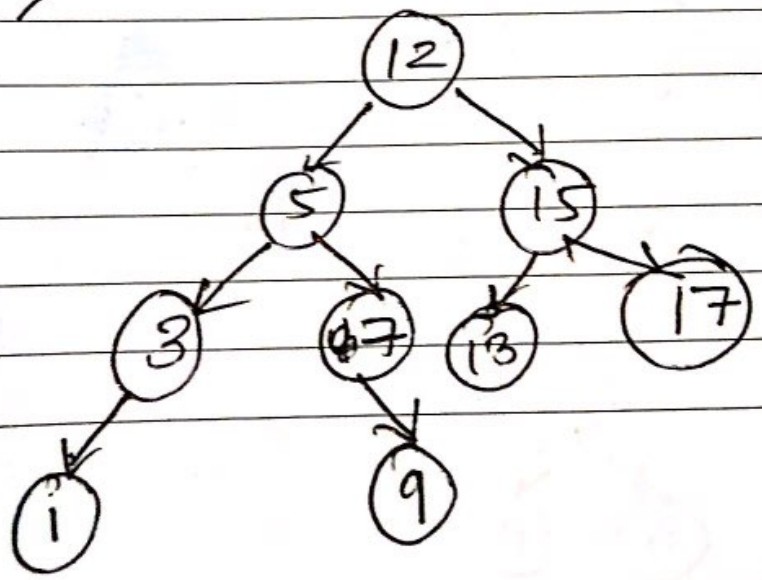
Delete a node from BST



(1) – Remove the reference (link)
(2) – free the node from memory

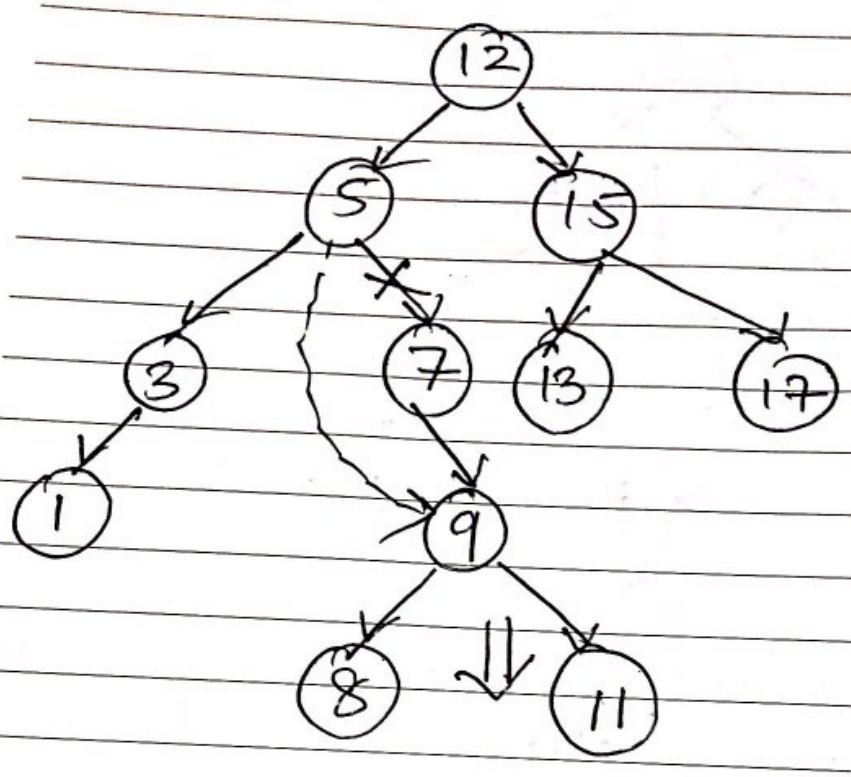
19 Sunday

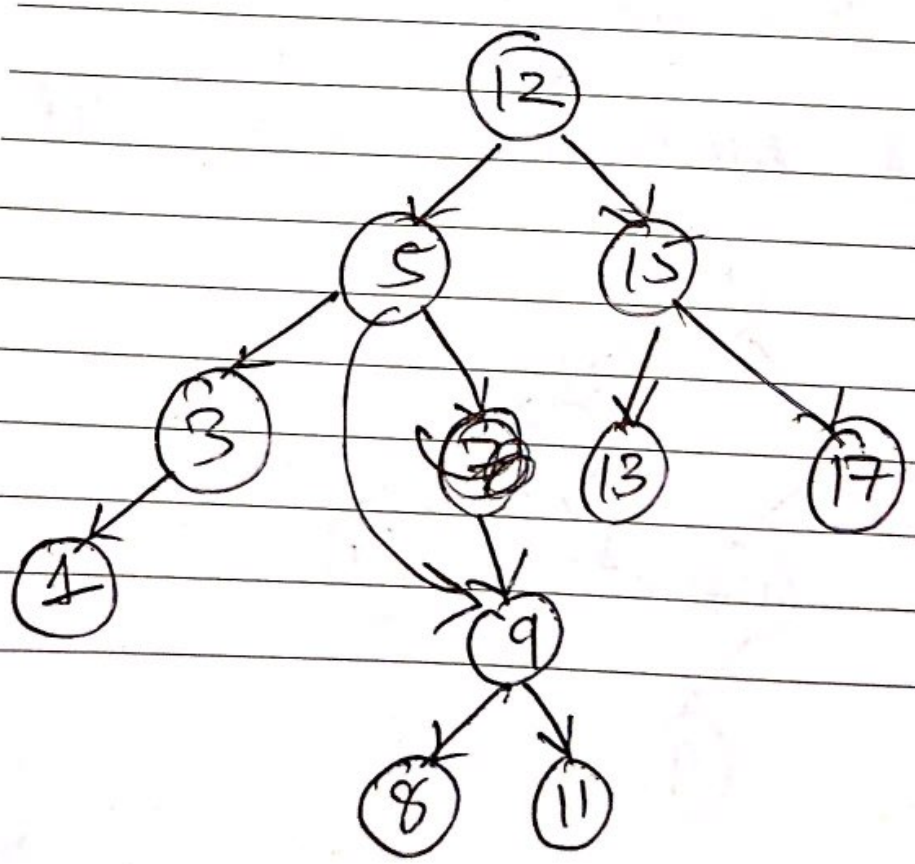
Case 1 Node with No child.



Scanned with CamScanner

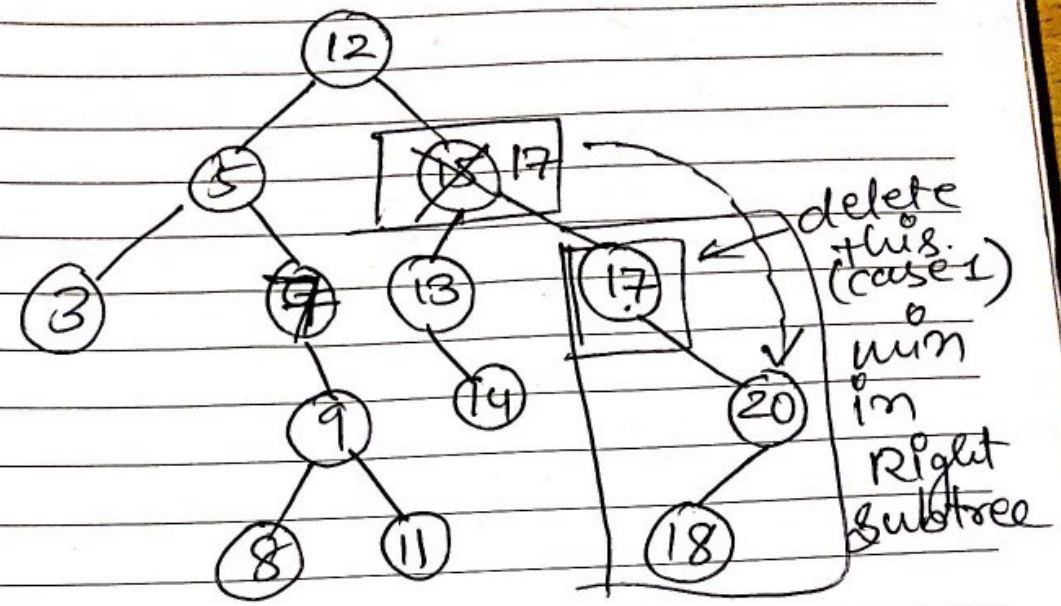
Case 2 Node with one child.





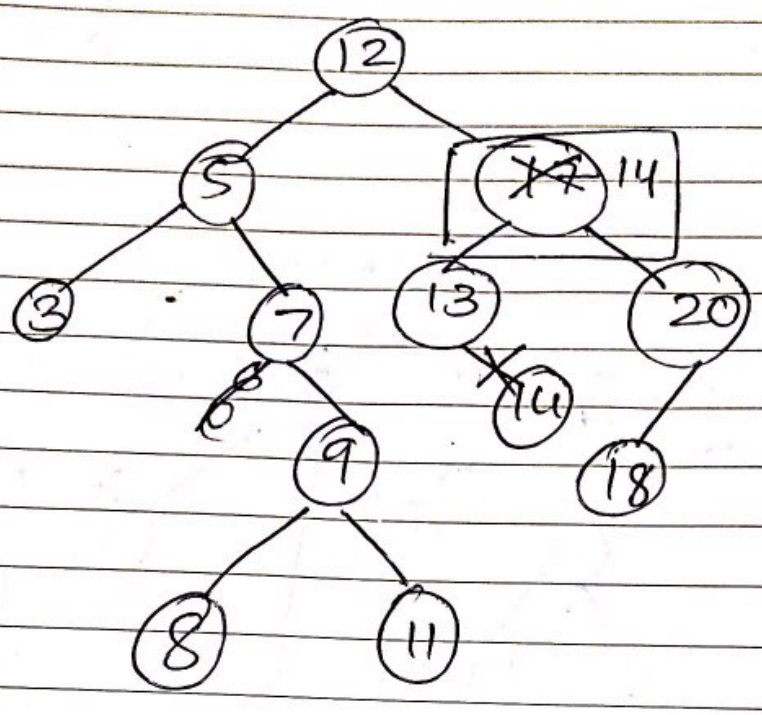
Scanned with CamScanner

case 3 : Node with 2 children



- Reduce to case 2 or case 1

1. Min of RST and copy in root → data.
2. We now have two node of 17
3. 17 in RST has only one child so its easy to delete (case 2)
4. Min from RST will have all node LST lesser & RST will have greater & equal values
5. If a subtree having min value, it won have a left child



BST after deleting node 15

① choose max from LST & copy ~~with~~ in (root → data)

② Delete the duplicate Node

Scanned with CamScanner

23 Thursday May 2

Case 3: 2 children

① Find min in RST	① Find max in LST
② Copy the value in targetted node	② copy the value in the targetted ~~value~~ node
③ Delete the duplicate from RST	③ Delete duplicate from left subtree.

Scanned with CamScanner

24 Friday May 2013

Delete a Node in BST

```
Struct Node* Delete (struct Node *root,
                     int data)
{
    if(root == NULL)
    return root;

    else if (data < root -> data)
    root -> left = Delete(root -> left, data)

    else if (data > root -> data)
    root -> right = Delete(root -> right, data);

// Node is found

    else
{
    // case 1: No Child

    if(root -> left == NULL &&
                root -> right == NULL)
    {
        free(root);
        root = NULL;
    }
```

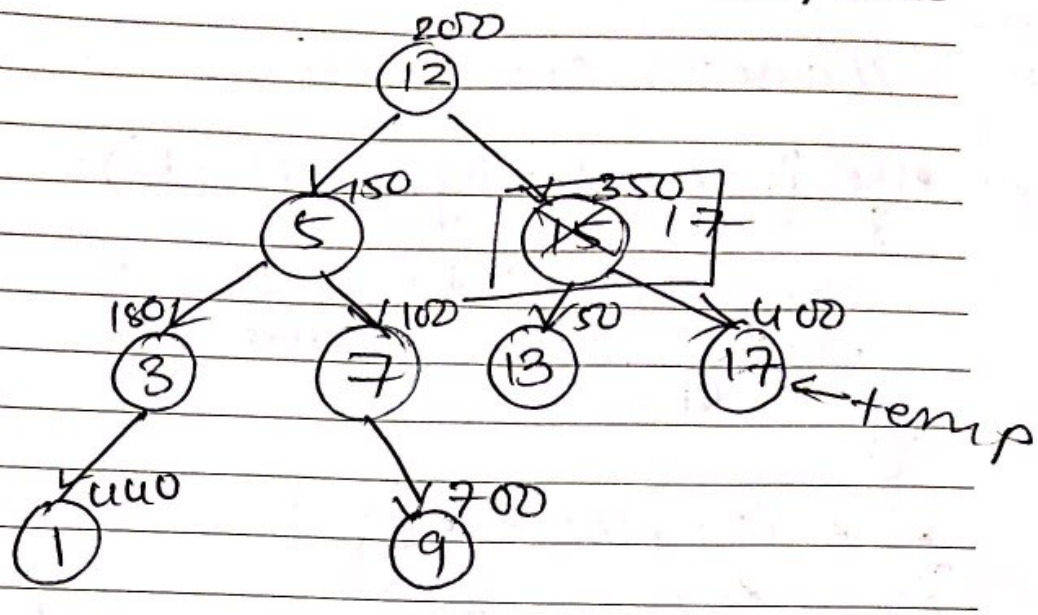
25 Saturday May 2013

```
// case 2: One child

else if(root->left == NULL)
{
  struct Node* temp = root
  root = root->right;
  free(temp);
}
else if(root->right == NULL)
{
  struct Node *temp = root;
  root = root->left;
  free(temp);
}
```

