ct 10 mark - 15 min
per ct 4 set

CO2 - Algorithm
CO3 - Implementation

Asymptotic Notations

I) O notations — checks worst case scenerio
—— >> best Possible >>
II) Ω >>
III) Θ >>
IV) o >>
V) ω >>

Data structure

I) Linear - example: Array, Linkedlist, Queue

II) Non Linear: example: Trees, Graph..

Data structure operation:

I) Searching
II) Inserting
III) Traversing, example: Printing
IV) Deleting
V) Sorting
VI) merging.

Algorithm : a well defined set of instructions used to solve a particular problem.

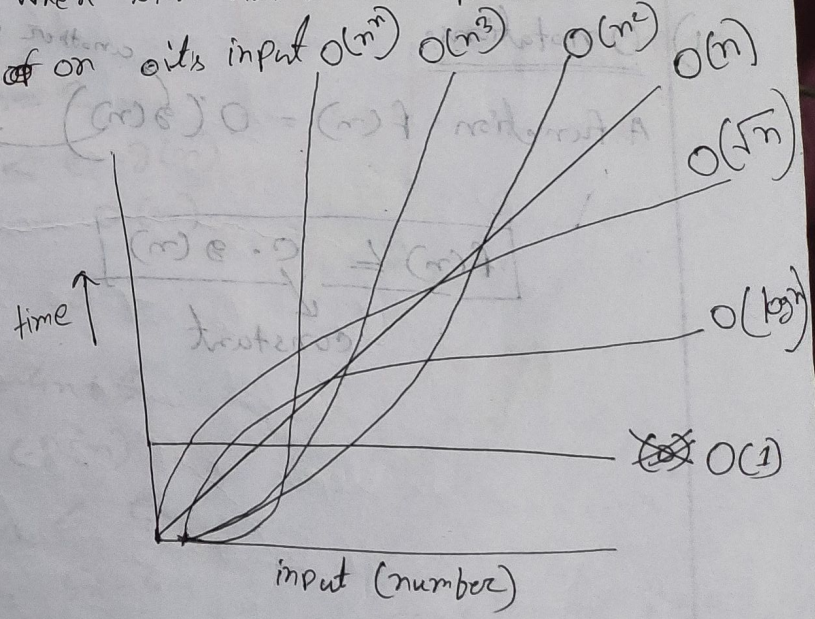
2/ Difference between Data structure and Algorithm ?

Data structure is for efficient Data manupulation and Algorithm is the steps for Data manupulation.

f(n) **Time complexity** — space

1) Assignment operation.
2) Comparison
3) Arithmatic
4) function call

Big O(1) : when total number of operation Doesn't Depend on its input (order of one)



24/07/2023

$f(n)$ → number of inputs

When, $n = 1$, % of running time due to

$$5n^2 = \frac{5}{5+6+12} \times 100$$

$$= 21.\overline{--}\,\%$$

Big O: Worst case scenario – Upper bound

1) **O notations:**

A function $f(n) = O(g(n))$ → Input number; $g(n)$ → another function

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

c ↓ constant

Notations works only in Loops

1) $f(n) = 3n + 2$ (constant)

If $f(n) \leq c \cdot g(n)$

$3n + 2 \leq c \cdot n$

$3n + 2 \leq 4n$

$+2 \leq 4n - n$

$2 \leq n$

∴ minimum number of input = 2

Here,

$c = 4$

$n_0 \geq 2$

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

$f(n) = O(n)$

2) $f(n) = 4n + 3$

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

$4n + 3 \leq 5n$

$3 \leq 5n - 4n$

$3 \leq n$

∴ min number of input = 3

$f(n)$ = n + total unit (It'll grow Linearly)

$f(n) = 3n + 2$

3 ← constant loop number; 2 ← unit count

$g(n) = n$ ← let

minimum constant will be needed

Before Mid question solves

Compare Linear Search, Binary search and Interpolation search.

Ans:

SP-21

Binary Search: (half interval search) is an efficient algorithm for finding a specific value in a sorted array or list. It works by repeatedly dividing the search interval in half and comparing the middle element with the targated value. If the middle element is greater than the target values it continues looking the left middle value. If the middle element is less than targated value the search continues in upper half. The search iteration is reduced half in every iteration and the time complexity is $O(\log n)$

Where n is the number of element in the array. It's only possible in sorted array.

SP-22

Linear Search: is also known as sequential Search to finding a specific value inside an array / list. It compares every element in with the targeted value until it finds a match. The time complexity of linear seach is $O(n)$

Binary Search is much faster than linear Seach. Because it has time complexity of $O(\log n)$ so it can search through much larger arrays much faster.

Array doesn't have to be sorted multidimentional array can be use

Interpolation Seach: is a extended version of Binary Seacich. It also works in Sorted array. It uses interpolation to estimate the position of the value in the array and then compare

tries to find the exact position of that targeted value on possible movement position where the targeted value might be in. it might have some few iteration than Binary search, and to But data uniformly dis

the estimated value with the target value, until it found a match. Interpolation search is Better when the data is Sorted and Uniformly distributed. It calculate the possible location where the data might be located.

Linear Search;

```
int Search (int arr[], int n, int x)
{ for (int i = 0; i < n ; i++)
  { if (arr[i] == x)
      return 1 }
  else
      return not found

}
```

Binary Search;

```
while (low ≤ high)
{ int mid = (low + high) / 2 ;
  if (array[mid] == x)
      idx = mid ; break ;
else if (array[mid] < x)
      low = mid + 1 ;
else
    { high = mid - 1 ;
}
```

Interpolation Search:

```
while (low ≤ high)
{
    Pos = low + ((x - a[low]) * (high - low)) / (a[high] - a[low]);

    If (arr[mid/Pos] == x)
        idx = pos; break;

    else if (a[mid Pos < x])
        low Pos = Pos+1

    else if (a[Pos > x])
        high = Pos-1
```

Algorithm

① take array size - 1 as high and 0 index as low
② take search input x
③ Continue loop until low ≤ high
④ If arr[Pos] == x break loop element found
⑤ else if pos > x, high = pos + 1
⑥ if Pos < x, low = pos + 1

to search

Spring 2022

Why Binary search is better than ~~Jon~~ Linear search?

