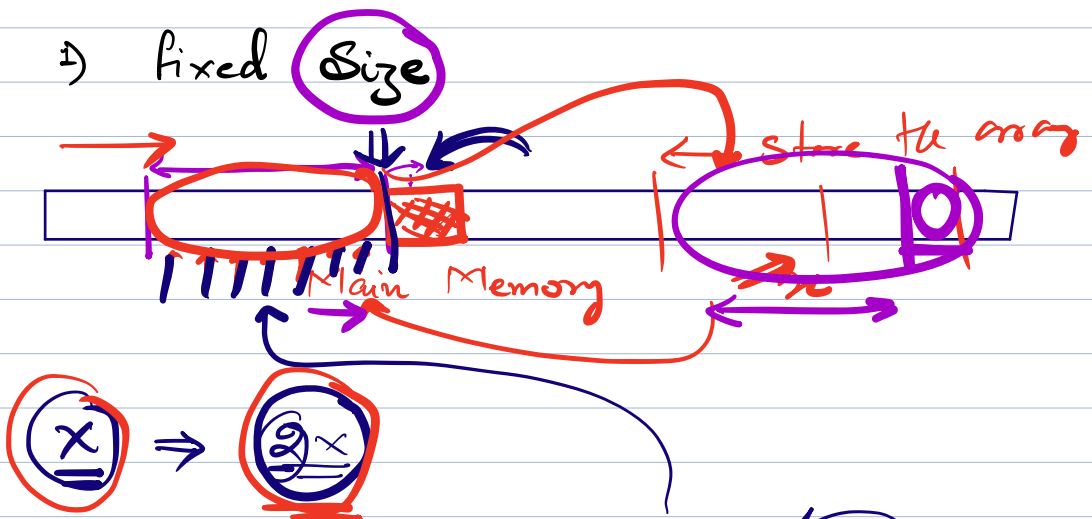
Arrays ⇒ random access ⇒ O(1) ×

⇒ contiguous memory location

Constraints

1) fixed Size



vector<int> = ( )

list<int> = ( )

Linked List

Kirana Shop

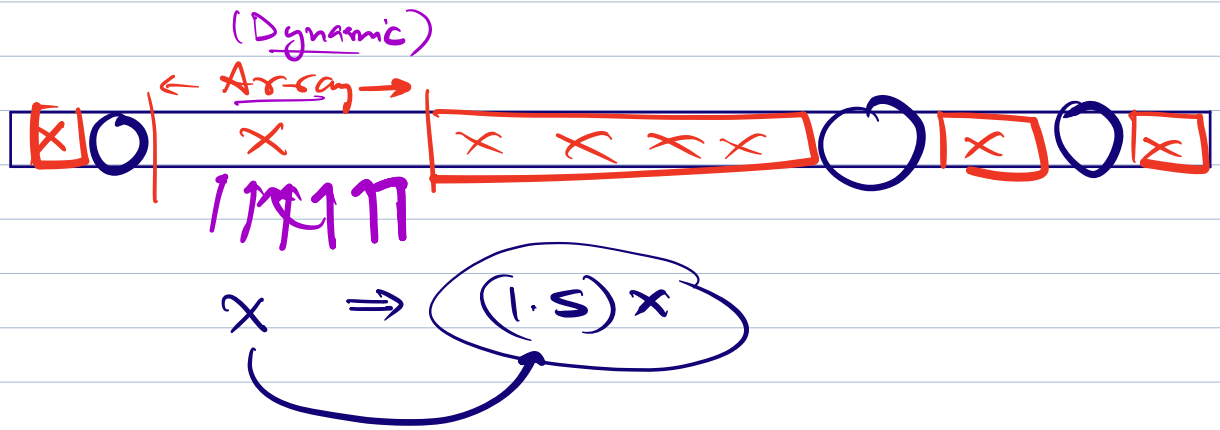
⇒ list of all the purchases

Drawback of Array.

