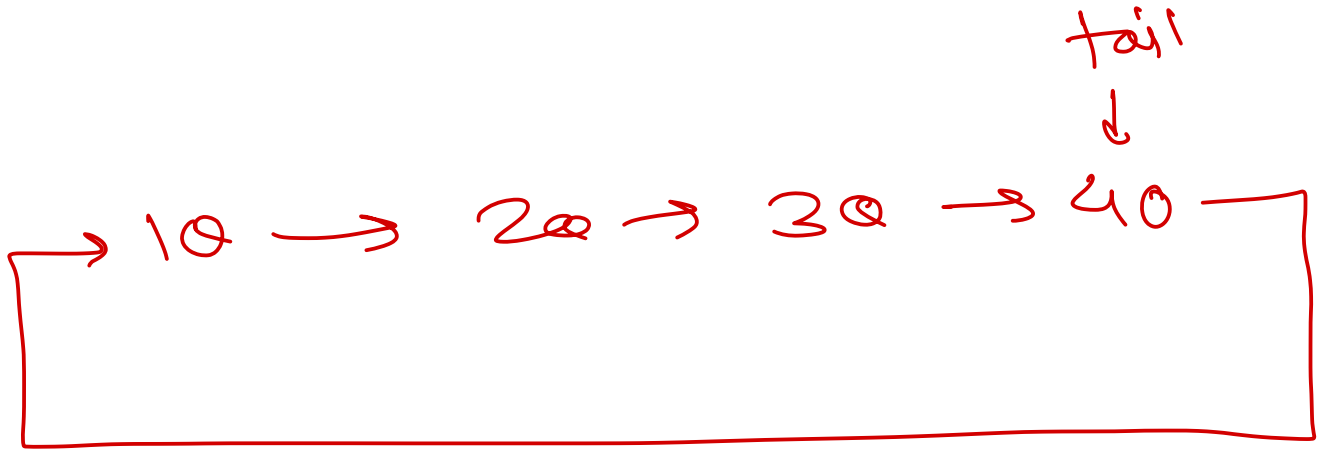
Data Structure & Algorithms

Sunbeam Infotech

Singly circular list with tail pointer.

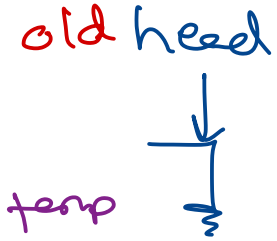


display():

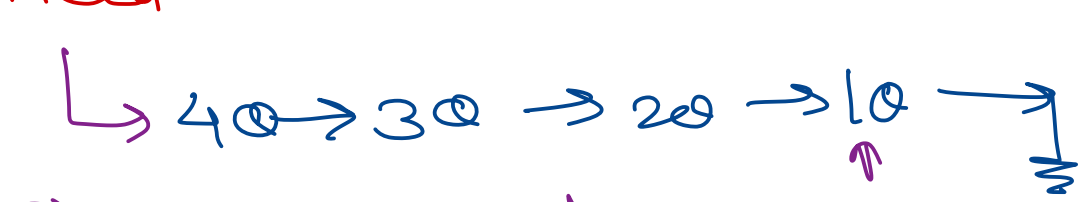
```
trav = tail->next;
do {
    cout<<trav->data;
    trav = trav->next;
} while(trav != tail->next);
```

Linked List

singly linear list → reverse the list.



① consider cur list as old (oldhead).
② consider new list as empty (head).
③ delete first node of old list.
④ add it at start of new list;
⑤ repeat 3 & 4 until old list is completed.



① oldhead = head;
② head = NULL;
while (oldhead != null) {
③ temp = oldhead;
oldhead = oldhead → next;
④ temp → next = head;
head = temp;
3 ⑤ repeat 3 & 4

Linked List singly linear list → display reverse → O(n²)

head

10 → 20 → 30 → 40 }

① Count num of ele in list(n).