Time complexity of Binary search is O(log n)
" " " Linear " O(n).
That means binary search can search through much larger arrays much faster than linear. Suppose we have an array of 1 million element and we want to search for a specific element. In the worst case scenerio where the element we are looking for is at the end of the array. In linear we have to go through all the 1 million data whereas binary we can go more efficiently we need to perform $\log_2(1000000) \approx 2$ comparisons to find it which is faster than Linear.

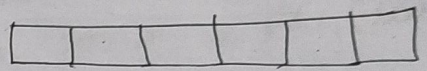
Why interpolation works better than Binary and why it takes more iteration than Binary. with example.

Fall-2022

Interpolation works better than binary but provide an instance where Binary works better and take less steps than interpol

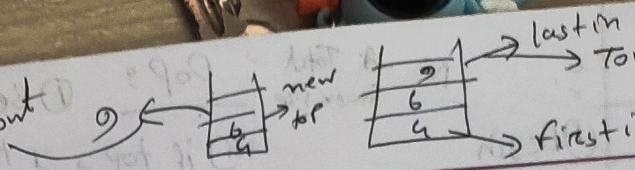
The common condition for binary and interpolation is that the list of the ~~condition~~ data/array has to be sorted (decending/ ascending). In addition to this interpolation search works best when the objects are arranged uniformly. if they become progressively less uniform the algorithm takes longer.

Interpolation:

a[0] 

(Spring - 2022) #

Interpolation is improved variant of binary search (20 more qns)



Stack

Stack: A stack is a list of element in which an element maybe inserted or deleted only at one end, called the top of the stack. Stack Sometimes known as Lifo. (Last in first out)

(i) Push() → Insert an element into the stack

(ii) Pop() → Deletes an element and deletations will be done from the top of the stack

(iii) top() → wants to see the element which is at the top

(iv) Peek() → returns the element of the stack without removing it

Stack overflow:

If stack is full we still want to push element is SS

Stack underflow:

If stack is empty we still want to pop element is SU

Top : N = Total stack size

(I) Begin

(II) If top = N then Print ~~Overflow~~ Overflow

(III) Insert new element Top = Top+1

PoP : (I) Begin

(II) if top = -1 then Print ("underflow

(III) Select stack[top]

(IV) top --

```
Int main() {
void push (int) {
if(top == (max stack -1))
{ Cout << "Overflow" ; }

else
{ cin>>x ; // taking an element
  top++ ;   // increasing top value
  Stack[top] = x ;   // Pushing the element inside
                     // the stack

}}
```

```
// PoP element
void pop() {
  If (top == -1)   // means stack is empty doesn't
                   // have any element so we can't
  { Cout << "underflow" << endl << "pop, it's underflow

  }

else {
  // Cout << Stack[Top] ; Checking which element
                          // is at top

  top-- ;

}

// Show output
void (Display) {
if(top == -1)
      Cout << "Stack is empty." <endl;

else {
      for(i=0; i< top ; i++)
          Cout << Stack[i] << endl;
```

Applications of Stack:

(i) Backtracking
(ii) Undo Redo
(iii) memory management

Define Push operation?

In Stack Push operation refers to the process of adding a new element to the top of the Stack. The newly added element becomes top of the stack. Other elements are shifted down by one position.

* Computer friendly Prefix & Postfix
* Human " infix

Arithmatic Expression

Infix: operator placed between 2 operands

2 + 4
operator

Prefix: operator placed before 2 operands

+ 2 4

Postfix: operator placed after 2 operands

2 4 +

() > -- > ^ > x, /, % > +, -

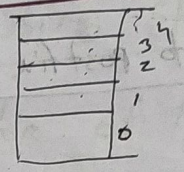
Infix to Prefix

1) $F = A + B * C$
$= A + [*BC]$
$= +A[*BC]$
$= x + ABC$

2) $F = (5+7) * 8 - 10$ // Infix
$= [+57] * 8 - 10$
$= *+578 - 10$
$= -x+57810$ // Prefix

3) $F = (A+B\hat{}D) / (E-F) + G$

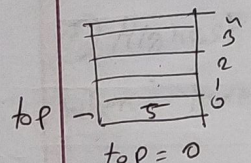
Construct a stack then show iteration and Diagram.



top = -1, max size = 5

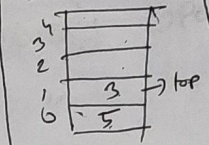
(i) is empty, since top = -1 stack is empty.

(ii) Push : 5



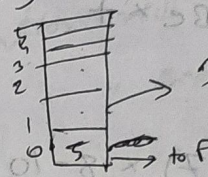
top = 0

(iii) Push 3



top = 1,

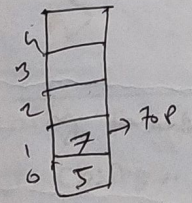
(iv) Pop



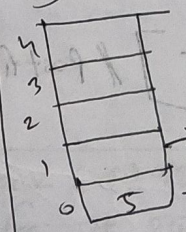
∴ top = 0

(v) Since top != max size stack is not full.

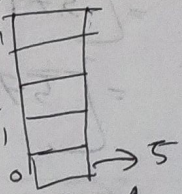
(vi) Push : 7



(vii) Pop

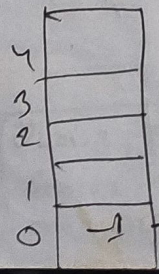


(viii) Pop



(ix) is empty, since top = -1 Stack is empty.

(x) Push -1



~~F = A+B~~

Infix to postfix

$F = A + B * C$

$= A + BC*$

$= ABC*+$

$F = (5+7) * 8 - 10$

$= [57+] * 8 - 10$

$= [57+8*] - 10$

$= [57+8*10-]$ // postfix

Using Stack

(I) Two operator of the same priority can't stay together in a stack.

(II) Highest priority operation will not stay in stack when lowest " " " occur.

(III) (+, *) all the operators between Paranthesis will be poped and place in postfix.