26 Sunday

```
// case 3: 2 children

else
{
  struct Node *temp = findMin(root->right);
  root->data = temp->data;
  root->right = Delete(root->right,
                       temp->data);
}
}.
```

Scanned with CamScanner



main()

Delete(200, 15)

↓ ↑ 200

200→right = Delete(350, 15)

~~temp~~ ↓ ↑ 350

350→right = Delete(400, 17)

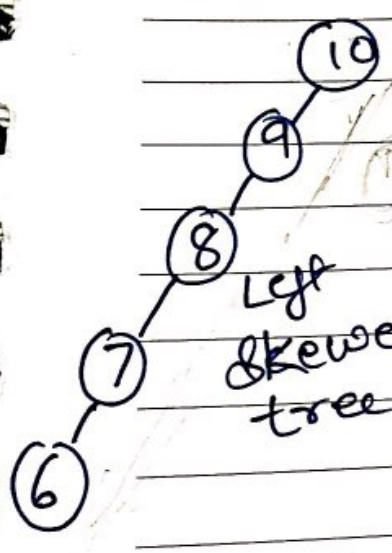
=NULL

Scanned with CamScanner

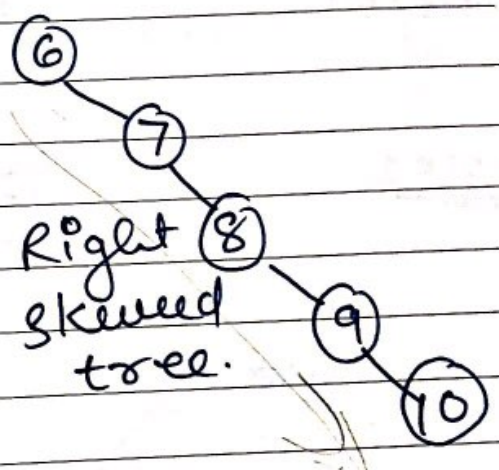
AVL Trees

- BST are always unbalanced
- when unbalance the operations on it might take more than ($\log_2 n$) time to complete.

- Extreme cases for unbalance BST



Left skewed tree



Right skewed tree.

Above BST will take n instruct^{n}s to perform any operation.

$\therefore$ BST needs to be Balanced.

Scanned with CamScanner

29 Wednesday May 2013

AVL Trees

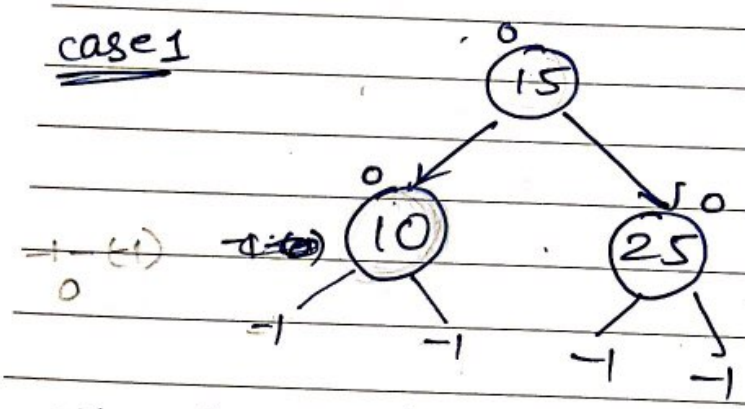
- Self-balancing BST
- Every Node has Balancing Factor

Balancing Factor

$$BF = \left(h_{(LST)} - h_{(RST)}\right)$$

$$BF = \boxed{-1, 0, 1}$$ ← Permissible Values

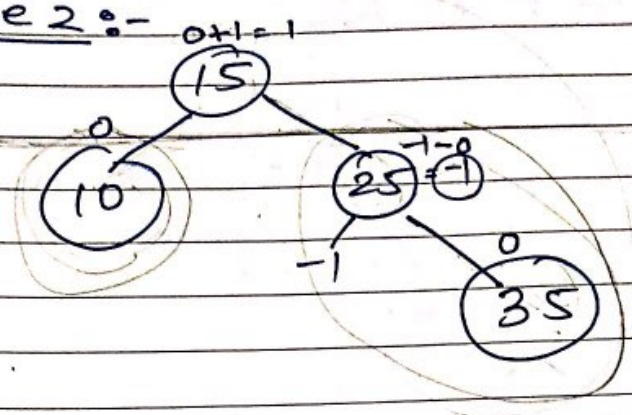
case 1



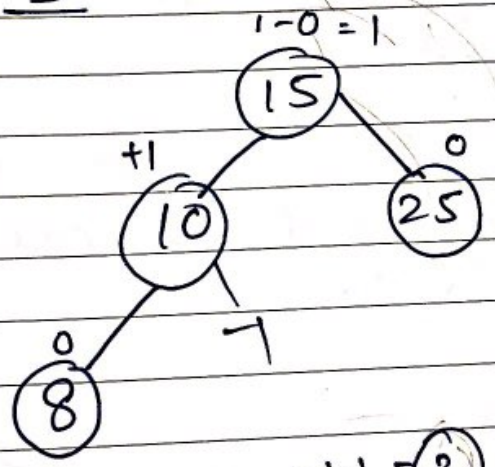
Ht of empty tree −1

Ht of ~~one~~ tree with one node 0

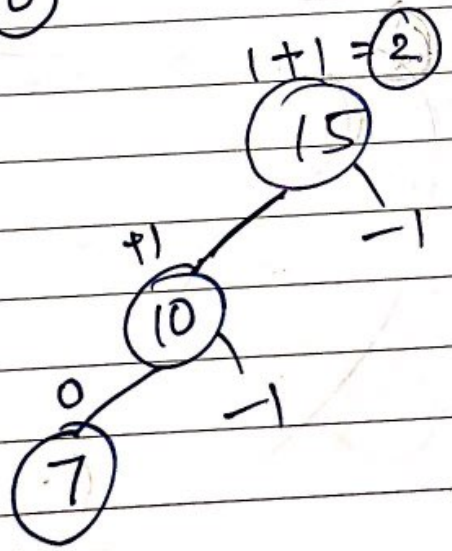
case 2 :-



Case 3

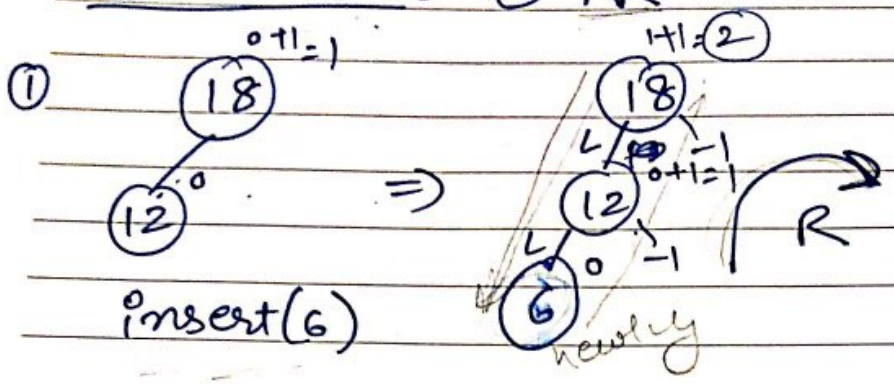


case 4



Scanned with CamScanner

Rotations :- ① RR Rotation



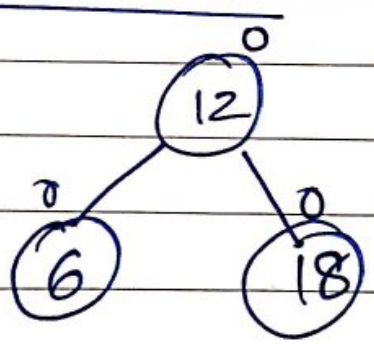
insert(6)

① Start from the newly inserted node to the node whose BF is not valid.

② from that node go back to newly inserted node & count first two moves

LL → R

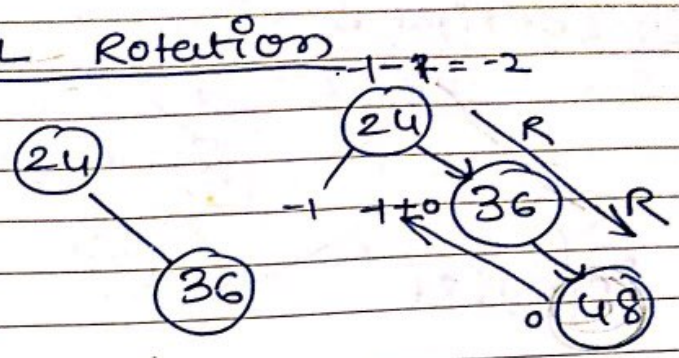
After Rotation



Scanned with CamScanner

01 Saturday June 2013

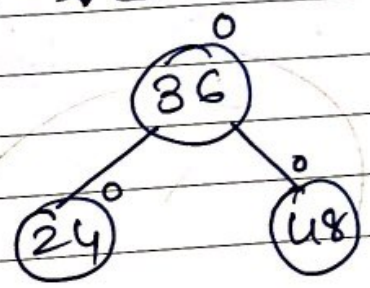
(2) ~~RR~~ LL Rotation



insert(48)

← Follow same steps.

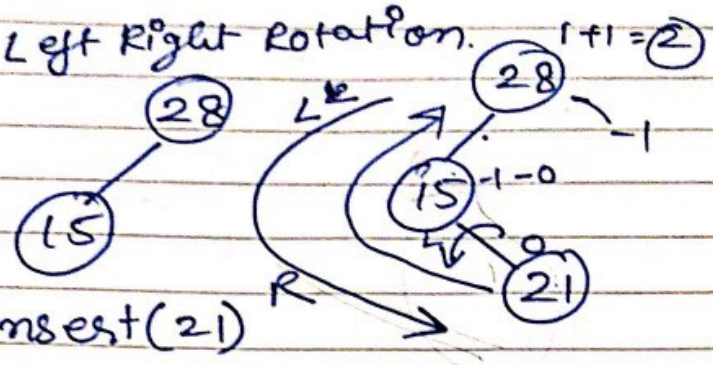
RR → L



02 Sunday

Scanned with CamScanner

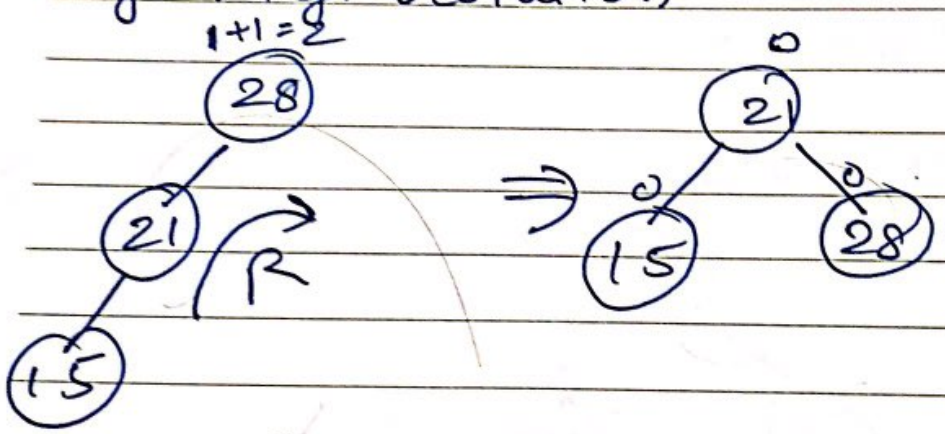
③ Left Right Rotation.



insert(21)