② for (pos=n ; pos >=1 ; pos --)

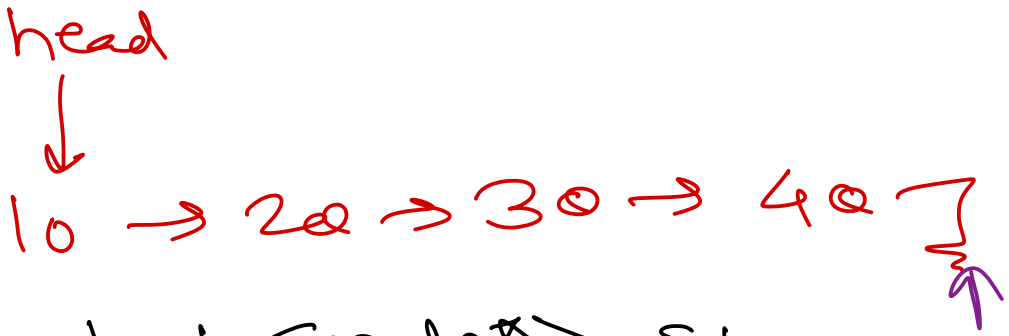
③ trav upto pos

④ print trav.

```
int n = 0;
for(trav=head ; trav!=NULL ; trav=trav->next)
    n++;
for(pos=n ; pos >=1 ; pos--) {
    trav=head;
    for(i=1;i<pos;i++)
        trav=trav->next;
    cout << trav->data << " ";
}
```

Linked List

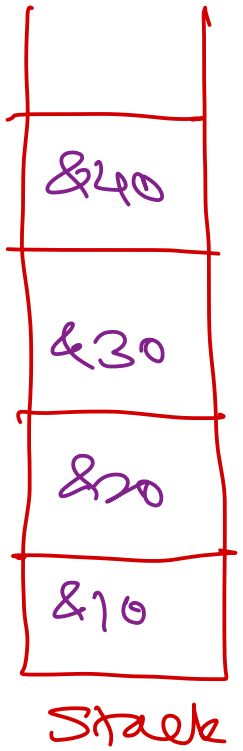
singly linear list → display reverse → O(n)




```
stack<node*> s;
node *trav=head;
while (trav!=null){
    s.push(trav);
    trav=trav->next;
}
while(!s.empty()){
    trav=s.top(); s.pop();
    cout<<trav->data;
}
```

① traverse & push each node addr on stack.

② while stack is not empty, pop each & display its data.



Linked List

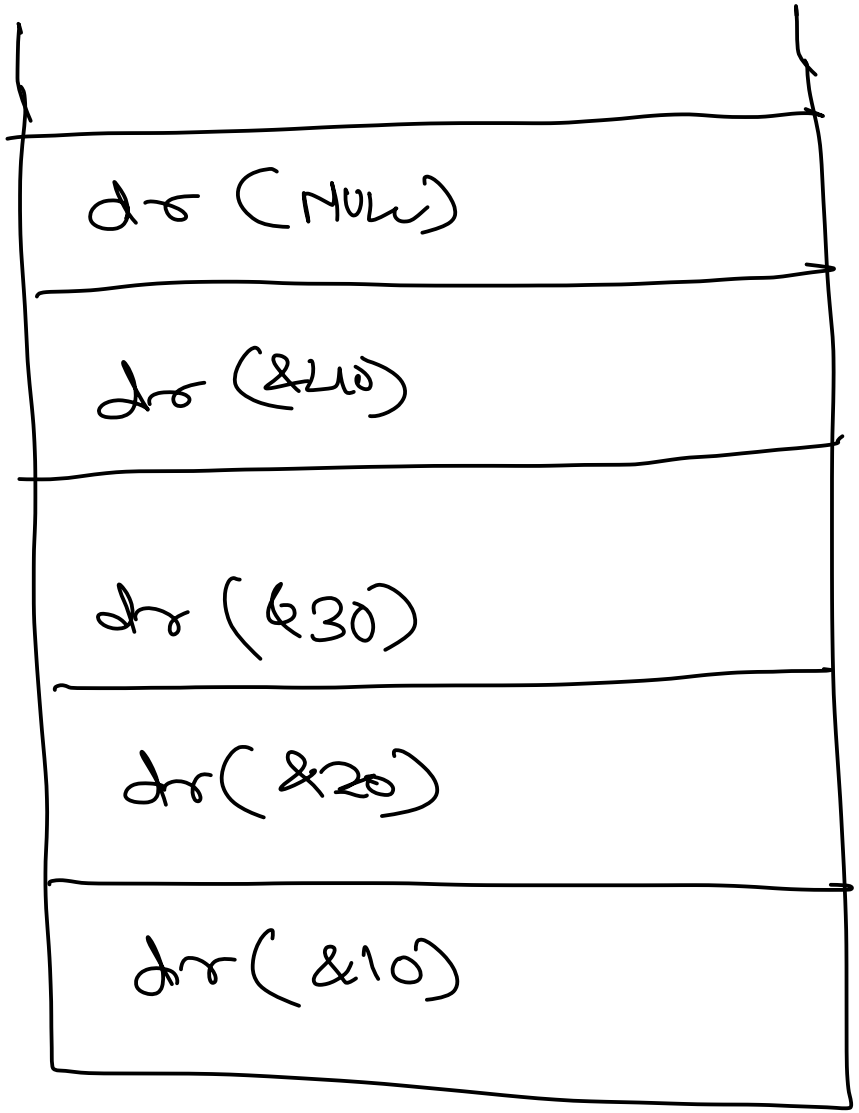
Singly linear list → reverse display - using recursion.