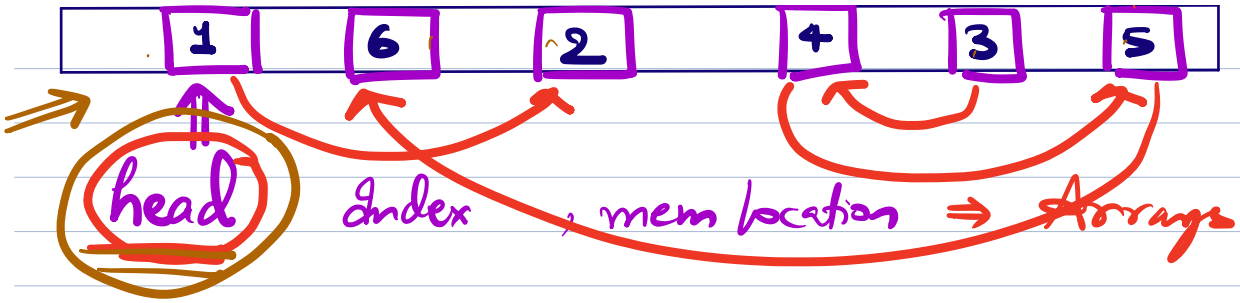
1) Because of contiguous memory allocation, size cannot be altered or is very expensive

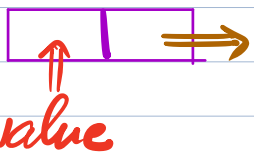
2) Inserting new when array is full is expensive

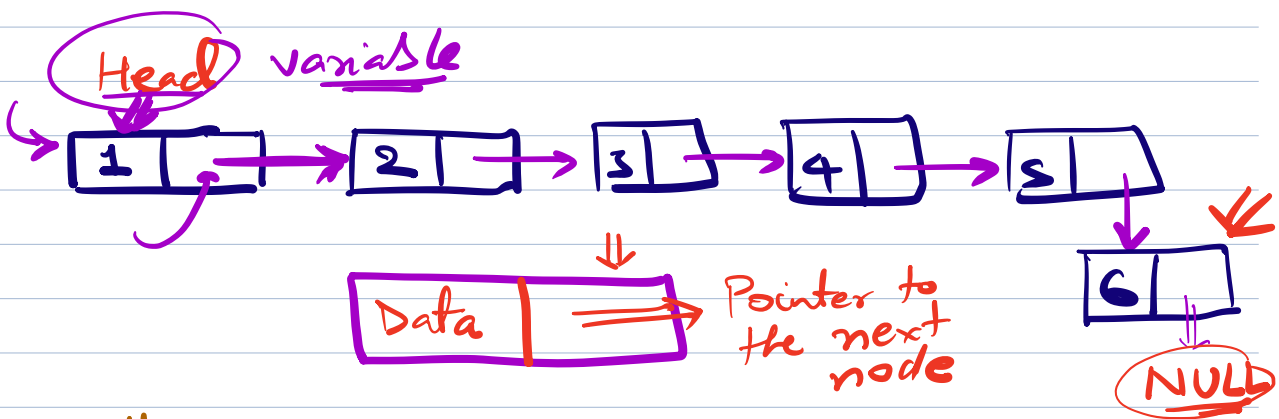
3) Deleting element does not free the space.



⇒ Elements in linked list are not stored in contiguous memory locations.