^ > ×, /, % > +, -

Opening Paranthesis যখন Closing ,, পাবে তখন ভিতরের মাঝের সব operator Pop হবে	Stack শেষ পর্যন্ত খালি না হলে Operator সুযোগ পেলে তবে Pop হয়	Stack এ যেই operator আছে তার Precedence নতুন Operator থেকে বেশি / সমান হলে আগের operator Postfix এ যাবে।
	Junior থাকলে Senior আসতে পারবে	Senior থাকলে Junior আসতে পারবে না, আগে Senior চলে যাবে

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

Infix	Stack	Postfix
A		A
*	*	A
(	*(	A
(	*((	A
B	*((	A,B
+	*((+	A,B
C	*((+	A,B,C
)	*(	A,B,C,+
-	*(-	"
D	*(-	A,B,C,+,D
)	*	A,B,C,+,D,-
/	/	A,B,C,+,D,-,*
E	/	A,B,C,+,D,-,*,E
		A,B,C,+,D,-,*,E,/

opparend এর আগেরটা stack এ নেই,

5*(6+2)-12/4

Infix	Stack	Postfix
5		5
*	*	5
(	*(	5,6
6	*(	5,6
+	*(+	5,6
2	*(+	5,6,2
)	*	5,6,2,+
-	-	5,6,2,+,*
12	-	5,6,2,+,*,12
/	-/	5,6,2,+,*,12
4	-/	5,6,2,+,*,12,4
	-	5,6,2,+,*,12,4,/
		5,6,2,+,*,12,4,/,-

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

Infix	Stack	Postfix
A		A
+	+	A
(	+(	A
B	+(	A, B
*	+(*	A, B
C	+(*	A, B, C
-	+(*, -	A, B, C, *
(	+(-(	A, B, C, *
D	+(-(	A, B, C, *, D
/	+(-(/	A, B, C, *, D
E	+(-(	A, B, C, *, D, E
^	+(-(^	A, B, C, *, D, E
F	+(-(^	A, B, C, *, D, E, F
)	+(-	A, B, C, *, D, E, F, ^, /
*	+(-*	""
G	""	A, B, C, *, D, E, F, ^, /, G
)	+	A, B, C, *, D, E, F, ^, /, G, *, -
*	*	""
H	*	"", H
		"", H, *

Last year d-2

(A) F = 8+(4*5-(27/3^2)*6)*7

Infix	Stack	Postfix
8		8
+	+	8
(	+(	8
4	+(	8, 4
*	+(*	8, 4
5	+(*	8, 4, 5
-	+(-	8, 4, 5, *
(	+(-(	8, 4, 5, *
27	+(-(	8, 4, 5, *, 27
/	+(-(/	8, 4, 5, *, 27

$8+(4*5-(27/3^2)*6)*7$

3	+(-(/	8,4,5,*,27,3
^	+(-(/^	8,4,5,*,27,3
2	+(-(/^	8,4,5,*,27,3,2
)	+(-	8,4,5,*,27,3,2,^,/
*	+(-*	8,4,5,*,27,3,2,^,/
6	+(-*	8,4,5,*,27,3,2,^,/,6
)	+	8,4,5,*,27,3,2,^,/,6,*,-
*	+*	>>
7	+*	8,4,5,*,27,3,2,^,/,6,*,-,7,*

∴ P = 8,4,5,*,27,3,2,^,/,6,*,-,7,*,+

Now calculate the value of P using stack.

2719 5+7
2719 57+

P	stack
8	8
4	8,4,
5	8,4,5
*	8,20
27	8,20,27
3	8,20,27,3
2	8,20,27,3,2
^	8,20,27,9
/	8,20,3
6	8,20,3,6
*	8,20,18
-	8,2
7	8,2,7
*	8,14
+	22

∴ value of P = 22

Using stack convert infix to postfix: Fall 22 a-3

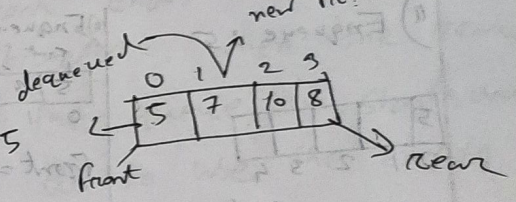
K+L-M*N+(O^P+Z)*W/U/V*T+Q

Infix	Stack	Postfix
K		K
+	+	K
L	+	KL
-	-	KL+
M	-	KL+M
*	-*	KL+M
N	-*	KL+MN
+	-+	KL+MN*
(	-+(	KL+MN*
O	-+(	KL+MN*O
^	-+(^	KL+MN*O
P	-+(^	KL+MN*OP
+	-+(+	KL+MN*OP^
Z	-+(+	KL+MN*OP^Z
)	-+	KL+MN*OP^Z+
*	-+*	KL+MN*OP^Z+W
W	-+*	KL+MN*OP^Z+W*
/	-+/	KL+MN*OP^Z+W*U
U	-+/	KL+MN*OP^Z+W*U/
/	-+/	KL+MN*OP^Z+W*U/V
V	-+/	KL+MN*OP^Z+W*U/V*
*	-+*	KL+MN*OP^Z+W*U/V*
T	-+*	KL+MN*OP^Z+W*U/V*T
		/V/T

queue

A queue is another special kind of list, where items are inserted at one end called rear and deleted at the other end called front.

Another name for a queue is FIFO (First In First Out)



Queue overflow.

when,

$r == n-1$

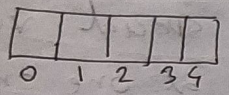
(maximum queue size)

when front == rear only last one element left. to delete this

$f = r = -1$

- enqueue করতে rear++
- dequeue " front++

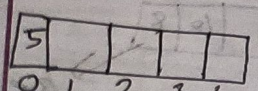
Construct a queue size of 5 then show insertion and diagram.



max size = 5, front = -1, rear = -1

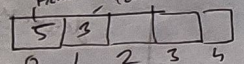
i) is empty since Both front and rear = -1

ii) Enqueue : 5



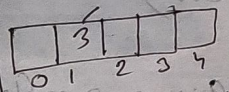
rear = 0, front = 0

iii) Enqueue : 3



Front = 0, rear = 1

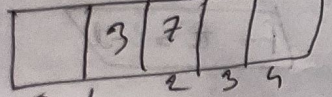
iv) Dequeue



front = 1, rear = 1

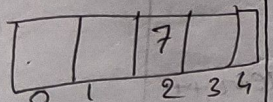
v) since rear + 1 != max size stack is empty

vi) Enqueue : 7



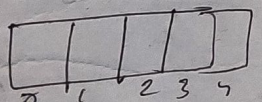
Front : 1, rear 2

vii) Dequeue



Front = 2
rear = 2

viii) Dequeue



front = -1, rear = -1

ix) as front is rear = -1 ~~stack~~ queue is empty

Enqueue -1

front = 0, rear = 0

Traverse array means Printing all the element inside array.