head

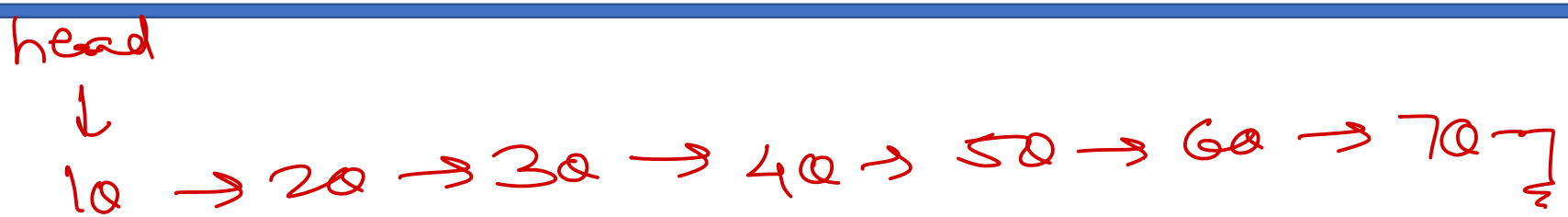
10 → 20 → 30 → 40 ⏚

```
void display_rev(node *trav) {
    if(trav==NULL)
        return;
    display_rev(trav->next);
    cout << trav->data;
}
```

call: display_rev(head);



Linked List singly linear list → find mid



① Count elements of list.

② Traverse upto count/2.

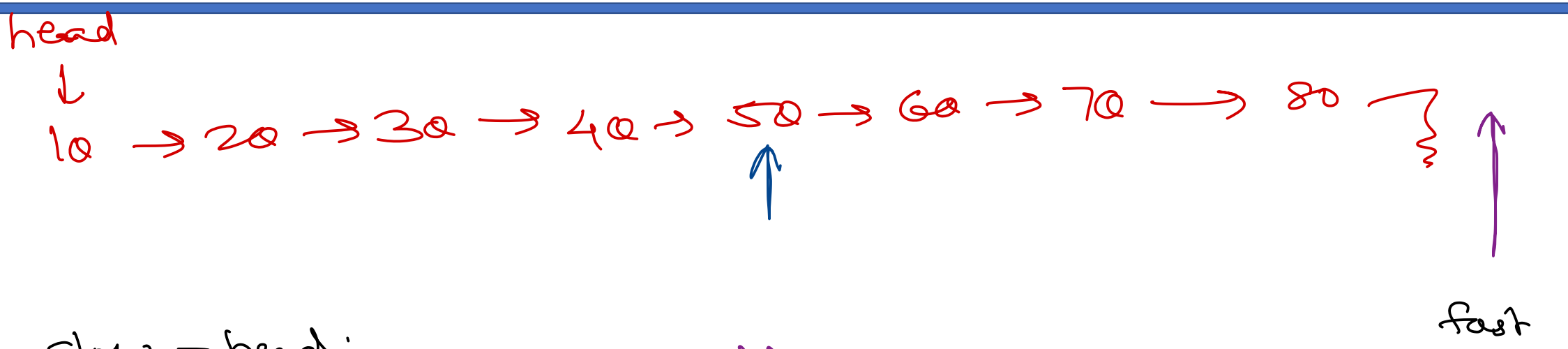
```
cnt = 0;
for( trav = head ; trav != NULL; trav = trav->next)
    cnt++;

trav = head;
for (i=1; i< cnt/2; i++)
    trav = trav->next;
cout << trav->data;
```