LR → LR

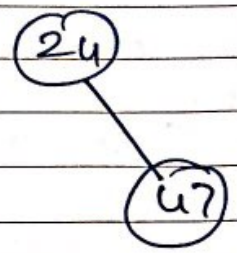
after left rotation

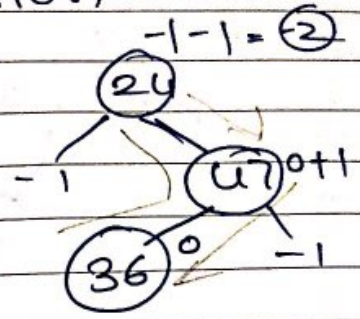


Scanned with CamScanner

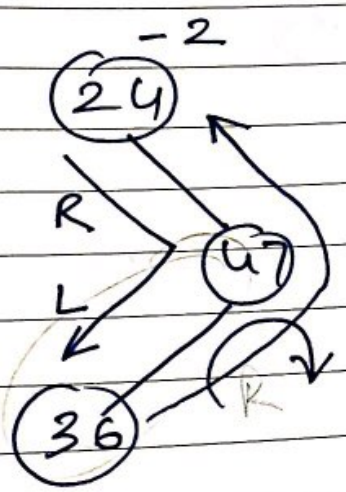
04 Tuesday June 2013

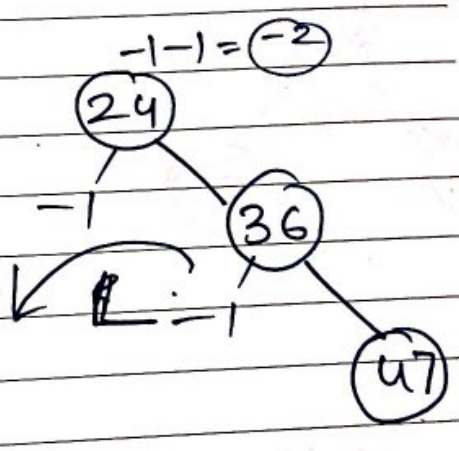
(4) Right Left Rotation

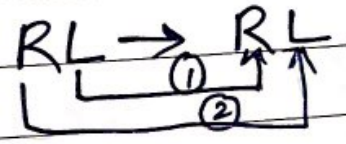


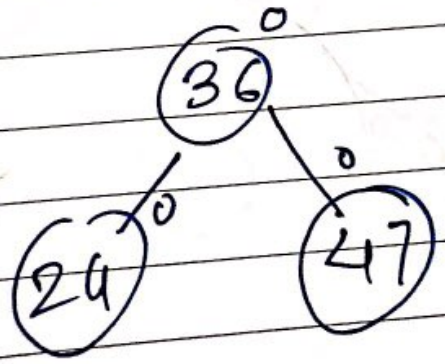


insert (36)