What is a circular queue?

A circular queue is a variation of the queue DS where the rear and front ends are connected forming a circular structure. When the rear reaches the end of the queue it starts over the queue. This allows us to use the empty space infront of the queue.

Define Enqueue in Queue?

The enqueue operation in queue used to add element to the rear of the queue.

Fall-22. ques-3

a) Describe advantages of Postfix expression over infix expression.

Sol: Post fix notation (Reverse Polish notation) RPN has several advantages over infix notation

(I) No parentheses Needed:

(II) Eliminate Ambiguity:

(III) Simplifies parsing

(IV) Simplifies expression manipulation,

(V) No operend Precedence concern

(VI) natural for computer,

Array

What is Array?

An array is a collection of elements stored in a sequential order by and index. It contains same data type. And it has an fixed size. It used for data storage and manipulation.

Advantages:

(i) Easy to access elements
(ii) efficient for accessing and iterating through elements
(iii) Can be implemented in stacks and queue.

Disadvantages:

(i) has fix size
(ii) insertion and deletation is expensive
(iii) Can lead to wastage of memory if array size is greater than required size.

Traversal: is the process of accessing each element sequentially. by loop / recursion.

Reversal: reversing the order of the element.

Array merging: Combining 2 or more array in a single array.

Array Split: Dividing and array into 2 or more sub array. we can split an array by specifying the index at which to split the array.

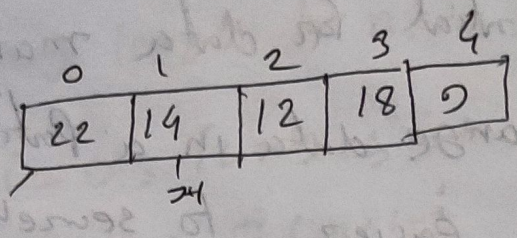
What is array insertion?

A process to add new element at specific index. or position. The existing elements are shifted to right to accomodate new element. If the array is full it's necessary to create a new array with larger size. O(n), n is the number of elements.

What is array deletation?

Process to remove an element from array. elements shifted to left to fill the gap. If the element is in the last index it doesn't need to be removed. It can simply be removed without shifting.

Bubble Sort




```
Int i, j;
for(i=0; i<n-1; i++)
{ for (j=0; j<=n-1-i; j++)
  { if (arr[j] > arr[j+1])
    { int temp;          // swapping
      temp = arr[j];
      arr[j] = arr[j+1]
      Arr[j+1] = temp;
    }
  }
}
```

Why Sorting is essential for data manipulation.

= Sorting is essential for data manipulation. it helps to arrange data in a particular order. making it easier to search, retrieve and analyze data. It makes the data more organize and reduce complexity. Sorting algorithms are used in many applications including database, search engines et.

Remove all the duplicates elements write a pseudo code:

(I) initialize an empty array.
(II) loop through each element in input
(III) for each element check if it already exists
(IV) If it does not append it to result array.
(V) If it exists do nothing
(VI) After loop print result. which as only unique elements.

Bubble sort : ascending order.

(I) Set array length to n.
(II) For i=0; i<n-1 ; i++
(III) For j=0; j<=n-1-i
(IV) If arr[j] is > arr[j+1] then swap
(V) Continue to next element.
(VI) Array is in assending order.

uninformed/Blind

Depth first search

DFS explores as far as possible along each branch before backtracking.
It uses Stack to remember the next vertex to visit from the current vertex

we use Stack to keep track of nodes that we have visited but not yet explored.

When?

(i) want to visit nodes as deeply as possible along each branch.

(ii) interested in path or finding cycles in the graph.

(iii) memory usage is concern. Dfs is more memory efficient compared Bfs.

Dfs is used for task involving all possibilities, like maze solving, topological sorting or cycle detection

Breadth First Search

Explores all the vertices at the present depth before moving on to to vertices at the next depth level.
It uses Queue

When?

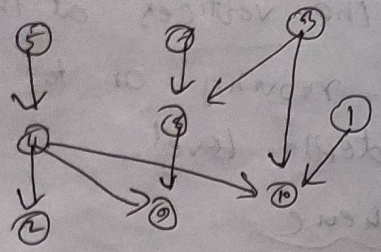
(i) want to find the shortest path between 2 nodes in unweighted graph.

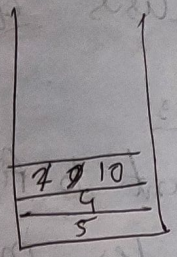
(ii) need to visit all node in the graph with minimum number of edge

(iii) memory usage is not concern.

Bfs is used for finding shortest path. or needs to explore nodes in layers such as web crawl or social network analysis. where want to find, connection with fewest intermediate.

DFS





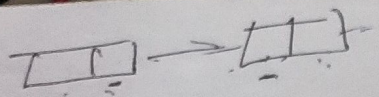
a) 5, 11, 2, 9, 10.

ind ; a=10 int a = to

Creating Linklist Assigning element

At the beggining, End, Specific Position.

```
# Include <stdio.h>
# Include <stdlib.h>
Struct Node {
    int data;
    int * next;
};
int main {
  Struct Node * a = malloc (sizeof (struct Node));
Struct Node * b = malloc (sizeof (struct node));
Struct node * c = malloc (sizeof (struct node));
  Struct Node * head = Null;
```



a → data = 100;

b → data = 200;

c ⇒ data = 300;

a → next = b

b → next = c

c → next = Null.

head = a;

Struct Node * newNode = (malloc (sizeof(

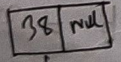
// insertion at beginning.