iters $= n + \frac{n}{2}$

$O(n)$

Linked List singly linear list → find mid




```
slow = head;
fast = head;
while (fast != null && fast->next != null) {
    slow = slow->next;
    fast = fast->next->next;
}
cout << slow->data;
```

even

odd

Linked List Singly linear list - find mid (recursion).

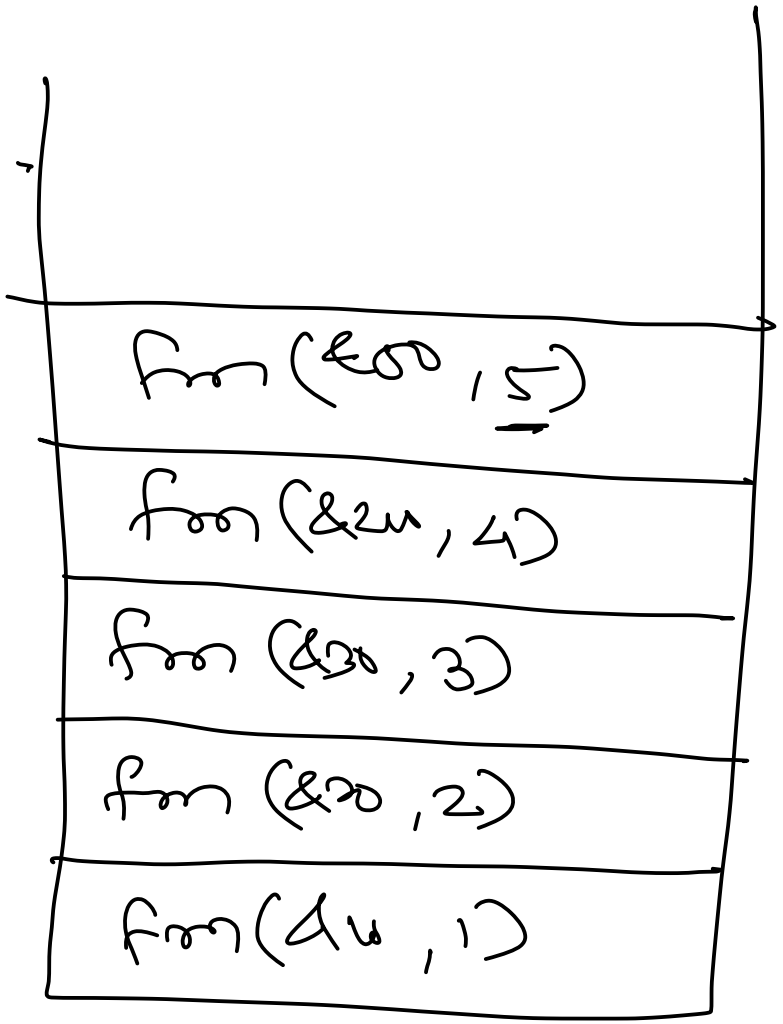
head

10 → 20 → 30 → 40 → 50 }

cnt
5

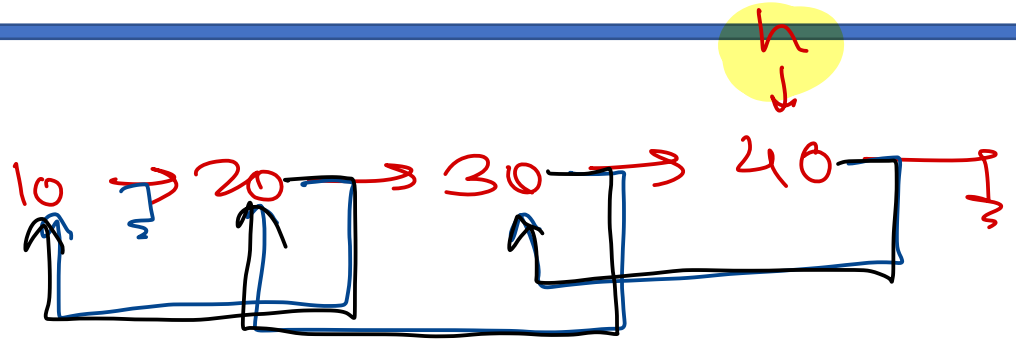
```
void find_mid(node *tmp, int pos) {