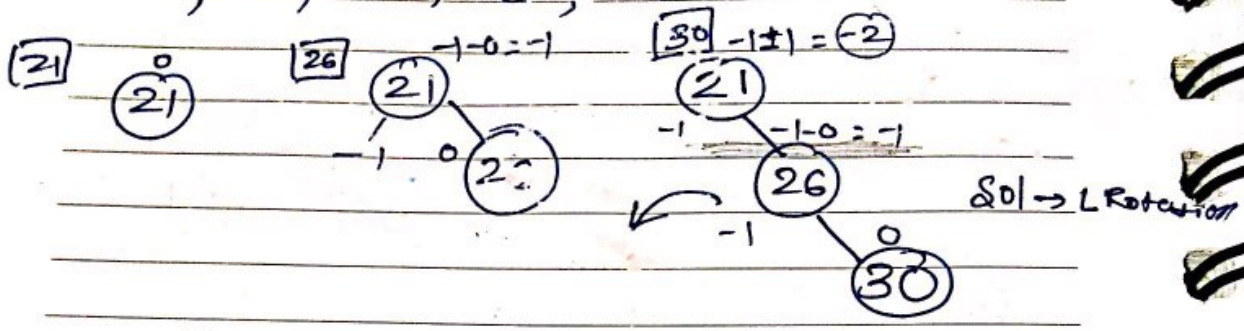


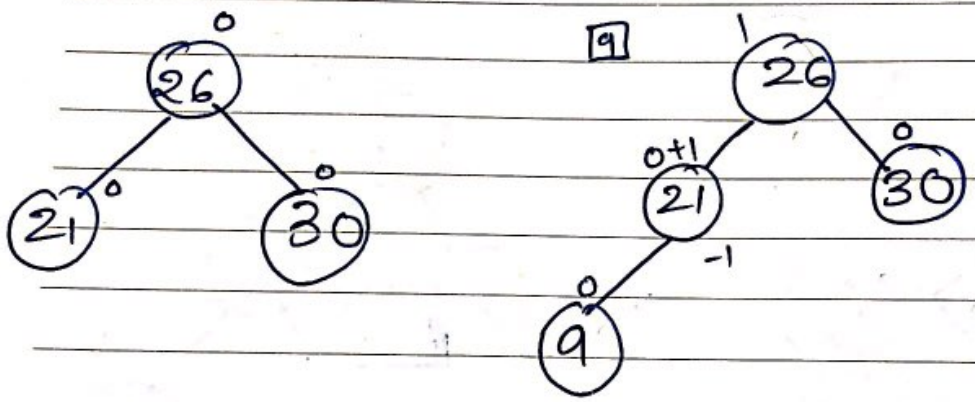


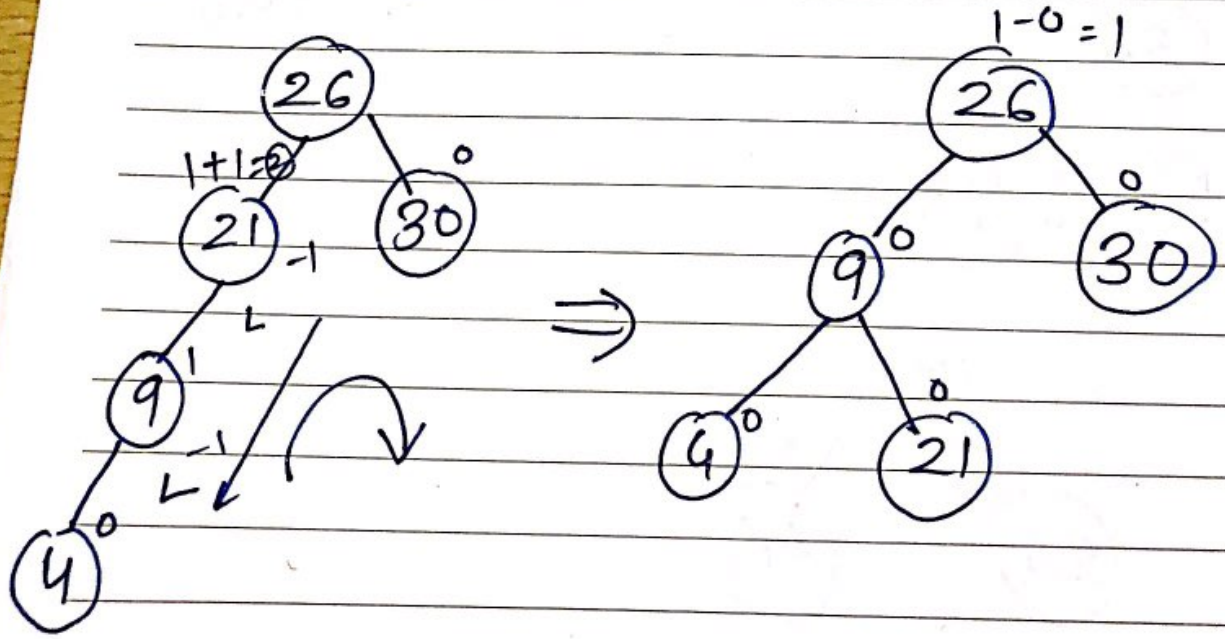
Scanned with CamScanner

05 Wednesday June 2013

Q 21, 26, 30, 9, 4, 14, 28, 18, 15, 10, 2, 3, 7



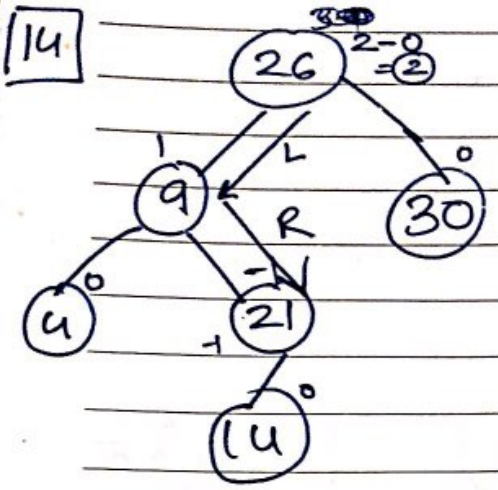




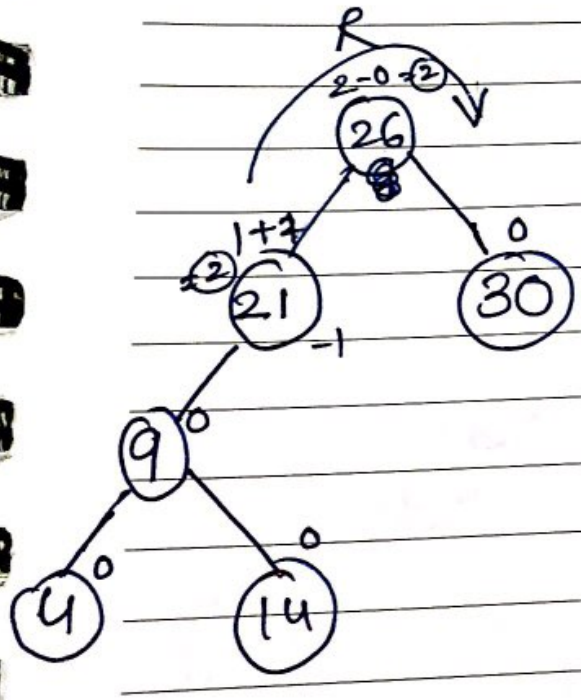
Scanned with CamScanner

06 Thursday June 2013

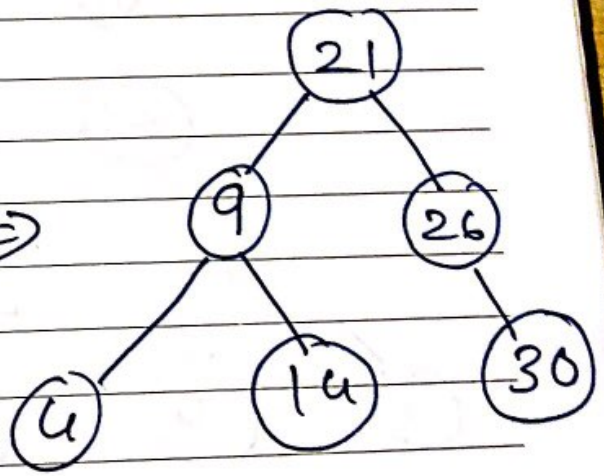
14



LR → LR



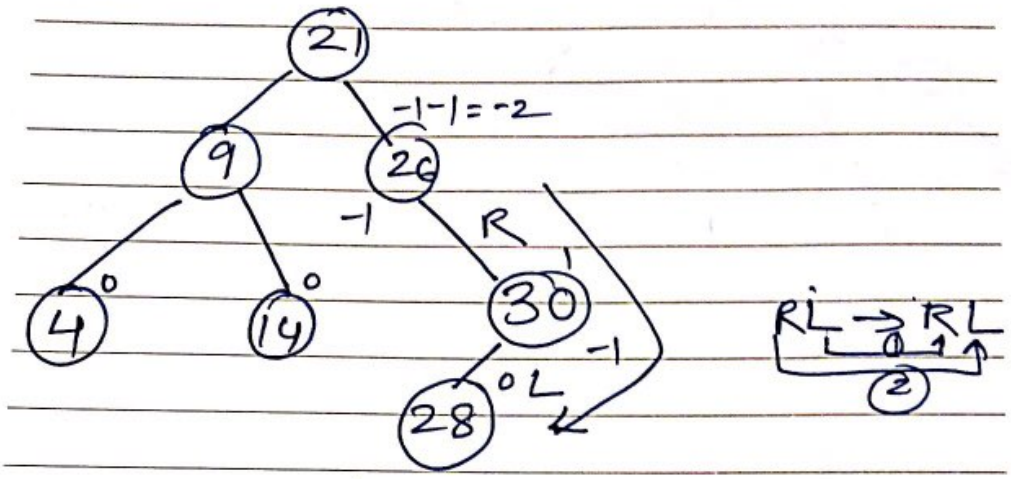
⇒

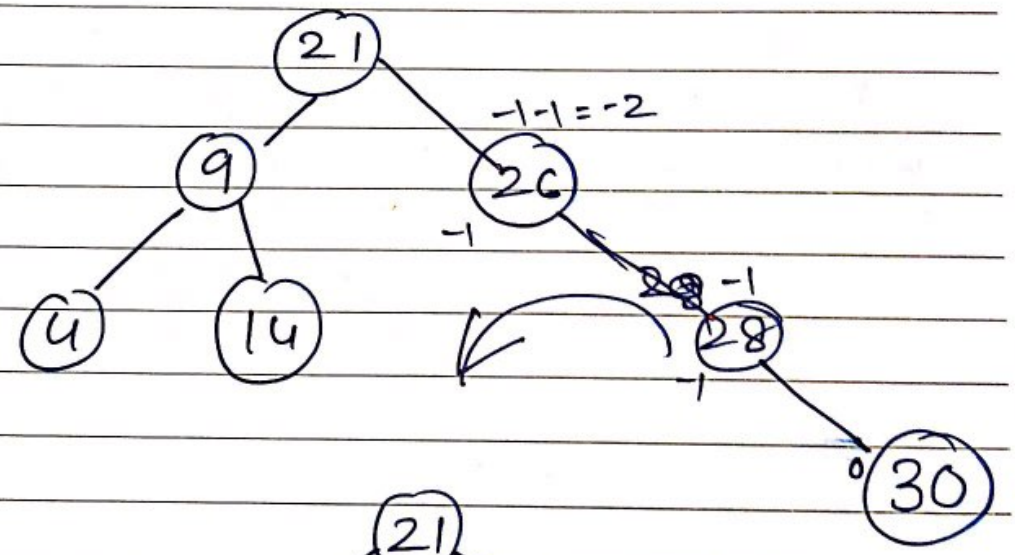


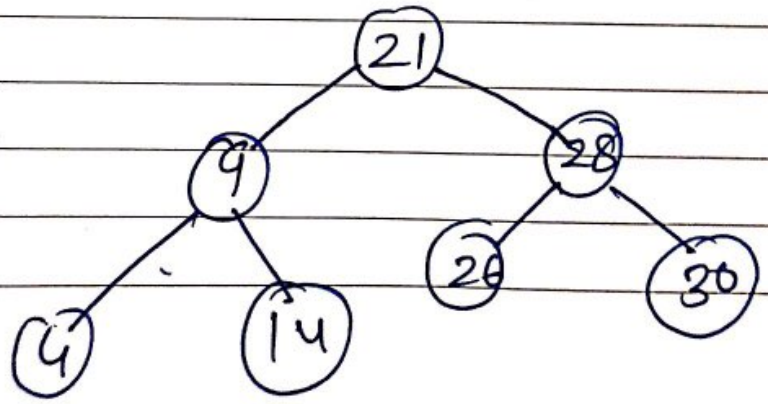
Scanned with CamScanner

07 Friday June 2013

28



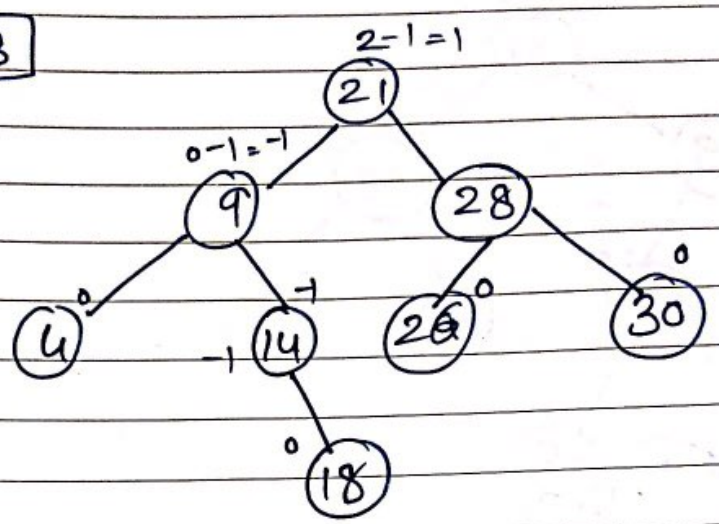




Scanned with CamScanner

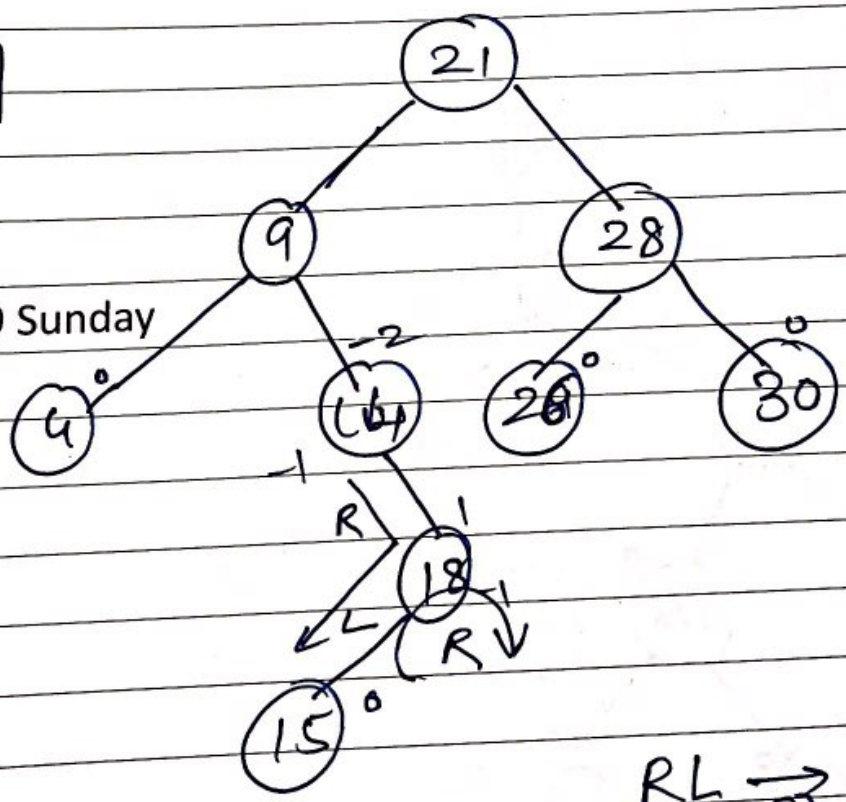
08 Saturday June 2013

18



NO problem

15



09 Sunday

