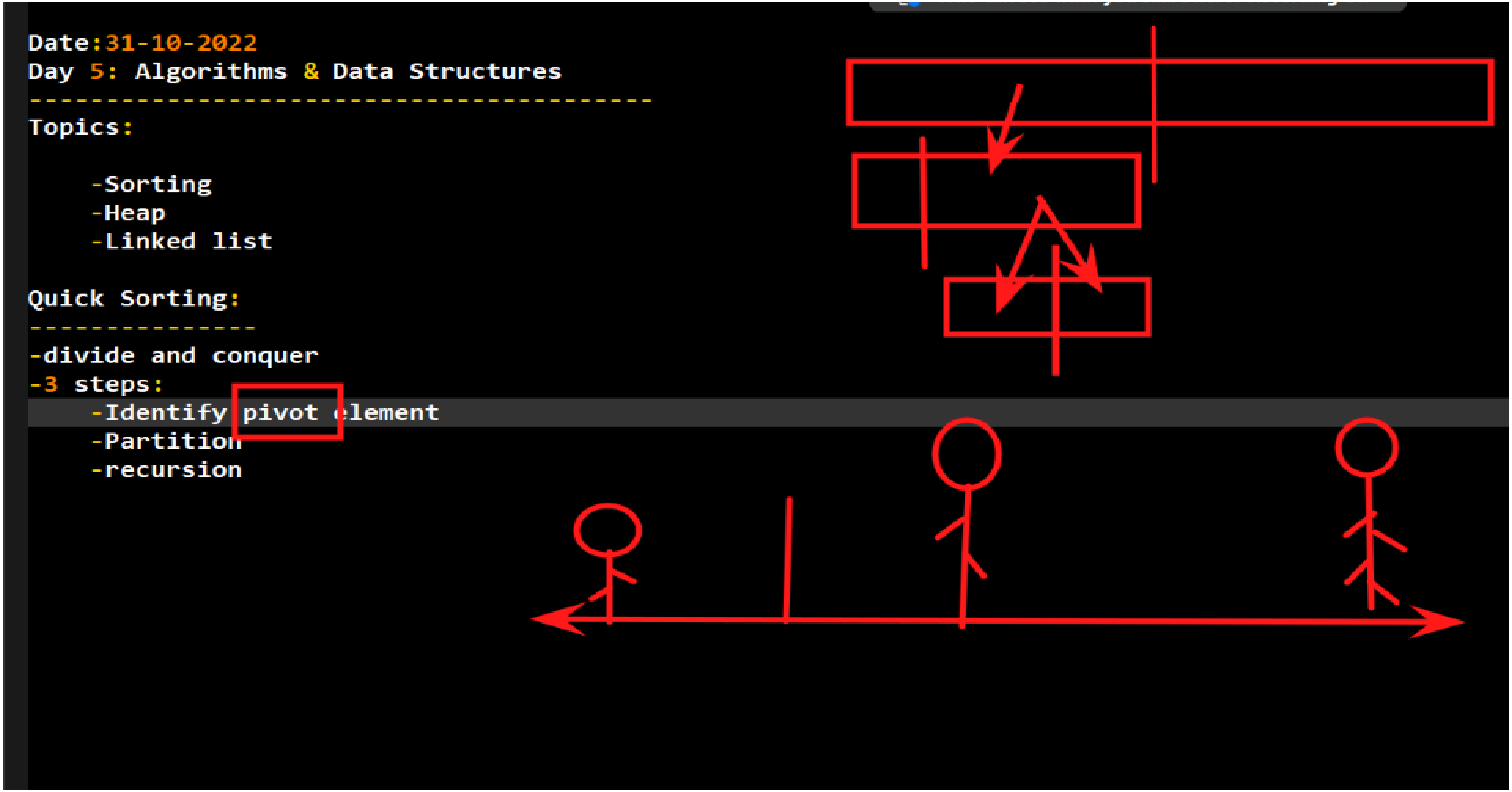