Printf (Enter your element);

Scanf ("%d", & newNode → data);

newNode → next = head; // we've updated the new node next to head

// And updated the head to newnode



So new node becom first node. which is linked with 2nd

head = Newnde;

ptr = address of current node

Struct Node * ptr = head;

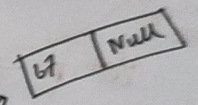
while (ptr != Null)

{ Printf (. , ptr → data);

ptr = ptr → next;

}

// insertion at the end



Struct node * Endnode = malloc (sizeof (struct node);

{ Printf (Enter end elem);

Scanf ("%d", & Endnode → data); }

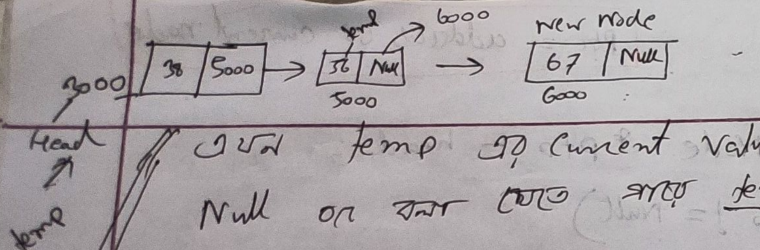
Struct Node * temp = malloc sizeof (struct Node));

*temp = head. // যখন head ও যে node point করছেন আমরা temp ও সে node point করছেন।

While (temp → next != Null) { // traversing through the list

temp = temp → next;

}



এখন temp এর Current Value হবে Null or বলা যেতে পারে temp এখন

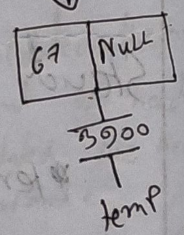
শেষ index এ।

এখন যদি temp এর শেষ index → next এ inserted end Node কে দিয়ে দেই তাহলে End insertion সম্পন্ন হবে

temp → next = EndNode.;

// Insert in given location

new pointer for new inserted node ← Struct Node * giveloc = malloc(sizeof(Struct Node));



Int Pos ; → যে idx এ data দিব

Printf (give idx location) ;

Scanf ("%d", &Pos) ;

If (Pos > counter) ;

{ Printf (Error) ;

}

Else {

Struct Node * temp2 ; int i = 1 ;

temp2 = head ;

while (i < Pos) → যেই Pos এ insert করব ঐ পর্যন্ত যাবে node

{ temp2 = temp2 → next ; → temp2 এ থাকবে যেই Pos এ insert করব তার আগের node

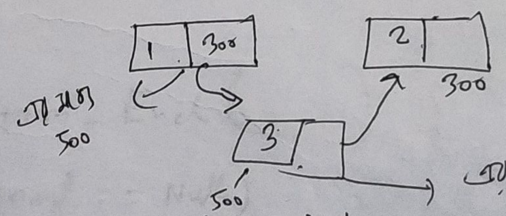
i++ ;

}

Printf (Enter new data) ;

Scanf ("%d" & giveloc → data) ;

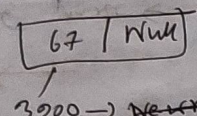
givloc → next = temp2 → next ;



temp → next = givloc ;

NB: we only have an head pointer and we can access the list only through head pointer

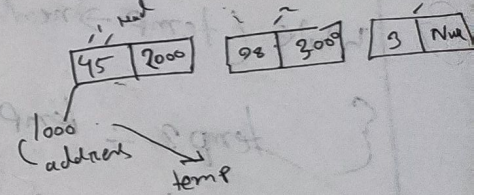
① Create new node



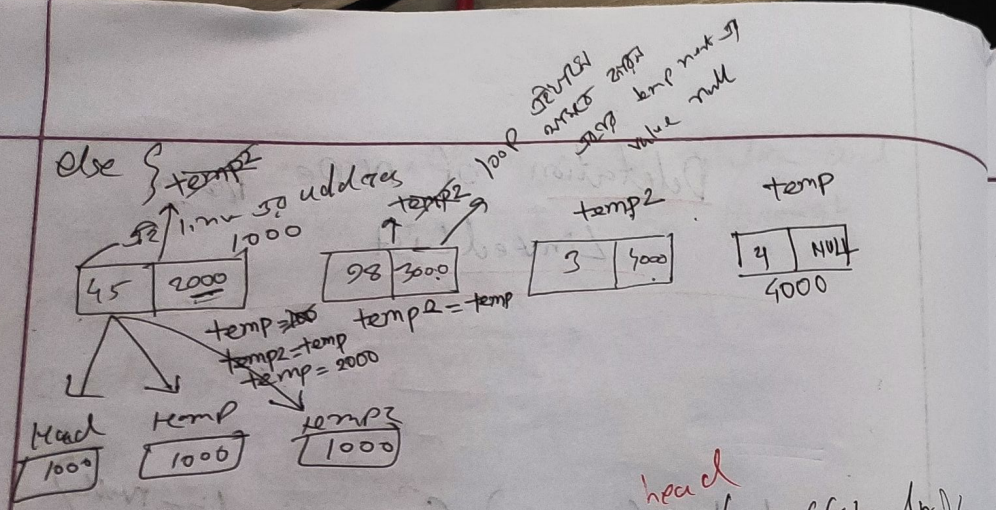
Step-2: Traverse; we need to have another pointer variable which will traverse through the Linked list. and will stop right before the position where we want to add the new node.

Deletation of node from Linked List

```
# Del_at_beg( ) { If (head==null) { List is empty }
else {
Struct node* temp.; *head;
    temp = head;
    head = head -> next;
    free (temp);
}
```




```
# Del_from_End( ) {
if (head == Null)
    Printf (list is empty);
else if (head -> next == null) → যদি linklist এ শুধুমাত্র একটি node থাকে. তাহলে
{ free (head);
    head = Null; }
```



Struct Node * ~~temp~~ temp = ~~malloc (sizeof (struct node))~~; head

=> => * temp2 = ~~malloc (sizeof (struct node))~~; head

While (temp -> next != Null) → traversing the list to the last node

{ temp2 = temp;

temp = temp -> next;

}

temp2 -> next = Null;

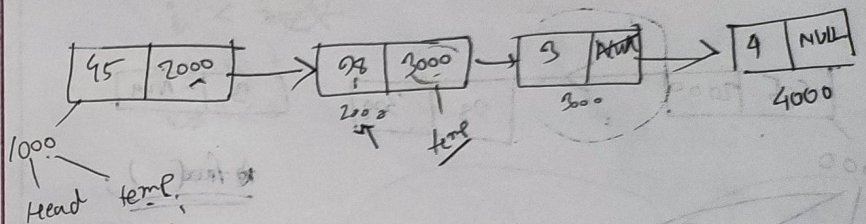
free (temp);

temp = Null;

}

temp2 কে সর্বশেষ node update.