RL → RL

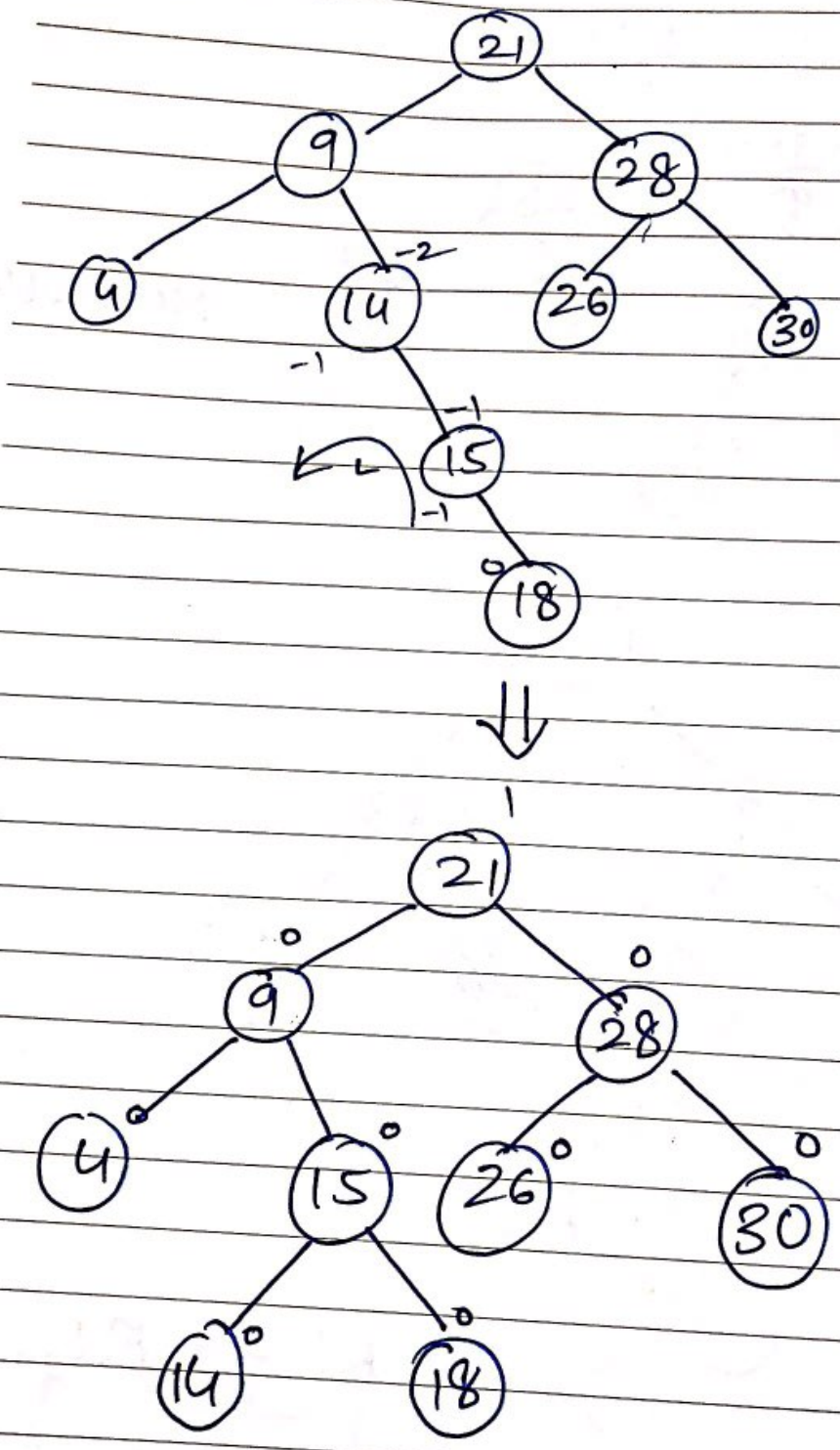
1

2

Scanned with CamScanner

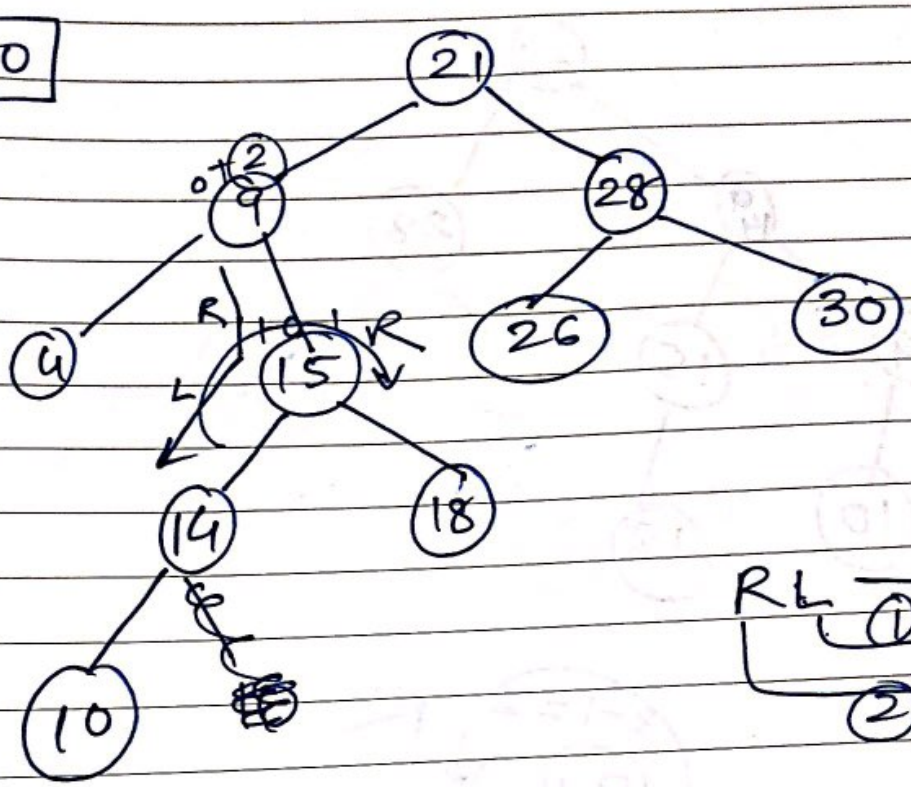
10 Monday

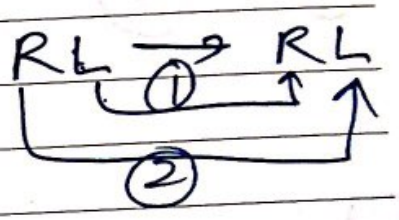
June 2013

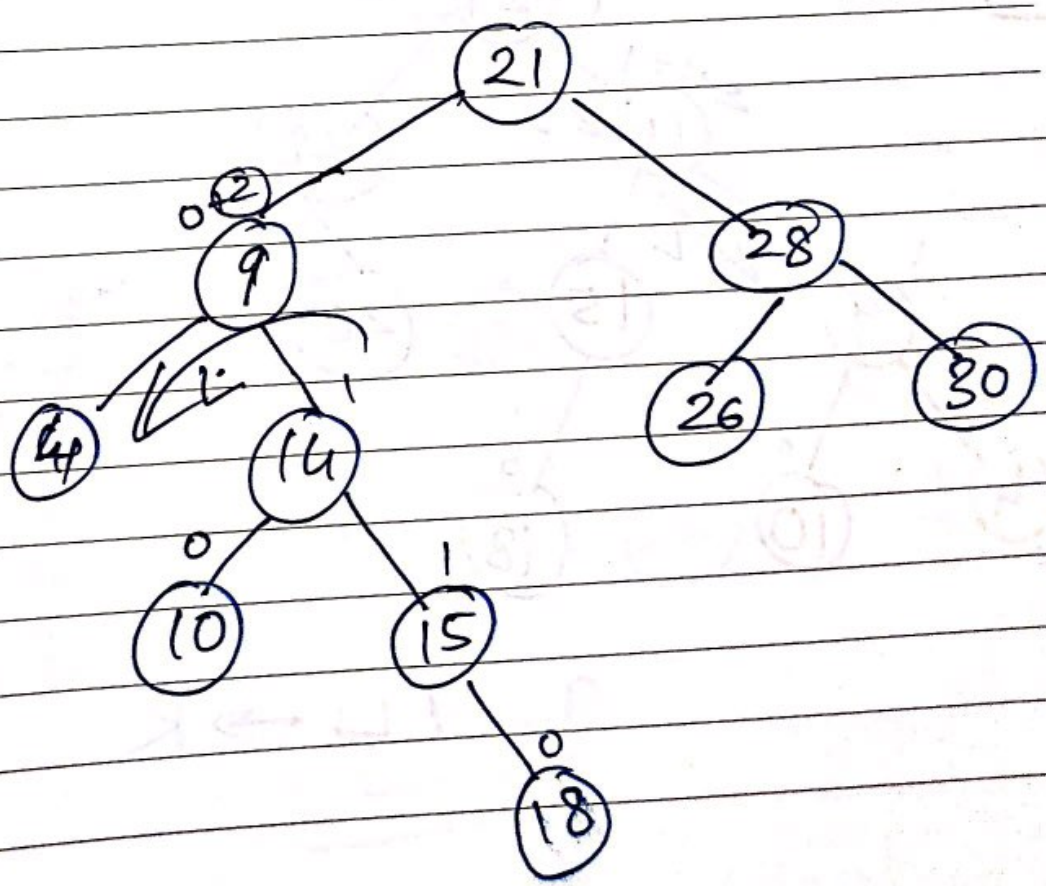


Scanned with CamScanner

10

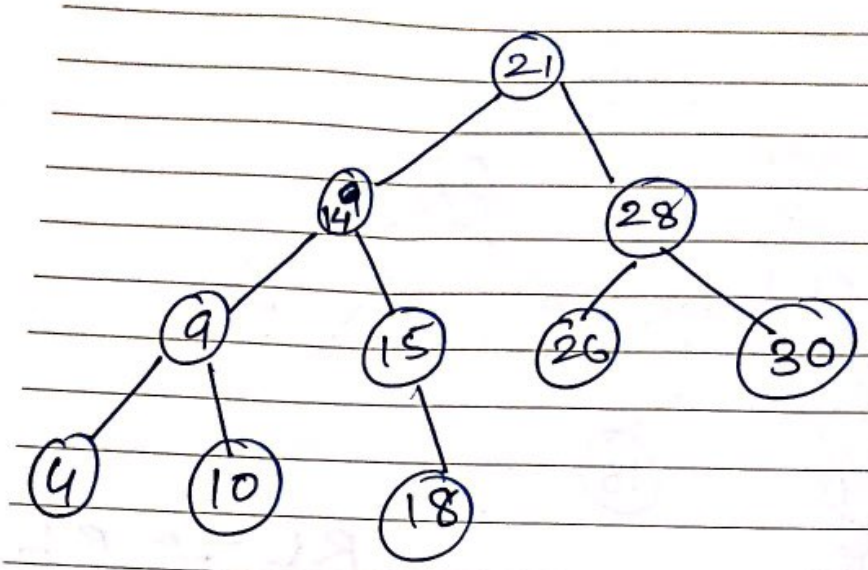




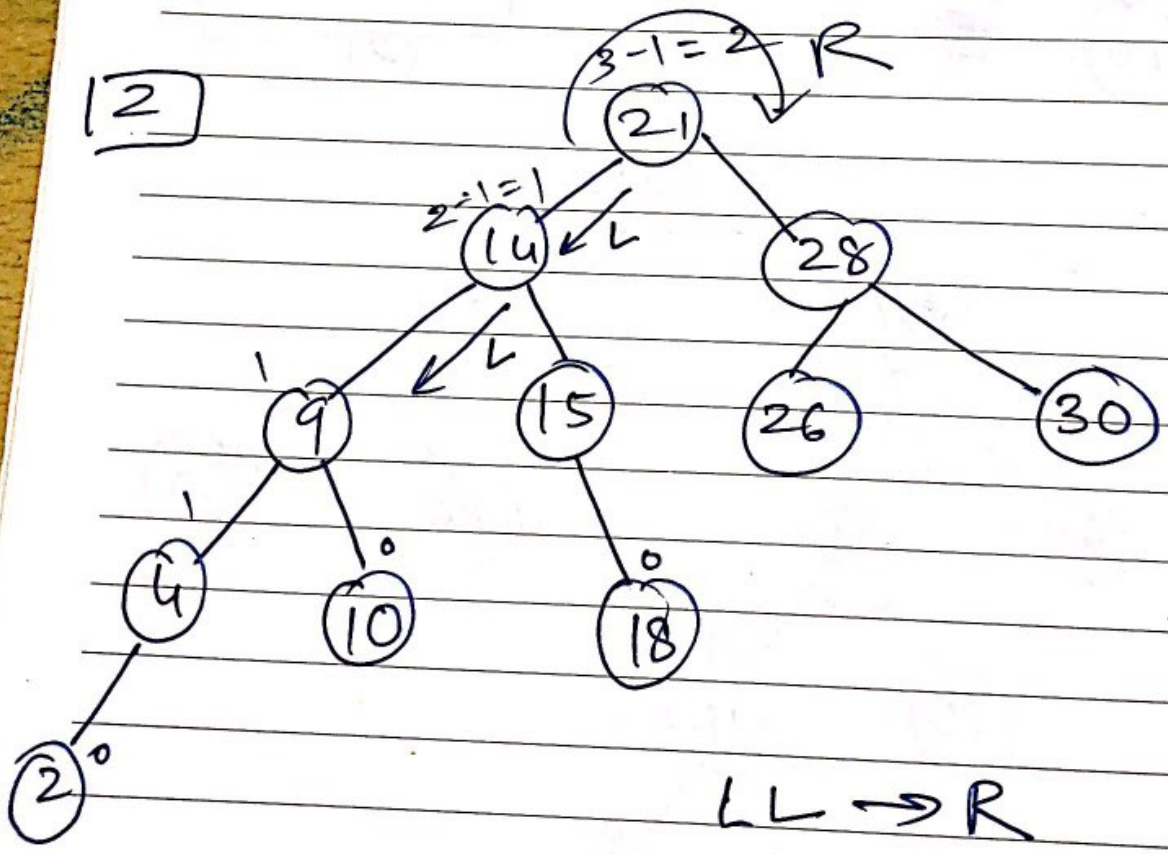


Scanned with CamScanner

12 Wednesday June 2013

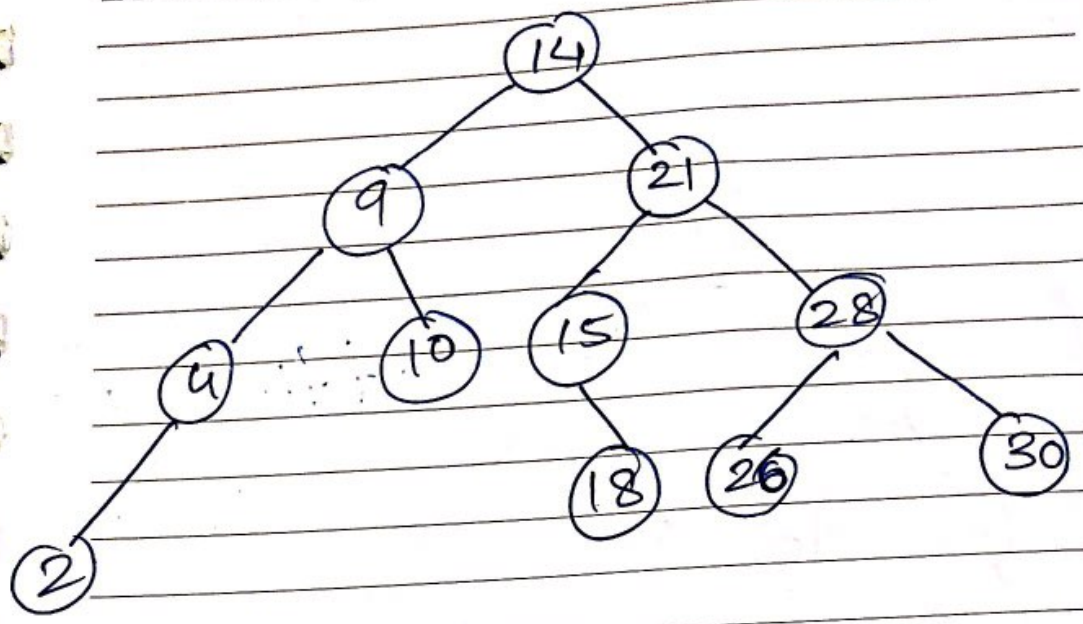


$\boxed{2}$

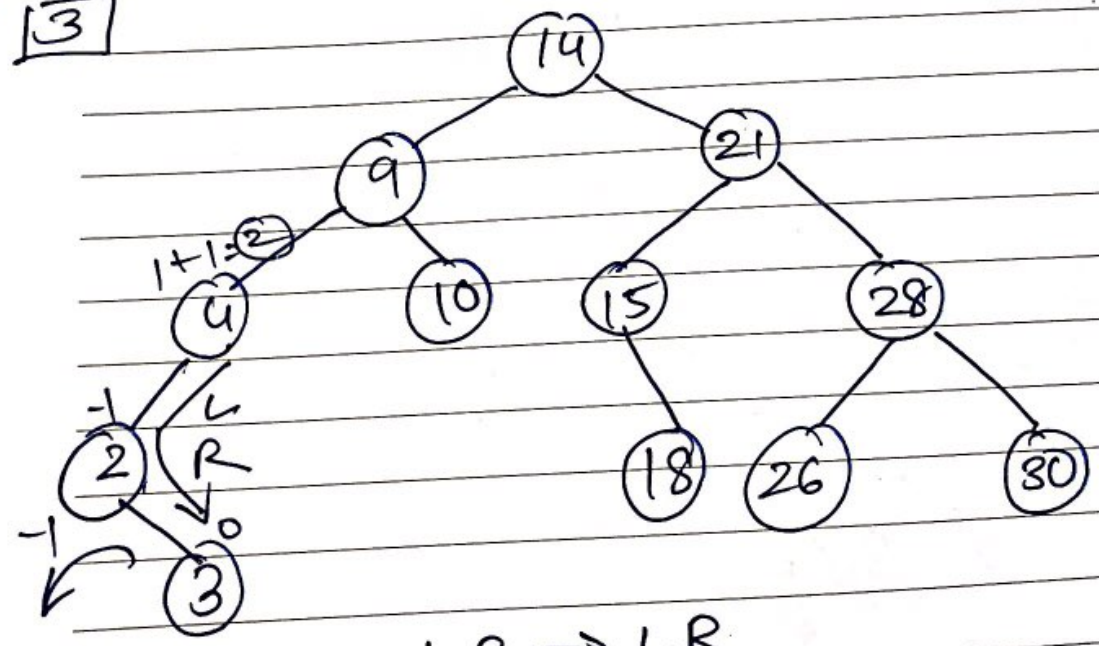


LL → R

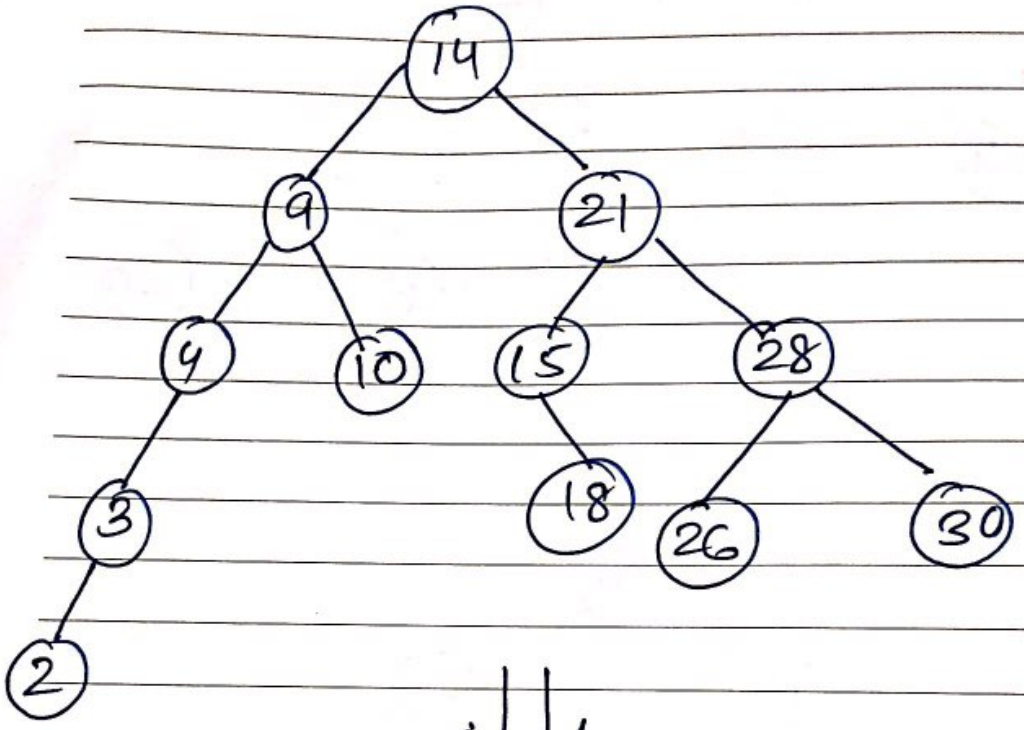
Scanned with CamScanner



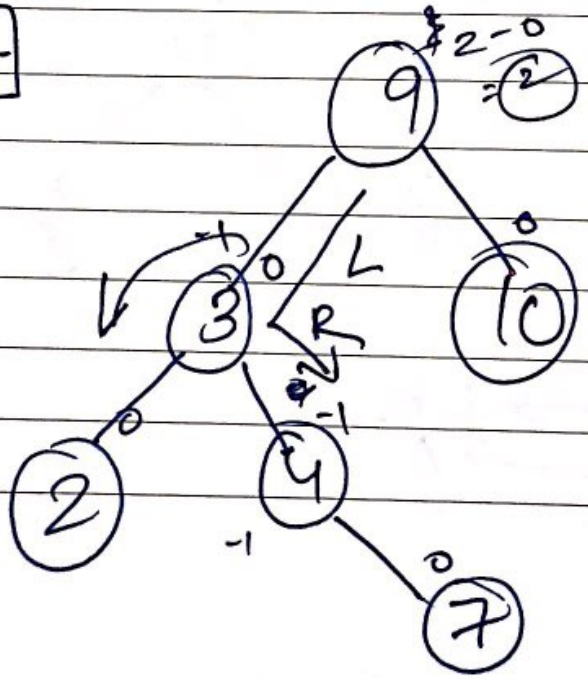
$\boxed{3}$



LR → LR



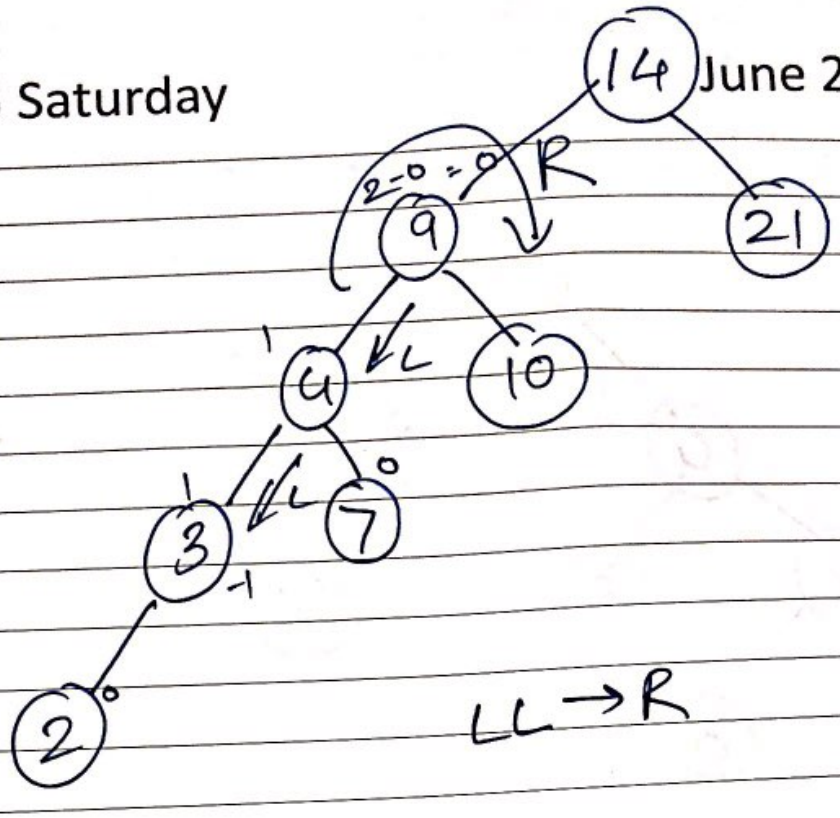
$\boxed{7}$



LR → LR

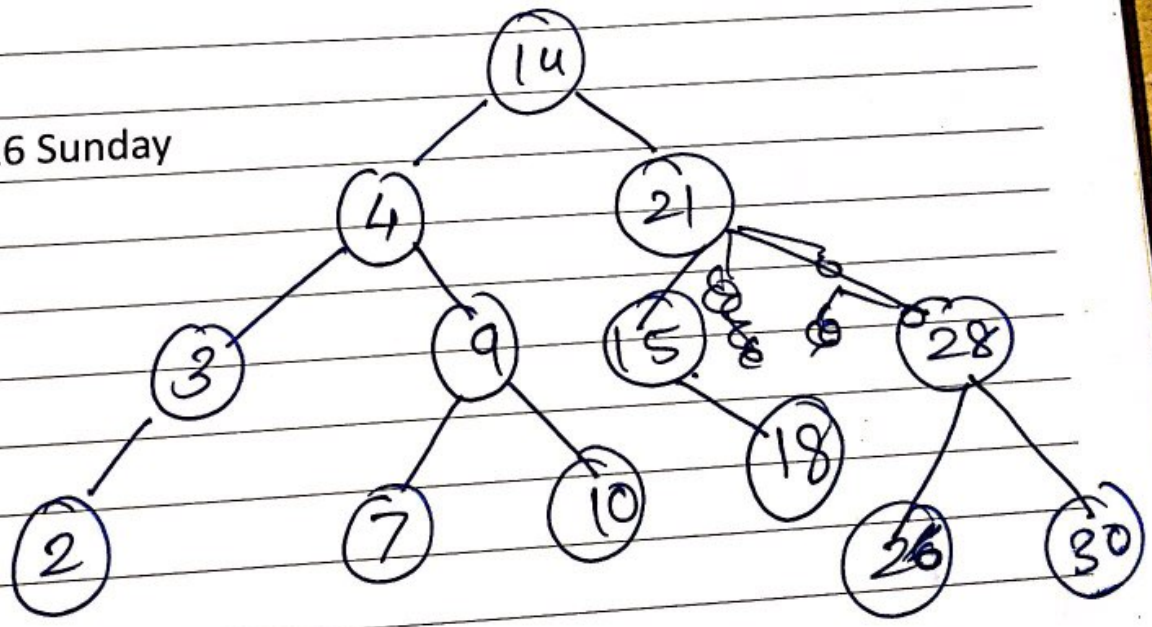
Scanned with CamScanner