    if(tmp->next == NULL) {
        cnt = pos;
    }
    return;
    find_mid(tmp->next, pos+1);
    if(pos == cnt/2)
        cout << tmp->data;
}
```

call : find_mid(head, 1);



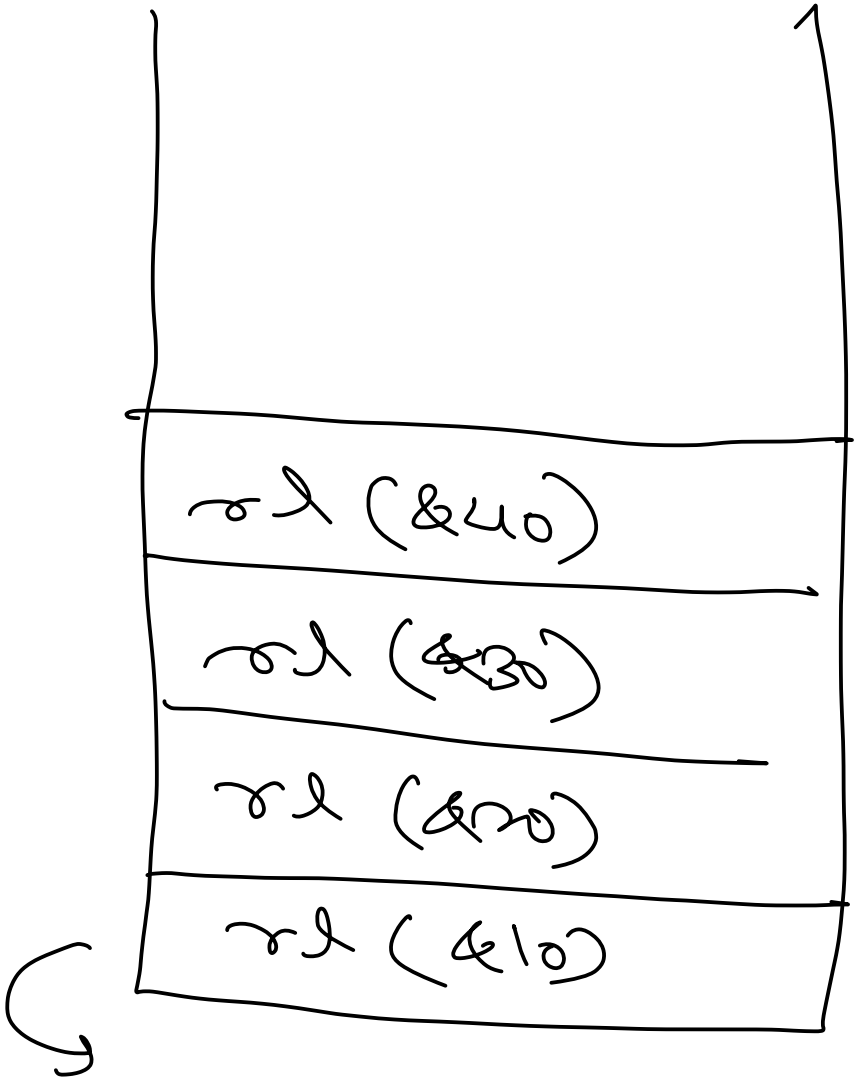
Linked List

singly linear LL - reverse. → recursion




```
node* revrec(node *trav) {
    if( trav->next == NULL) {
        head = trav;
        return trav;
    }
    tm = revrec( trav->next);
    tm->next = trav;

    return trav;
}
```