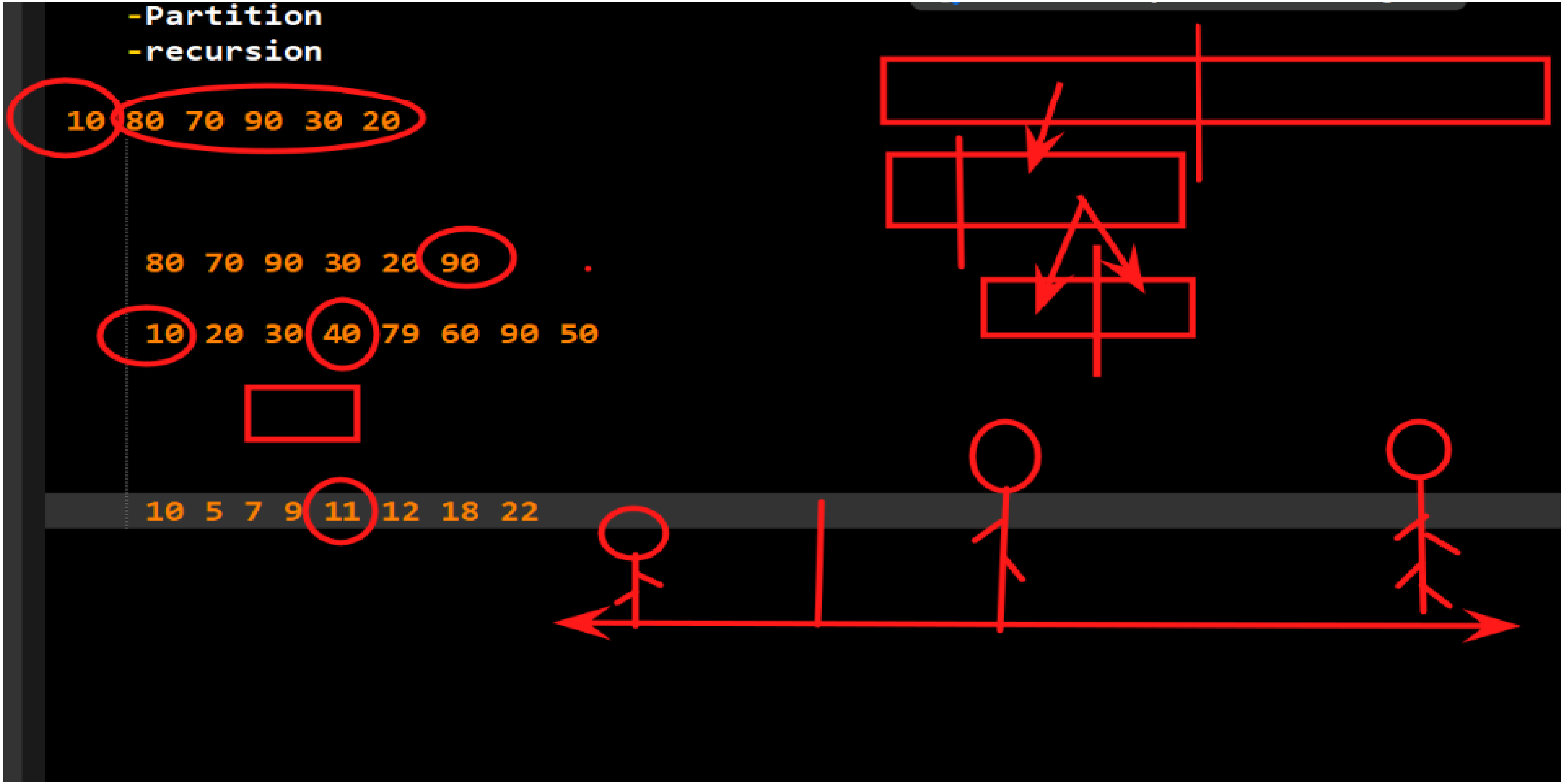
Sep22 : Day 3