15 Saturday June 2013



LL → R

16 Sunday

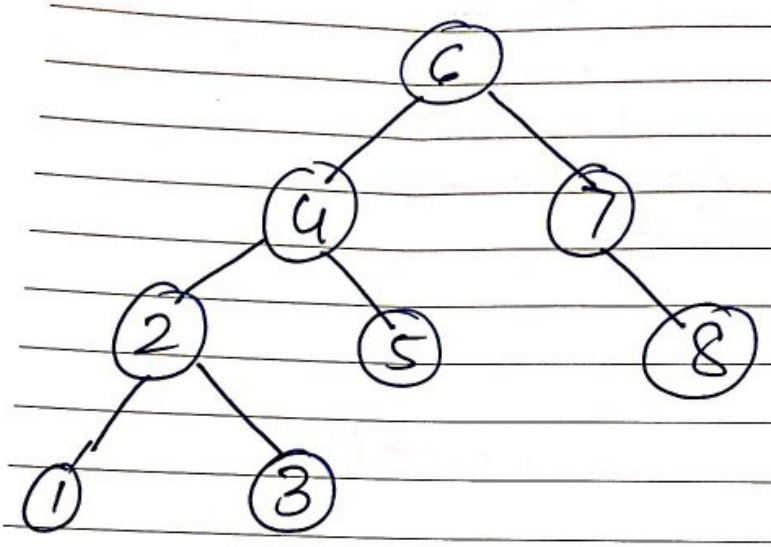


eg

Scanned with CamScanner

17 Monday June 2013

eg 3, 4, 8, 6, 7, 5, 2, 1

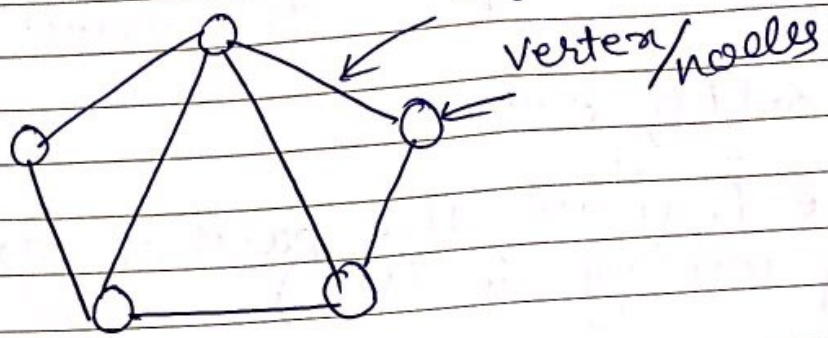


Scanned with CamScanner

June 2013

18 Tuesday

Graphs



- collection of nodes connected by edges.

Applications:-

1. Set of cities via rail route
2. connection among computers (routers)
3. ~~Tree~~ Tree is a graph.

Scanned with CamScanner

19 Wednesday June 2013

A graph G consists of 2 things:-

- A set V of elements called nodes
- A set E of Edges
- We indicate the parts of graph by writing $G = (V, E)$.

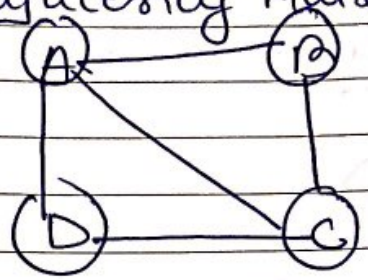
Types:

1. Undirected
2. directed
3. Weighted.
4. Cyclic

Scanned with CamScanner

Representation of Graphs.