Node ⇒  memory location of the next node in the linked list



C

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

Java / C#

```
class Node {
    int data;
    //Reference next object
    Node next;
    Constructor Node(int d) {
        data = d;
    }
}
```

C/C++

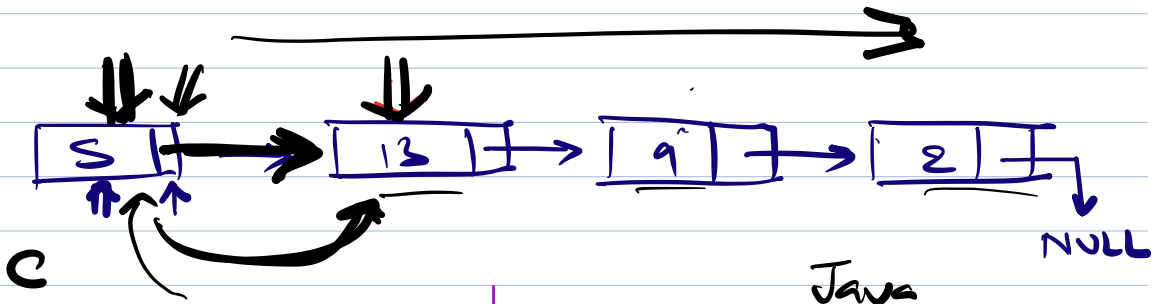
```
struct node* one = malloc(sizeof(struct node));
```

Object Oriented Language

// Constructor

```
Node nodeOne = new Node(5);
```

Eg



C

```
main() {

struct Node* head
 = malloc(sizeof(struct Node));

head -> data = 5;
head -> next = NULL

Struct Node* nextNode
 = malloc (        )
nextNode -> data = 13
next Node -> next = NULL

head -> next = next Node;
```

Java

```
main () {

Node head = new Node(5)

Node nextNode
  = new Node (13);

head.next = nextNode;

}
```

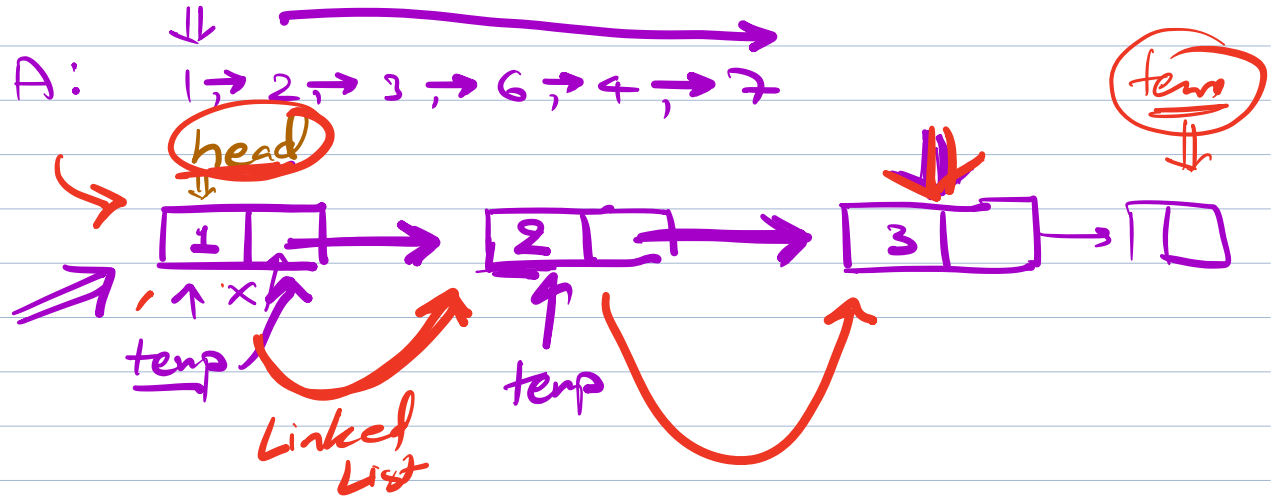
Task

Given an array A. of integers
Write a function which creates
a Linked List from the array
& return the head of the LL.