Common nodes in unsorted list

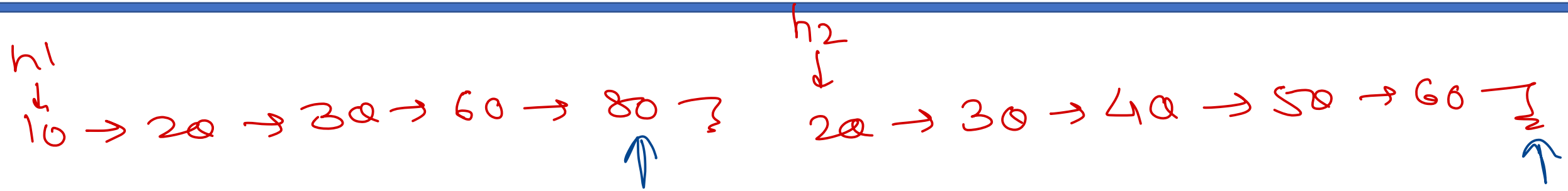
(n) h1 → 10 → 30 → 60 → 20 → 40 → }

O(n²)

(m) h2 → 30 → 50 → 80 → 20 → 10 → }

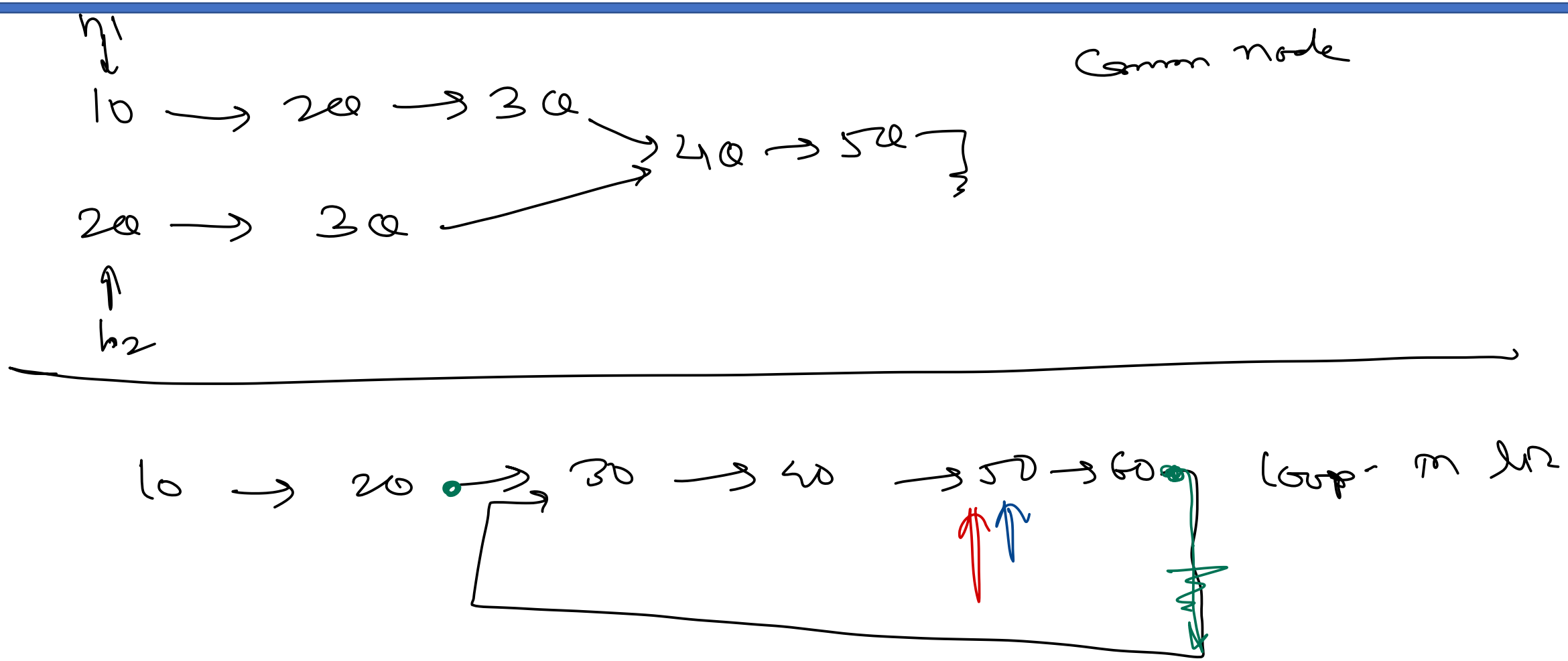
```
node *t1, *t2;
for(t1 = h1; t1 != NULL; t1 = t1->next) {
    for(t2 = h2; t2 != NULL; t2 = t2->next) {
        if(t1->data == t2->data)
            cout << t1->data;
    }
}
```