Another version delete last node;



Struct Node * temp = head;

While (temp -> next -> next != Null) {

{ temp = temp -> next;

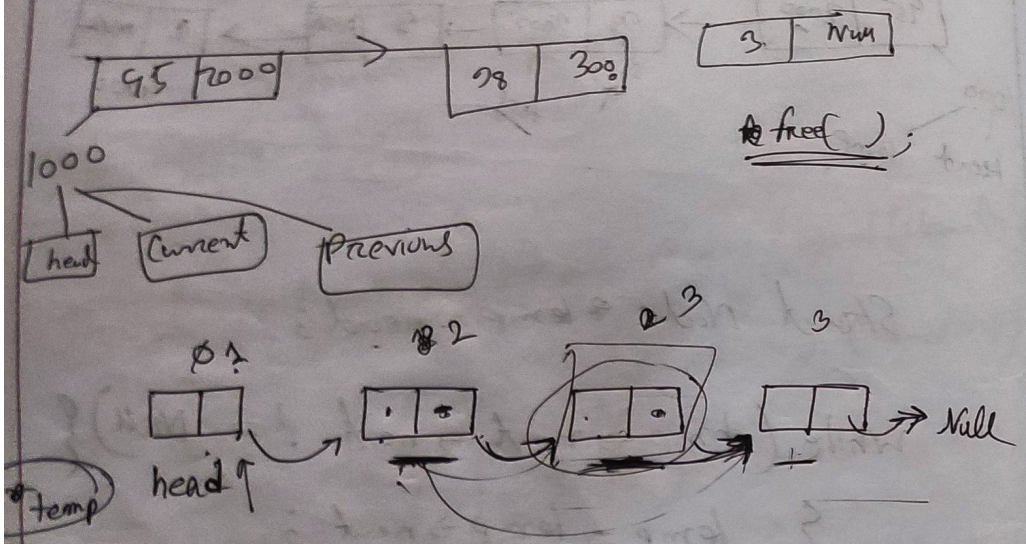
}

free (temp -> next);

temp -> next = Null;

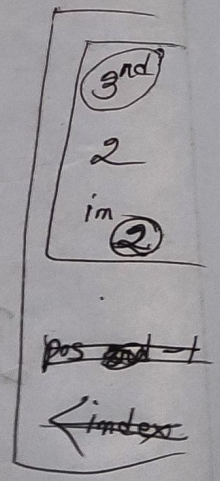
}

Deleting node at particular position;




```
for ( i=0; i < pos-1 ; i++) {
    if ( temp->next!=NULL) {
        temp = temp->next;
    }
}

temp->next = temp->next->next;
```



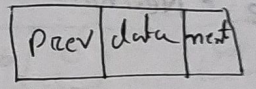
pos = 2

i < 2

0 < 1

1 < 2

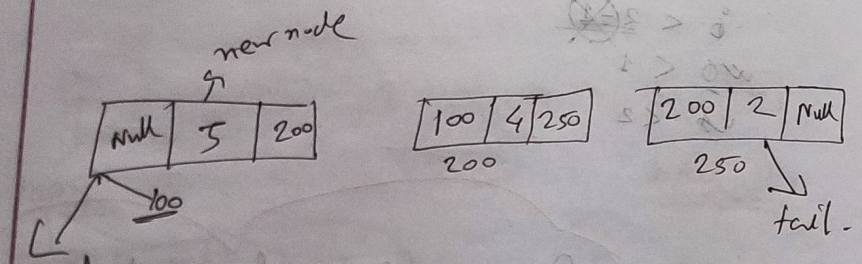
Doubly Linked List



Insertion at the beginnin

```
Struct Node {
    int data;
    struct node * next;
    struct node * Prev;
};
```

Struct node * head, * tail ;



creating new node

void createDouble ()

{ Struct node * newnode = malloc (sizeof (Struct node));

Entering data into newnode

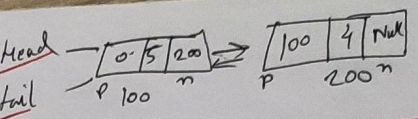
scanf ("%d", &newnode → data);

newnod → Prev = Null ;

newnode → next = Null ;

if (head == 0) { head = tail = newnode ; Currently new node heads tail এবং new node কে indicate করছে। creating 1st node.

} else {



tail → next = newnode ;

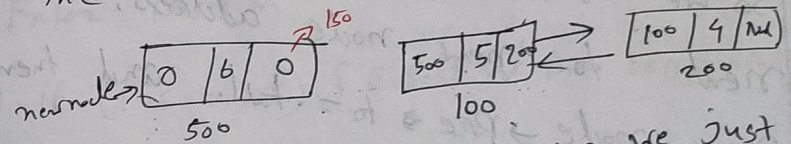
newnod → Prev = tail ;

tail = newnode ;

// creating 2nd node.

}

Now to insert a new node at the beginning into the List we have to create a new node and assign Null into the Prev and next node.



As we gonna insert at beginning we just have to update the Prev node of new second node and next node of new node.

head → Prev = newnode

new node → next = head

head = newnode ; // return node কে এখন head node বানালাম

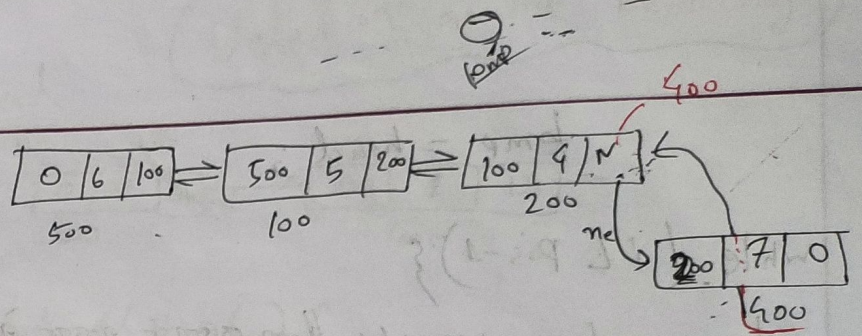
Now Insert at End

For inserting at the end again we have to create a new node. And as head is already pointing at beginning node. we used tail for pointing at the end of the node. So we just have to update the tail→ next to new node address, and

new node → prev = tail. and then

Update tail = newnode.

```
tail → next = newnode;
newnode → prev = tail;
tail = newnode;
```