Code

- C — struct Node*
- Java — Node

```
Node createLinkedList (int[] arr) {
⇒   Node head = new Node(arr[0]);
    Node temp = head;
    for (i = 1; i < arr.size; i++) {
        Node newNode = new Node(
                                arr[i]);
        temp.next = newNode
        temp = temp.next;
    }
    return head;
}
```



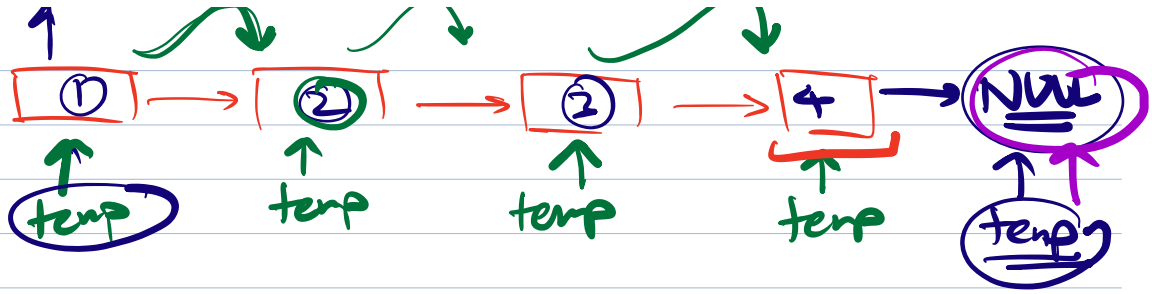
Q. Given a head pointer of a LL
Write code to iterate over LL.
print all ele of LL

if (head==NULL) return

Soln

```
int iterateLL( Node head) {
    Node temp = head;
    count=0;
    while ( temp != NULL) {
        print ( temp.data);
        count++;
        temp = temp.Next; (i++)
    }
    ret count;
}
```

head



1, 2, 3, ○

temp.next == NULL

temp != NULL

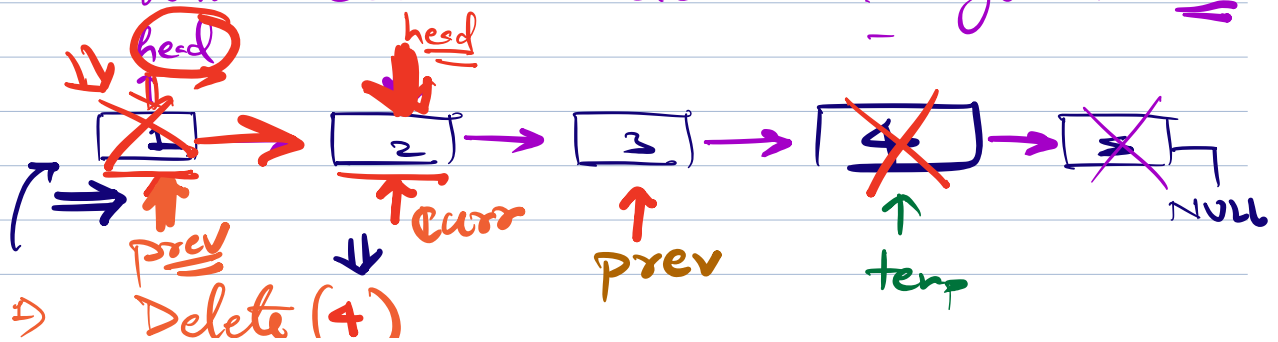
LL (N) ⇒ O(N)

i^{th} element ⇒ Iterate i times

O(i)

Q) Given head points to a LL.