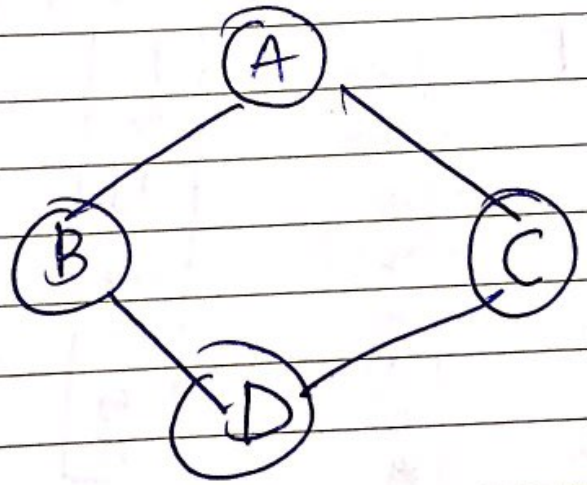
① Adjacency Matrix

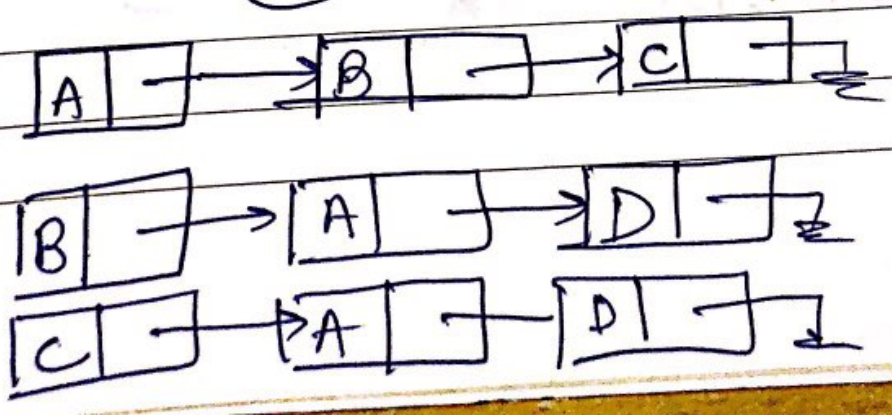


2D array

	A	B	C	D
A	0	1	1	1
B	1	0	1	0
C	1	1	0	1
D	1	0	1	0

②



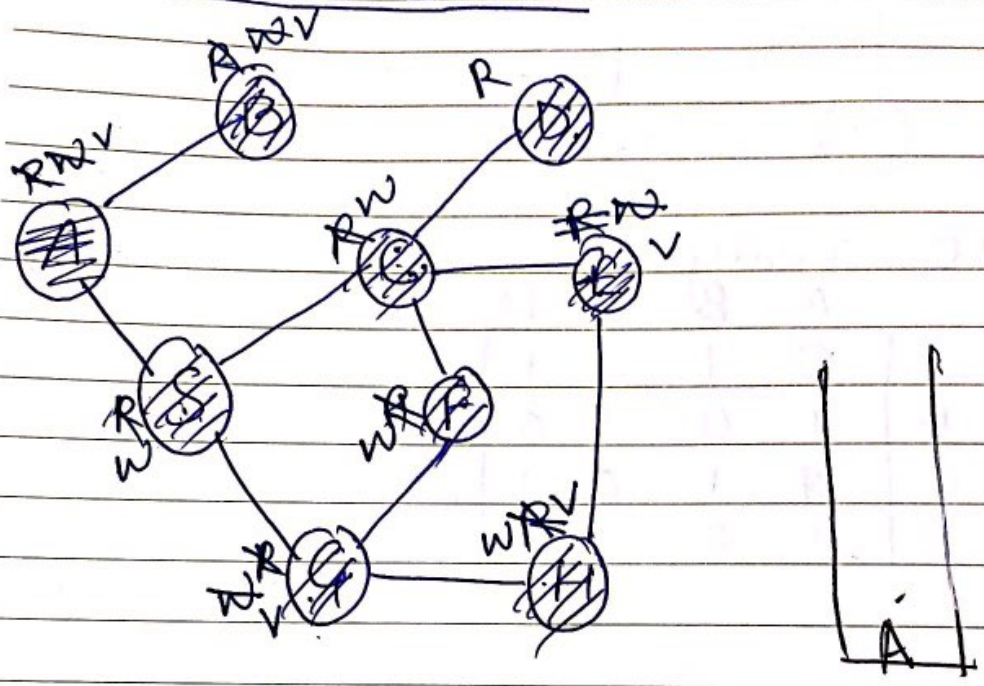


Scanned with CamScanner

21 Friday June 2013

Graph Traversal algorithms

① Depth first Search :-

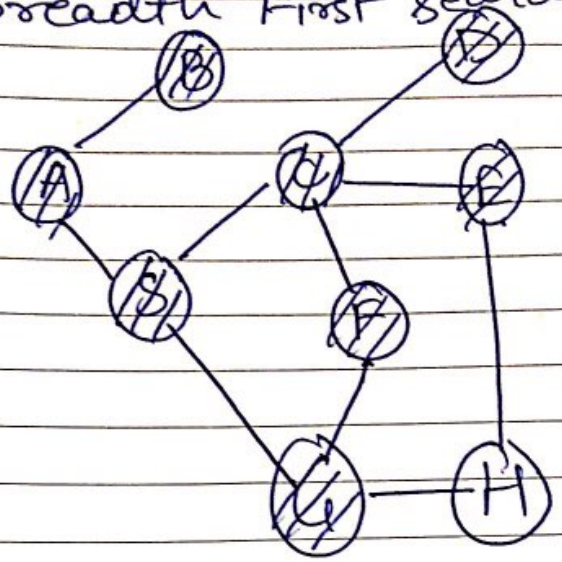


ABSCDEHGF

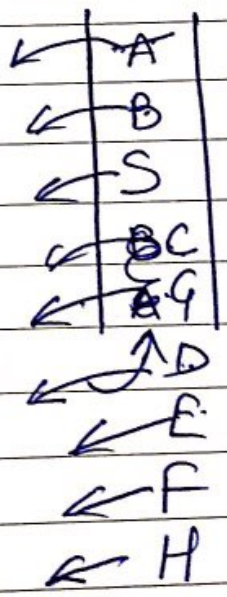
ASGHEFCDB

22 Saturday June 2013

Breadth first search.



O/P: ABSCGDEFH



23 Sunday

Scanned with CamScanner

BFS Algorithm :-

During execution of algo each node N of G will be in one of three nodes called status of N:

Status 1 = Ready
2 = Waiting
3 = ~~Processed~~ visited

Algorithm :-

1) Initialize all nodes to ready state(1)

2) Put the starting node A in Q & change it status ~~to~~ to the waiting state (2)

3) Loop
Repeat 4 & 5 until Q is Empty

Scanned with CamScanner

4) Remove the front node N of Q
Process N & change status of
~~processed~~ visited state (3)

5) Add to the rear of Q all
the neighbors of N that are
in the ready (1) state &
change their status to
waiting state (2)

end of loop

6) Exit.

Scanned with CamScanner

28 Friday June 2013

DFS Algorithm

1) Initialize all nodes to the ready state (1)

2) Push the starting node A onto stack & change its status to the waiting state (2)

3) Loop
Repeat step 4 & 5 until stack is empty

4) Pop the top node N of stack. Process N & change it status to the ~~processed~~ visited state (3)

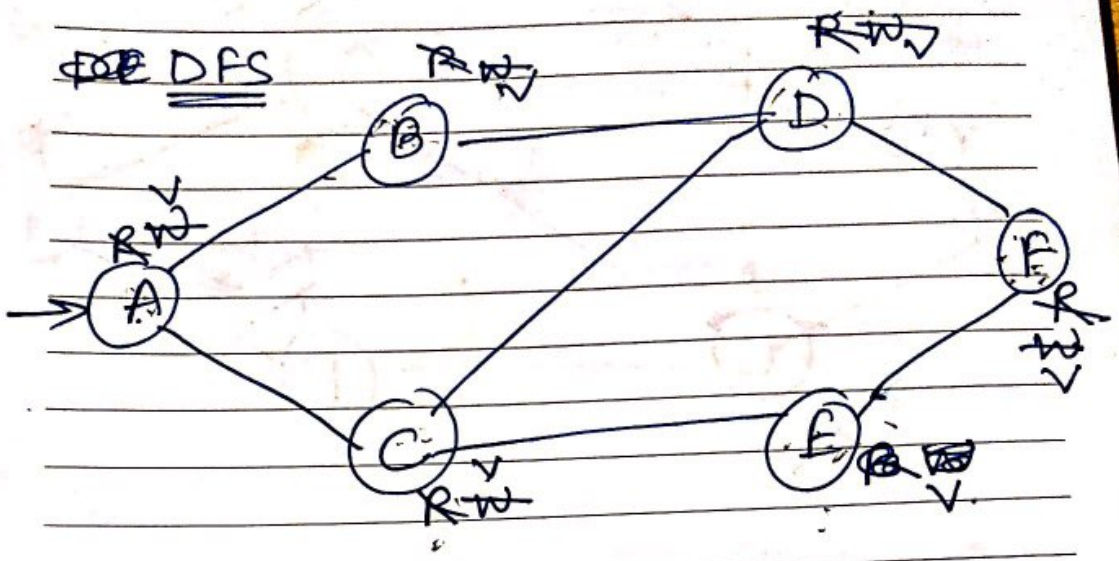
5) Push onto stack all the neighbours of N that are still in ready state (1) & change their status to waiting state (2)
end loop

6) Exit.

Scanned with CamScanner

29 Saturday June 2013

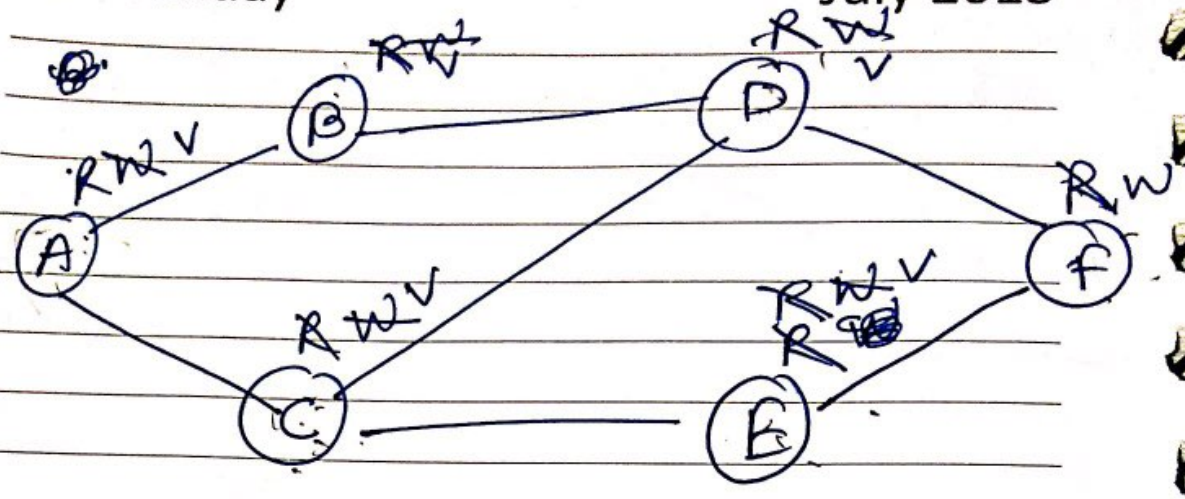
DFS



30 Sunday

A C E F D B

01 Monday July 2013



A B C D E F

03 Wednesday July 2013

Singly Linked List operations

(1)
```
void search(int num)
{
struct Node *temp = head;
int a=1;
while (temp! = NULL)
{
  if (temp->data == num)
  {
    break;
  }
else
  {
    a++;
    temp = temp-> next;
  }
}
printf("Enter value is at: %d", a);
}
```

Scanned with CamScanner

04 Thursday July 20.

Count No. of Nodes :-

```
void count( )
{
Struct Node *temp=head;
 int a=0
  while (temp!=NULL)
  {
   a++;
   temp= temp->next;
  }
 printf("No. of nodes : %d", a);
}
```

Scanned with CamScanner

(3) Concatenate two linked list

```
void concat()
{
 struct Node *temp1, *temp2
  temp1 = head1
  temp2 = head2

 if(temp1 == NULL)
 {
  temp1 = temp2
 }
 else
 {
  while(temp1 -> next != NULL)
  {
   temp1 = temp1 -> next;
  }
 ~~temp1 -> next temp1 ->~~
  temp1 -> next = temp2
}
```

Scanned with CamScanner

(4) Sorting nodes in Linked List

```
void sorting ()
{
struct Node * ptr1, ptr2;
int temp;
ptr1 = head;
while (ptr1 -> next != NULL)
{
  ptr2 = ptr1 -> next;
  while (ptr2 != NULL)
  {
    if (ptr1 -> data > ptr2 -> data)
    {
      temp = ptr1 -> data;
      ptr1 -> data = ptr2 -> data;
      ptr2 -> data = temp;
    }
    ptr2 = ptr2 -> next;
  }
  ptr1 = ptr1 -> next;
}
}
```


Scanned with CamScanner

```
void sort( )
{
struct Node *p, *q;
int temp;
for( p=link; p->next != NULL ; p=p->next)
{
  for(q=p->next; q != NULL ; q=q->next)
  {
    if(p->info > q->info)
    {
      temp= p->info;
      p->info = q->info;
      q->info = temp;
    }
  }
}
}
```

Scanned with CamScanner

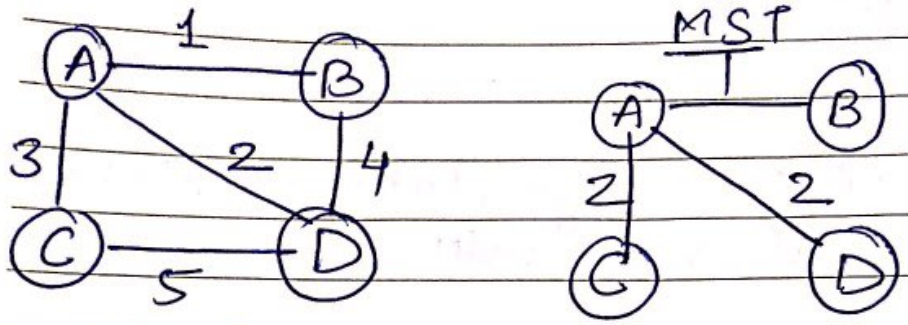
09 Tuesday July 2013

Sorting Linked List

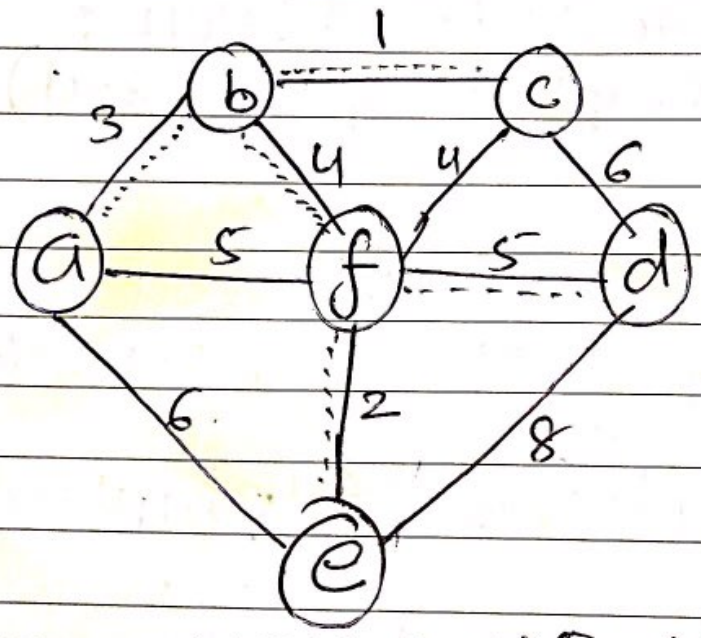
```
void sort()
{

/* start, ~~head~~ next & data declared
   as global variable */

int num;
struct node *temp1, *temp2;

for(temp1 = start; temp1 != "NULL";
        temp1 = temp1 -> next)

{
  for(temp2 = temp1 -> next;
      temp2 != "NULL"; temp2 = temp2 -> next)
  {
    if(temp1 -> data > temp2 -> data)
    {
      num = temp1 -> data;
      temp1 -> data = temp2 -> data;
      temp2 -> data = num
    }
  }
}
}
```