Insert at specific Position

(I) Create a new node,

(II) Enter position

(III) if (pos == 1) {Insert at the beginning}

if

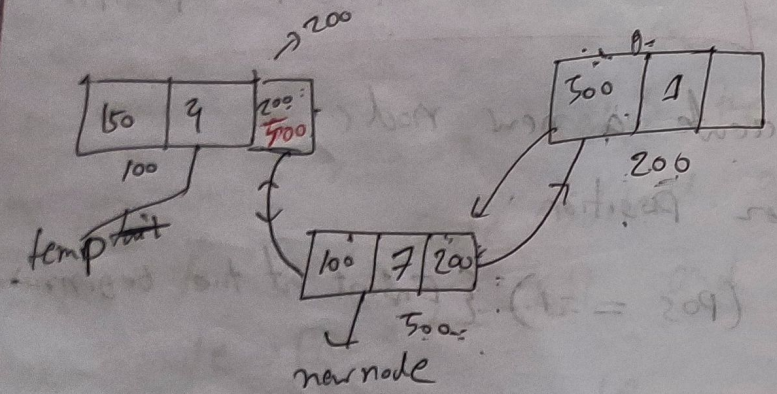
While (i < pos - 1) // যত pos এ insert করতে চাচ্ছি pos এর আগে যেতে হবে

Struct node *newnode, *temp, *tail.

temp is for traversing through LL

```
temp = head;

while (i < pos-1) {
    temp = temp -> next;   // loop যেখানে থামবে ঐ node টি traverse করে থাকবে
    i++;
```

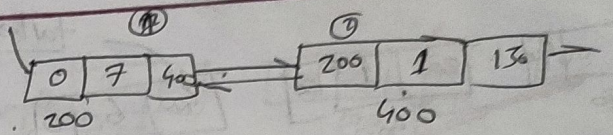



```
newnode -> Prev = temp
newnode -> next = temp -> next
temp -> next = newnode
newnode -> next -> Prev = newnode.
```

200

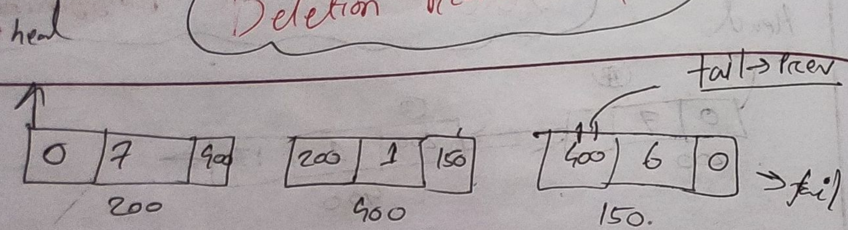
Deletion at the Beggining

Head



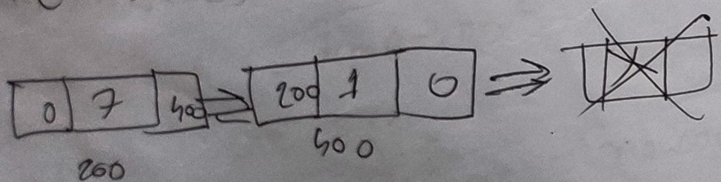

```
Void delBeg() {
    Struct node *temp;
    if(head == 0)
        { ("list is empty") };
else, {
    temp = head;
    head = head -> next;   // head এর মধ্যে current head এর পর node টি নিয়ে নিলাম।
    head -> Prev = Null;   // end node (new head) এর Prev node null আগের node এর সাথে connection নেই বোঝা
    free (temp);
```

Deletion from end

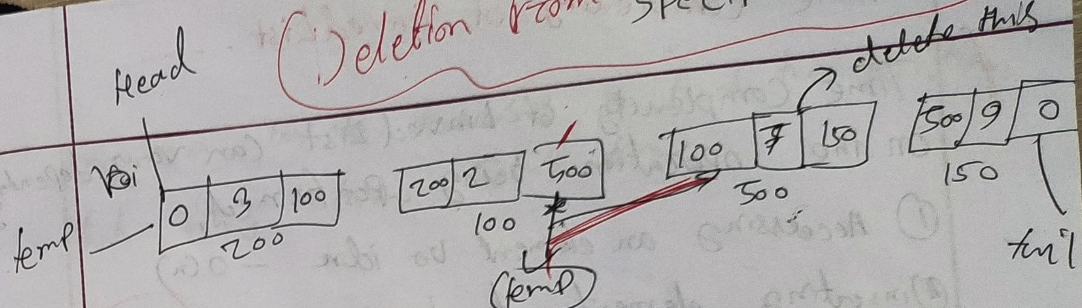



```
void delend() {
  Struct node *temp;
  if (tail == 0)
     { "List is empty" }
  else {  temp = tail;
     tail->Prev -> next = Null;
  tail = tail -> Prev;
  free(temp);
```

// tail Prev এর Node এখনো ওর next. Basically 400 এর next কে indicate করছে।



Deletion from Specific index




```
void del from Pos () {
   Struct node *temp;    temp = head;
   Scanf("%d", Position);
   While (i < pos) {
     // temp = head;
     temp = temp->next.
     i++
   }
   temp -> Prev -> next = temp -> next
   temp -> next -> Prev = temp -> Prev
   free(temp);
```

// traverse করে ওর Node delete করবো

Time Complexity of LinkedList.---

Time Complexity of LinkedList can vary depending on the operation being performed,

(i) Accessing an element by index — O(n)

(ii) inserting element at beginning — O(1)

(iii) " " at end — O(n)

(iv) " " at position — O(n)

(v) removing element from beginning → O(1)

(vi) " " " end — O(n)

(vii) " " " position — O(n)

(viii) searching for an element — O(n)

accessing element in LL takes O(n) because we have to traverse the list for finding desired element. However adding and removing at beginning a term O(1) because we just have to update the head pointer. adding or removing elements at the end of the List or given position requires us to traverse to find the location that why take O(n) time.