Write code to delete a given node



⇒ Delete (4)

1) Head

```
temp = head
head = head.next;
delete (temp);
```

2) Last Node

3) Middle

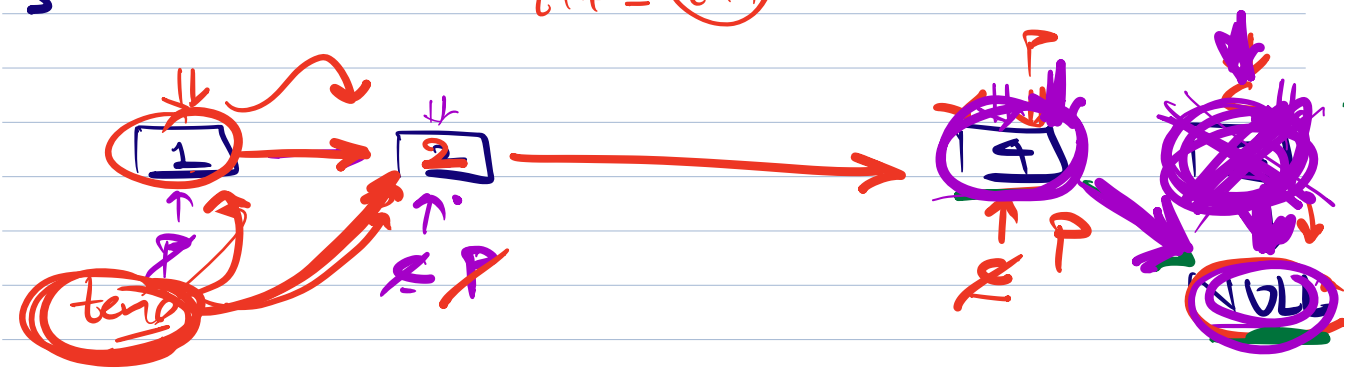
```
bool deleteNode (head, key) {
    if (head.data == key) {
        Node temp = head;
        head = head.next;
        delete (temp);
        return;
    }
    Node curr = head.Next;
    Node prev = head;
    while (curr != NULL) {
```

head

```
if (curr.data == key) {
    prev.next = curr.next;
    delete(curr);
    return;
}
curr = curr.next;
prev = prev.next;
}
}
```

any other node is deleted

i++ = (i+1)



delete (3) ⇒

delete (5)