find common in sorted list

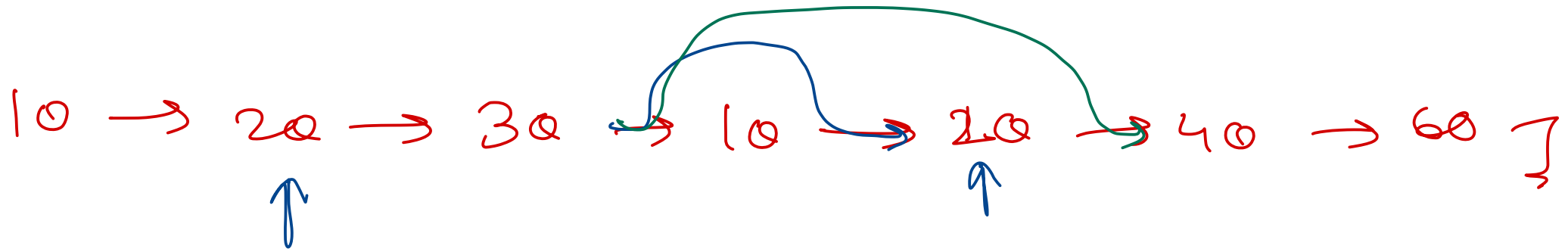



```
node *t1 = h1, *t2 = h2;
while(t1 != NULL && t2 != NULL) {
    if(t1->data == t2->data) {
        cout << t1->data;
        t1 = t1->next;
        t2 = t2->next;
    }
    else if (t1->data < t2->data)
        t1 = t1->next;
    else
        t2 = t2->next;
}
```

Linked List



Linked List




```
bar = head;
while (bar != null) {
    ???
    bar = bar->next;
}
```

STL - Standard Template Library

- STL is part of C++ standard.
- It has template implementations of common data structures.
- STL has three main components
  - Containers → data structs
  - Algorithms → global fns
  - Iterators → traverse through list.
- Additionally STL also have
  - Function objects
  - Allocators
  - Utility