Binary Search Tree

(I) First element is root

(II) left child root theke choto হবে

(III) Right " " " বড় হবে.

(IV) ~~Searching~~ কোনো element যদি root থেকে ছোট হয় তাহলে আমাকে সেটি বামে যাবে, এইভাবে খুঁজতে থাকবে যতক্ষন না পর্যন্ত leaf node এ পৌঁছায়.

Inorder gives ascending order.

Delete Data from BST

(I) node Can have 0 child → Can directly delete.

(II) " " 1 " → child root এর সাথে connect

(III) " " 2 " → 2 situation

Inorder Predecessor

(I) largest element from left subtrees

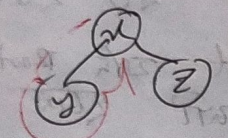
Inorder Successor

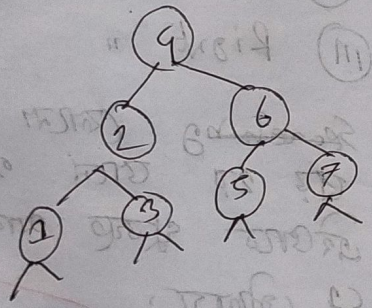
(I) Smallest element from right subtree

(1) Inorder → bottom left, root, right

(2) Postorder → Left, Right, Root

(3) PreOrder → root, left, right





(1) PreO: x, y, z

(3) Post: y, z, x

(2) In: y, x, z

↓

Sorted

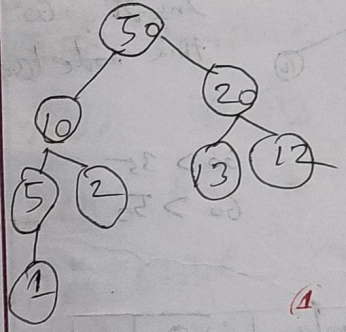
Heap

Data is left focused

Heap: A complete binary tree

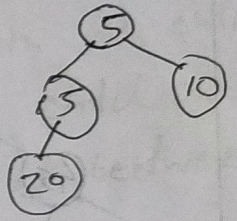
Max Heap

(1) Children will always be smaller than Parent (root)



Min heap

Children Parent থেকে বড় হবে



(I) Node = i

(II) Parent (i) = $\lfloor \frac{i}{2} \rfloor$ → floor value → যেকোনো node এর Parent বের করতে

(III) left child (i) = 2×i → left child বের করতে

(IV) right child (i) = 2×i+1 → right child বের করতে।

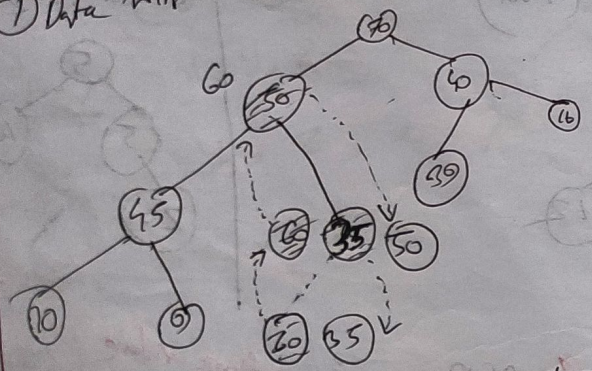
Heap insertion

For max heap

(I) Find the position and insert element

(II) while(new element > it's Parent) exchange them,

(I) Data will always inserted from left leaf node, then right



Insert 60 in this node tree.

60 > 35
60 > 50

A	70	50	40	45	35	39	16	10	9	60
	1	2	3	4	5	6	7	8	9	10

(I) will added at the left node

(II) Now check if new element > Parent if true then swap

Max Heap Deletion

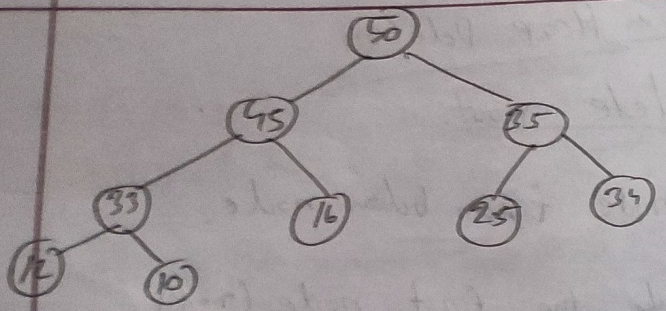
always delete root.

Delete a node from below node.

(I) First delete the first node (root)

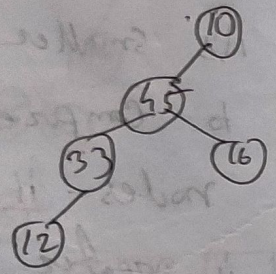
(II) then put the last element as root node.

(III) The compare root node with it's child node if it's smaller than child nodes then try to compare node between this child nodes if between 2 child One child is greater than other child then root will be swaped with that greater child.

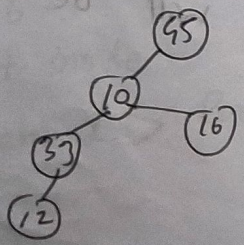


(I) Delete root 50

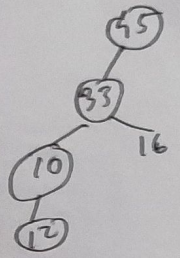
(II) last data is 10. because first add left side 12 then 10 is added.



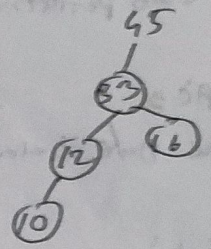
(III) Compare new node with it's child node child is greater, now swap



(IV) Now Compare 10 with it's childs. 33 is greater than 16 so, 10 will be swapped with 33.



again compare and swap with 12



Insertion → Bottom to root
Deletion → root to bottom

Min heap Deletion

(I) Can directly delete only root → O(log n) worst

(II) right most node → O(1) Best

for n element complete tree → nlogn

(III) Between 2 child root will go to the lesser child