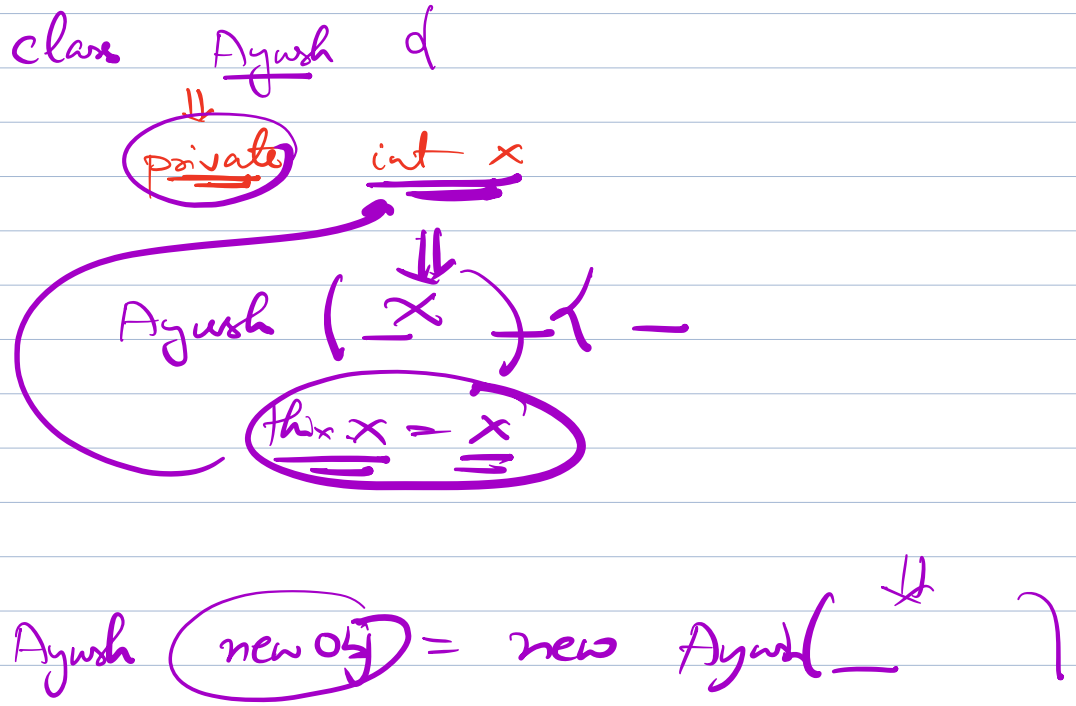
1 → 2 → 4 → NULL

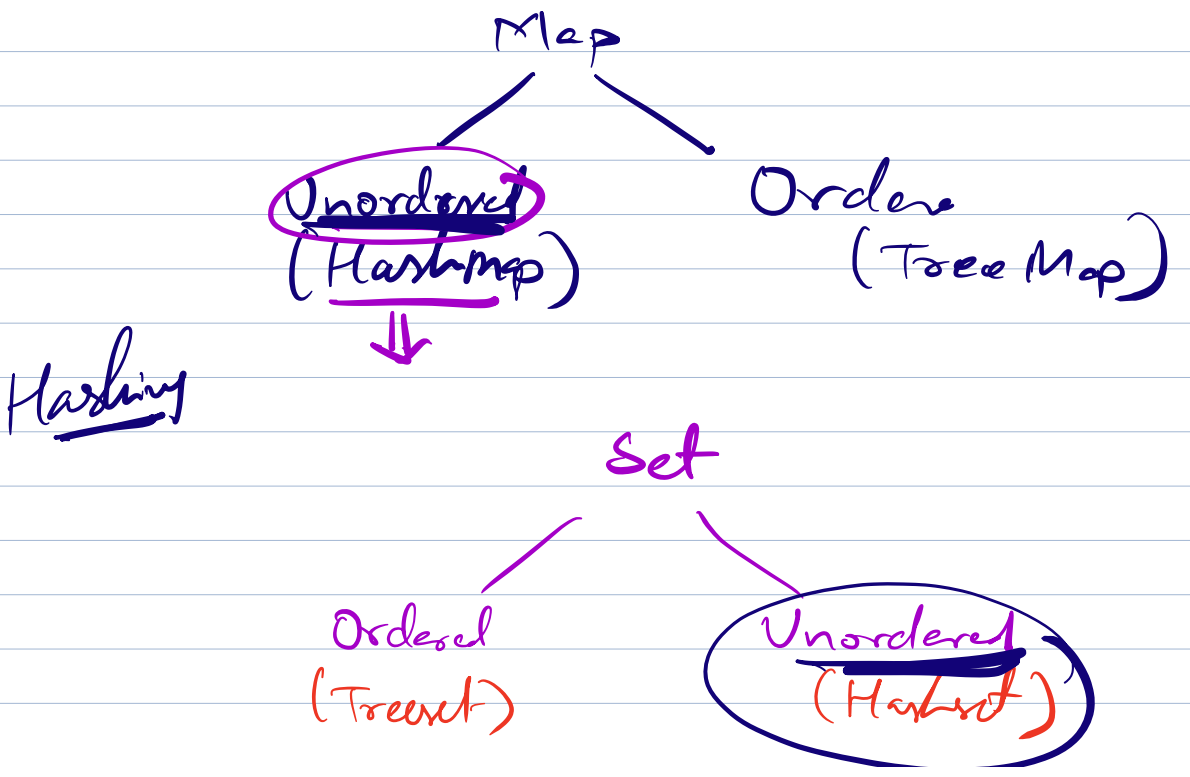
4.next = NULL

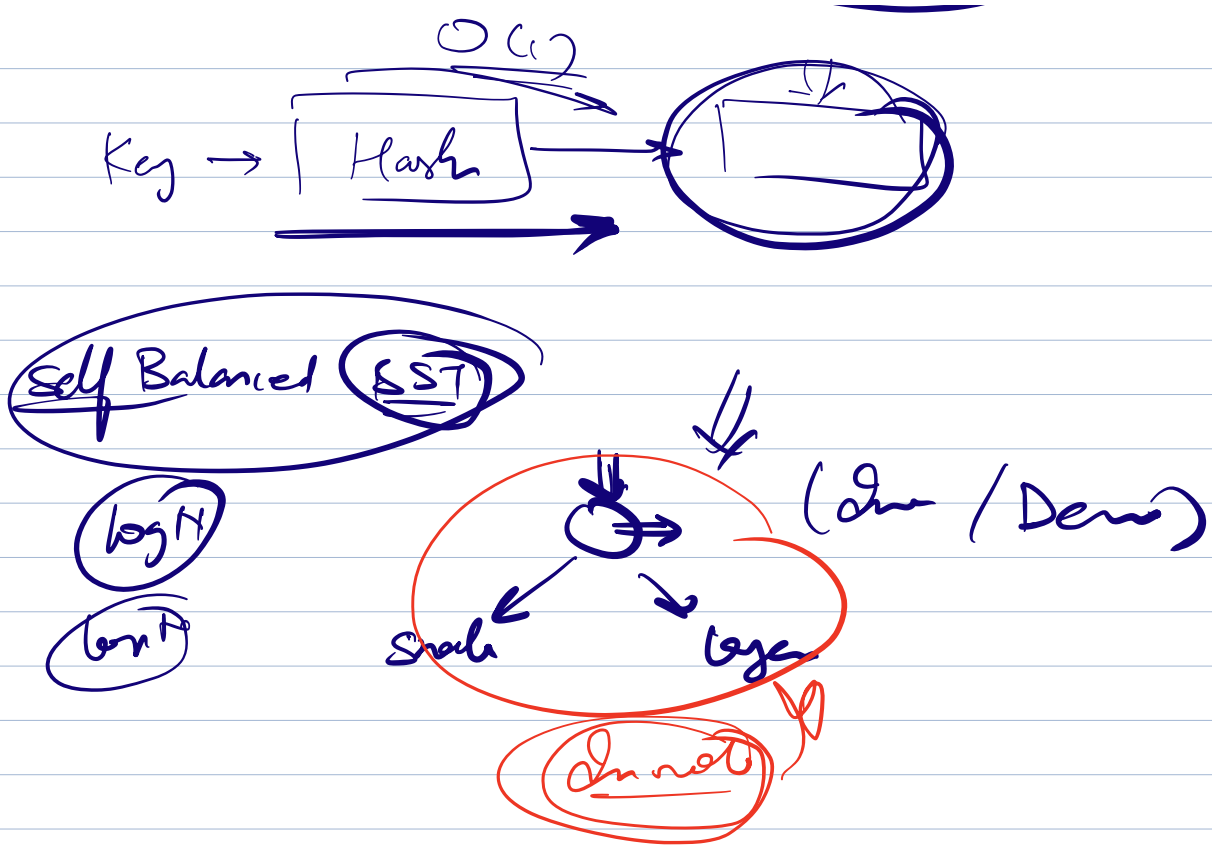
5.next = NULL

4.next = 5.next

print (4.next)


class Ayush {
private int x
Ayush (x) {
this.x = x
}
Ayush obj = new Ayush( )


Map
Unordered
(Hashmap)
Ordered
(TreeMap)
Hashing
Set
Ordered
(Treeset)
Unordered
(Hashset)



$$T(n) = T(n-1) + T(n-2)$$

$$T(n-1) = T(n-2) + T(n-3)$$

$$T(n) = 2T(n-2) + T(n-3)$$

$$T(n) = 2\left(T(n-3) + T(n-4)\right) + T(n-3)$$

$$T(n) = 3\left(T(n-3)\right) + T(n-4)$$