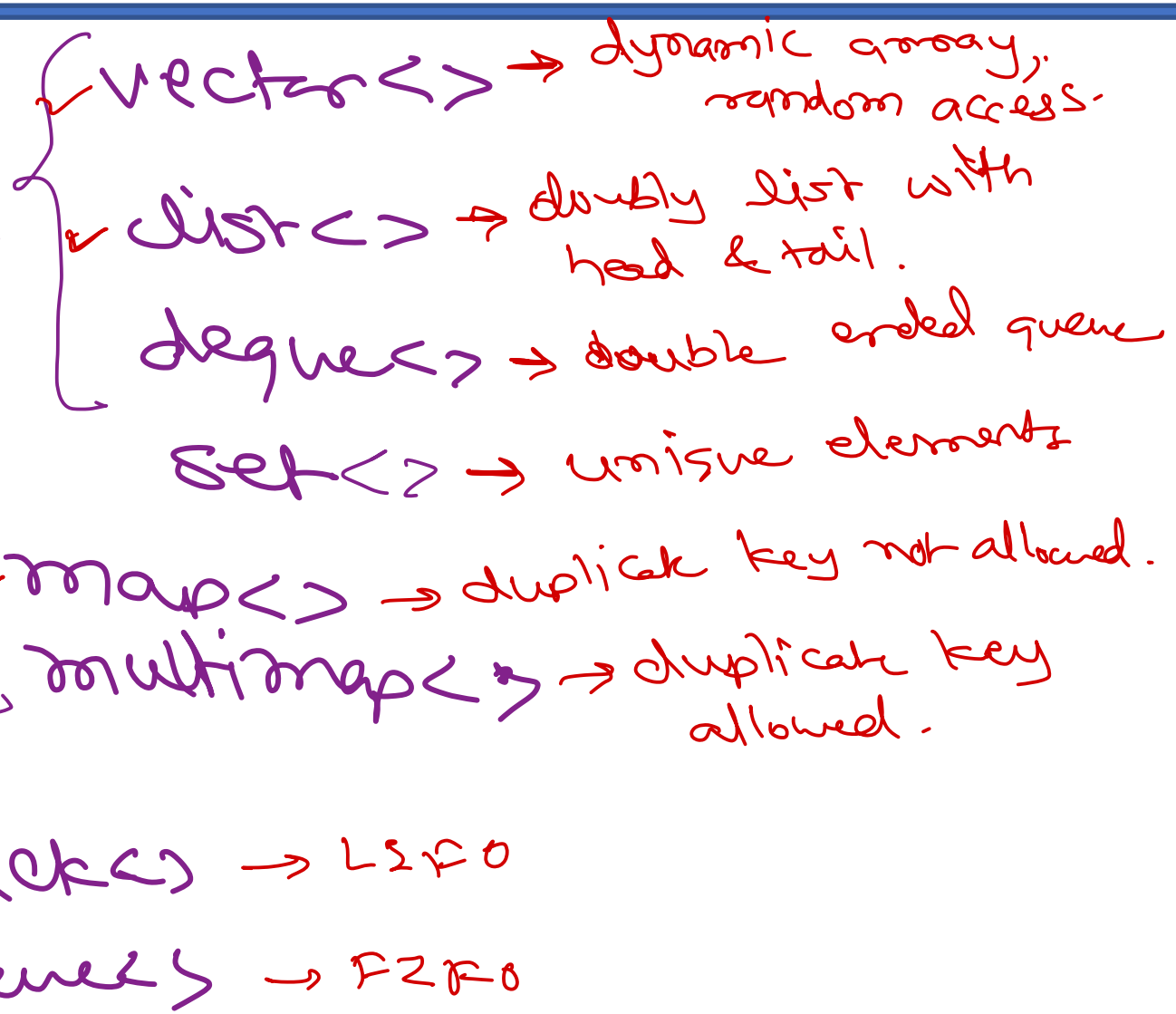
vector<>
list<>
set
deque<>

stack<>
queue<>

map<>
multimap<>

STL

- Containers hold data and operations to be performed on data.
- STL containers are of three types
  - Sequential: Linear collection
    - vector, list, deque
  - Associative: Key-value pair collection
    - set, map, multimap
  - Adapters: Limited container functionality
    - stack, queue



STL

- Containers are traversed using iterators.
- Usually iterators are implemented as nested classes in containers.
- Iterators are smart pointers (with -> and * operators overloaded).
- There are six types of iterators
  - Input iterator (read ops, fwd)
  - Output iterator (write ops, fwd)
  - Bi-directional iterator (rw, bi-dirn)
  - Forward iterator (rw, fwd)
  - Reverse iterator (rw, rev)
  - Random access iterator (rw, any)

STL

- Algorithms are global functions that operates on containers.
- They can be classified as
  - Search functions
  - Sort functions
  - Manipulaion functions
  - Non-modifying functions
  - Numeric functions

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>