Scanned with CamScanner

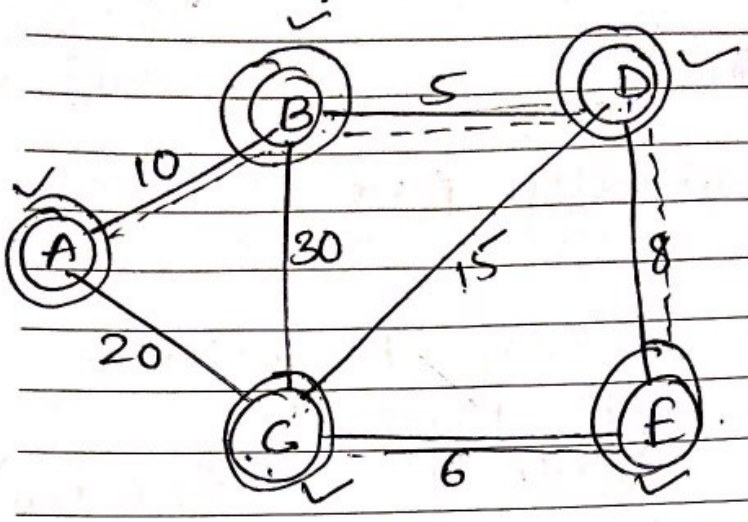
Minimum Spanning Tree



Prim's Algorithm



sum = 1+3+4+2+5 = 15



Vertex	Key	Parent
A X	0	
B X	~~∞~~ 10.	A
C X	~~∞~~ ~~20~~ ~~15~~ 6	~~A~~ ~~D~~ E
D X	~~∞~~ 5.	B
E X	~~∞~~ 8	D

$10 + 6 + 5 + 8 = 29$

A	0	
B X	10	A
C	~~20~~ 15.	~~A~~ D
D.	~~∞~~ 5	B
E.	~~∞~~ 8	D

Scanned with CamScanner

Algorithm:

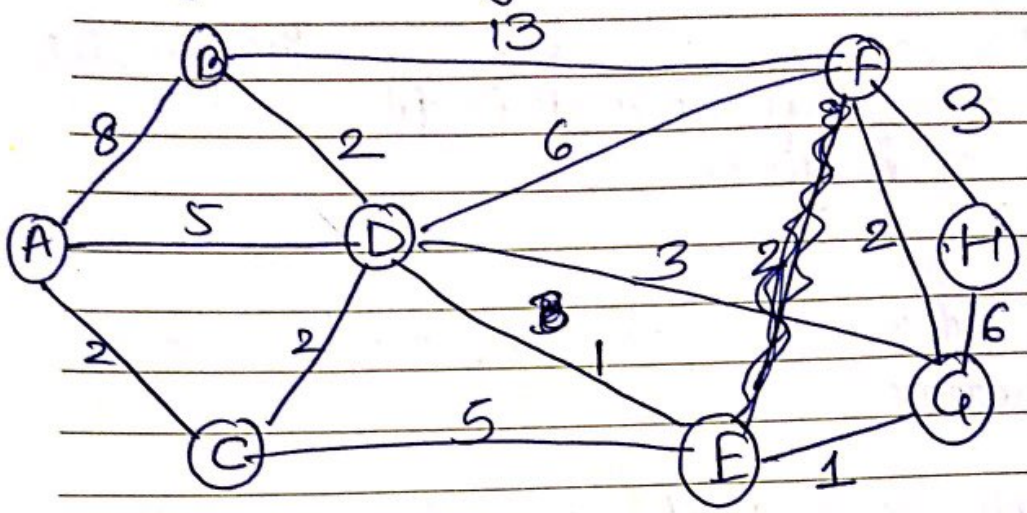
1) Create an array parent[] of size V & initialize to NULL.

2) Create an array H which all vertices such that key value of starting vertex is 0 & key value of other vertices are infinite.

3) while H is not empty
{
a) u = extractMin(H)
b) for every adjacent v of u
{
if V is in H
{i) update key value of v in H if weight of edge uv is smaller than current key value of v
ii) parent[v] = u
}
}
}

Shortest Path Algorithm :-

Dijkstra's Algorithm :-



14 Sunday

V	A	B	C	D	E	F	G	H
A	$\boxed{0_A}$	8_A	2_A	5_A	∞	∞	∞	∞
C	—	8_A	$\boxed{2_A}$	4_C	7_C	∞	∞	∞
D	—	6_D	—	$\boxed{4_C}$	5_D	10_D	7_D	∞
E	—	6_D	—	—	$\boxed{5_D}$	10_D	6_E	∞
B	—	$\boxed{6_D}$	—	—	—	10_D	6_E	∞
G	—	—	—	—	—	$\boxed{8_G}$	$\boxed{6_E}$	12_G
F	—	—	—	—	—	$\boxed{8_G}$	—	11_F
	—	—	—	—	—	—	—	$\boxed{11_F}$

G → A C D E G

15 Monday July 2013

Analysis of Algorithm :-

Why Performance Analysis

- Performance is currency with which we can buy new things.
- Speed is fun.

Two Algos for task, which one is better?

- Implement both → which takes more time.
  - * for some i/p 1st algorithm better
    " 2nd " "
  - * ~~for One~~ 1st Algo — works on 1st half

Asymtotic Analysis:-

- we evalute the performance in terms of i/p size (not actual running time)

Scanned with CamScanner

- we calculate, how does the time (or space) taken by an algorithm increases with input size

Linear	Binary
fast	slow
less i/p size	More input size
less time	takes less time

Example of Linear Search & analyze it using Asymptotic analysis.

3 cases to analyze an algorithm.

1) worst case
- calculate upper bound on Running time of an algo.
- max no. of operations.
- eg. Linear Search - x not present in array.
- worst case complexity of Linear Search would be $O(n)$

Scanned with CamScanner

Avg case

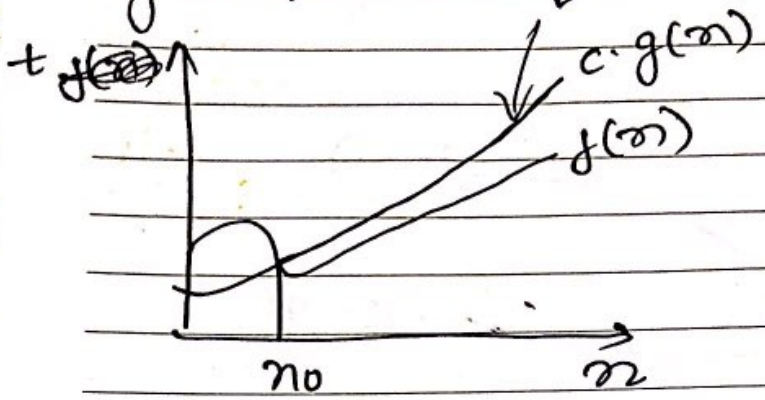
Best case :-

- Calculate lower bound
- min no. of operations to be executed.
- Guaranting lower bound doesnt provide any info
- bogus case.

Scanned with CamScanner

18 Thursday July 2013

Big oh(O) (Upper Bound)



$$\boxed{\begin{array}{l} f(n) \le c \cdot g(n) \\ \qquad n \ge n_0 \\ c > 0, \quad n_0 \ge 1 \end{array}}$$

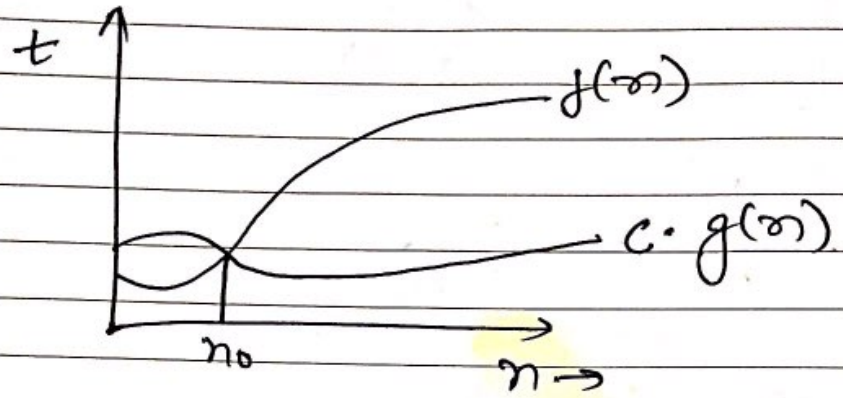
$$f(n) = O(g(n))$$

- useful to describe limit of worst case running time for a algo.

Q

19 Friday July 2013

Big Omega (Ω) → Lower bound (Ω)



$f(n) \geq c \cdot g(n)$

$n \geq n_0$

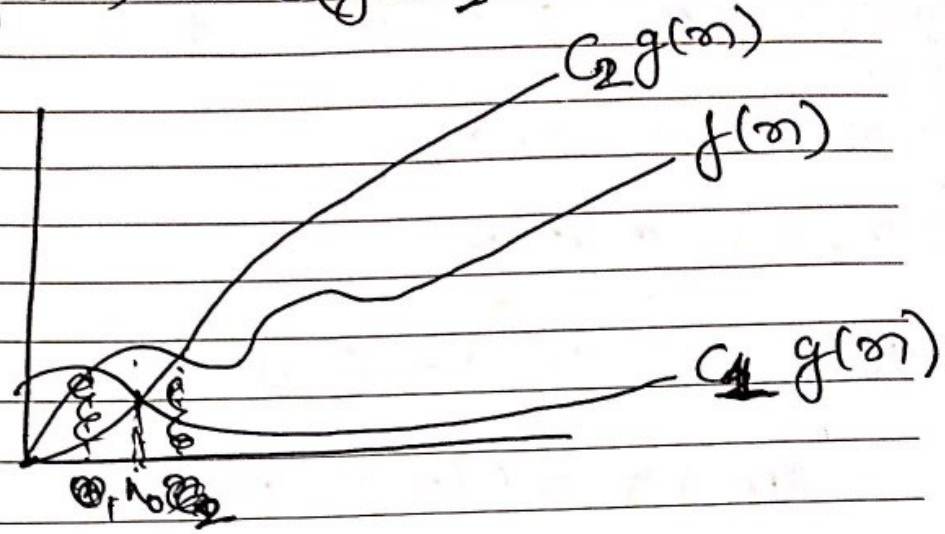
$f(n) = \Omega(g(n))$

Scanned with CamScanner

20 Saturday July 2013

Θ (theta) [tight Bound]

$f(n) = \Theta(g(n))$



21 Sunday $C_1 g(n) \leq f(n) \leq C_2 g(n)$

$n \geq n_0$

22 Monday July 2013

Big oh

$f(n) = 2n+5 \quad g(n) = n$

$c = 3$

$2n+5 \leq n$

$f(n) = O\,g(n)$

$2n+5 = O(n)$ for $\forall\, n \geq 5$

& $c = 3$

Omega (Ω)

$f(n) \geq c \cdot g(n)$

$2n+5 \geq c \cdot n$

$c = 2$

$n = 1 \quad 7 \geq 2$

$2n+5 = \Omega(n)\ \forall\, n \geq 1$

$c = 2$

Scanned with CamScanner

23 Tuesday July 2013

Theta

$c_1 g(n) \leq f(n) \leq c_2 \cdot g(n)$

$c_1 \cdot n \leq 2n+5 \leq c_2 \cdot n$

$c_1 = 2 \quad c_2 = 3$

$2n \leq 2n+5 \leq 3n.$

$n = 5 \quad f(n) = \Theta(g(n))$

$2n+5 = \Theta(n) \ \forall n \geq 5$

$c_1 = 2, \ c2 = 3.$

Scanned with CamScanner

24 Wednesday July 2013

① • She is not supervised.

② Thinking of solⁿ.
- alternatives
- encourage in diff. situations.
- Good mentor.

Leadership skills

18-34 work hard.
35-52 OK
53-90 Excellent

SWOT analysis

This Multidisciplinary conference aims to impart advanced Knowledge by bringing together all academicians, professionals, researchers & students to facilitate close interaction for exchange of views & dissemination of knowledge

Scanned with CamScanner

26 Friday July 2013

1. Emotional Intelligence. Connection
2. Persuade Team members
3. Make them aware of your vision time to time with example.
4.

Scanned with CamScanner

27 Saturday July 2013

10 Technologies in MIT

1. 3D Metal Printing.
2. Artificial Embryos.
3. Smarter Smart City
4. AI for Everybody
5. Dueling Neural Network
6. Babel fish Earbuds
7. 28 Sunday Zero Carbon Natural Gas
8. Perfect Online Privacy
9. Genetic Forture Telling
10. Material Quantum Leap
11. Google Duplex
12. Smart Clothes
13. Little Dairy takes cows to cloud

Scanned with CamScanner

29 Monday July 2013

Assignment :-

2COS → One assignment

Test

Tabular Form — Question Paper.

- atleast 2 intercollege events.
- interview.
- today induction
- CSI magazine
- Industrial visit.
- Something diff than regular.

Scanned with CamScanner