Kiran Waghmare

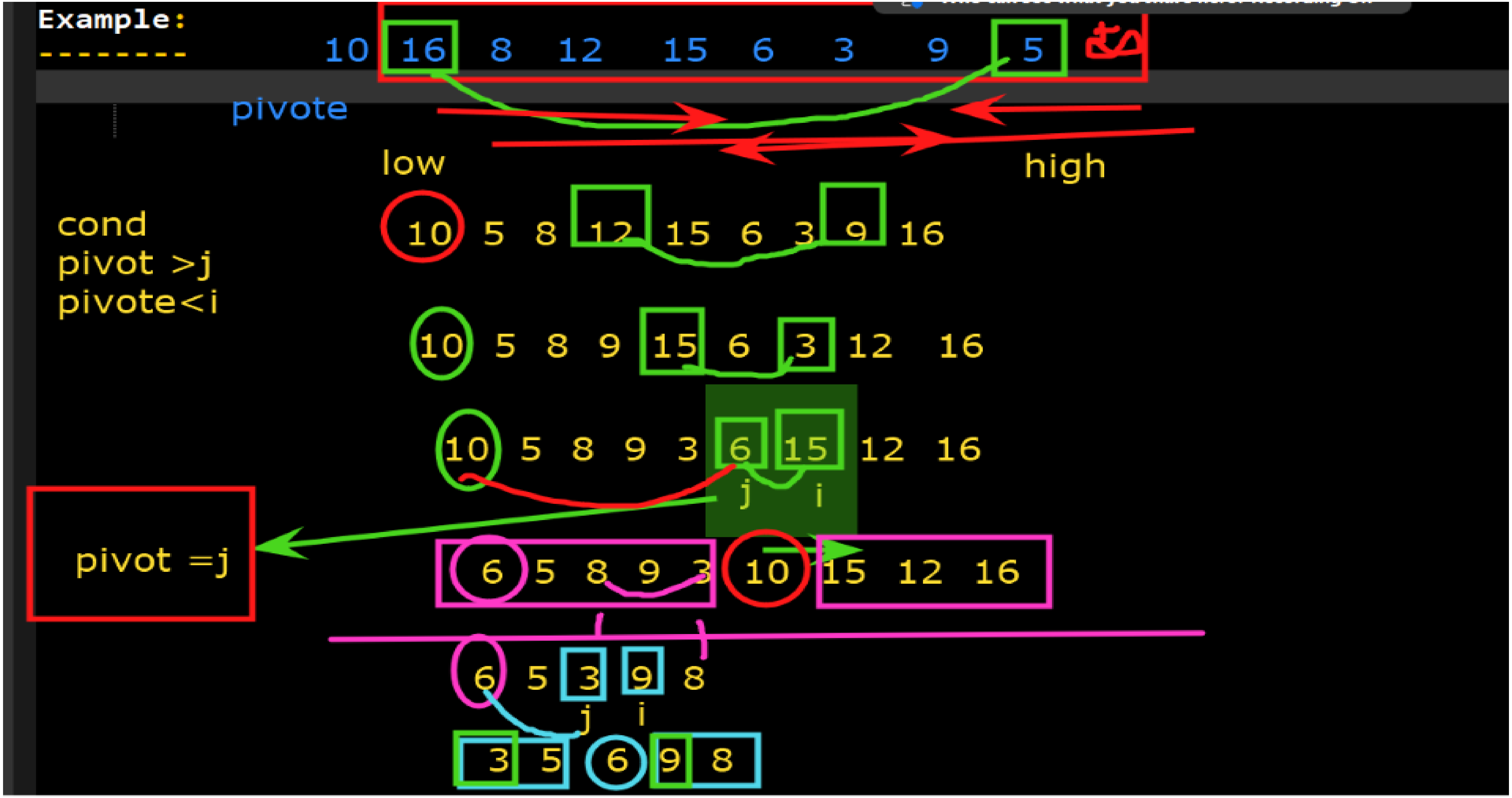
CDAC Mumbai



```
Date:31-10-2022
Day 5: Algorithms & Data Structures
-----------------------------------------
Topics:

    -Sorting
    -Heap
    -Linked list

Quick Sorting:
---------------
-divide and conquer
-3 steps:
    -Identify pivot element
    -Partition
    -recursion
```



```
   -Partition
   -recursion

 10 80 70 90 30 20


   80 70 90 30 20 90

   10 20 30 40 79 60 90 50


   10 5 7 9 11 12 18 22
```



Example:

----

10 16 8 12 15 6 3 9 5

pivote

low

high

cond
pivot >j
pivote<i

10 5 8 12 15 6 3 9 16

10 5 8 9 15 6 3 12 16

10 5 8 9 3 6 15 12 16

j i

pivot =j

6 5 8 9 3 10 15 12 16

6 5 3 9 8

j i

3 5 6 9 8

The following procedure implements quicksort:

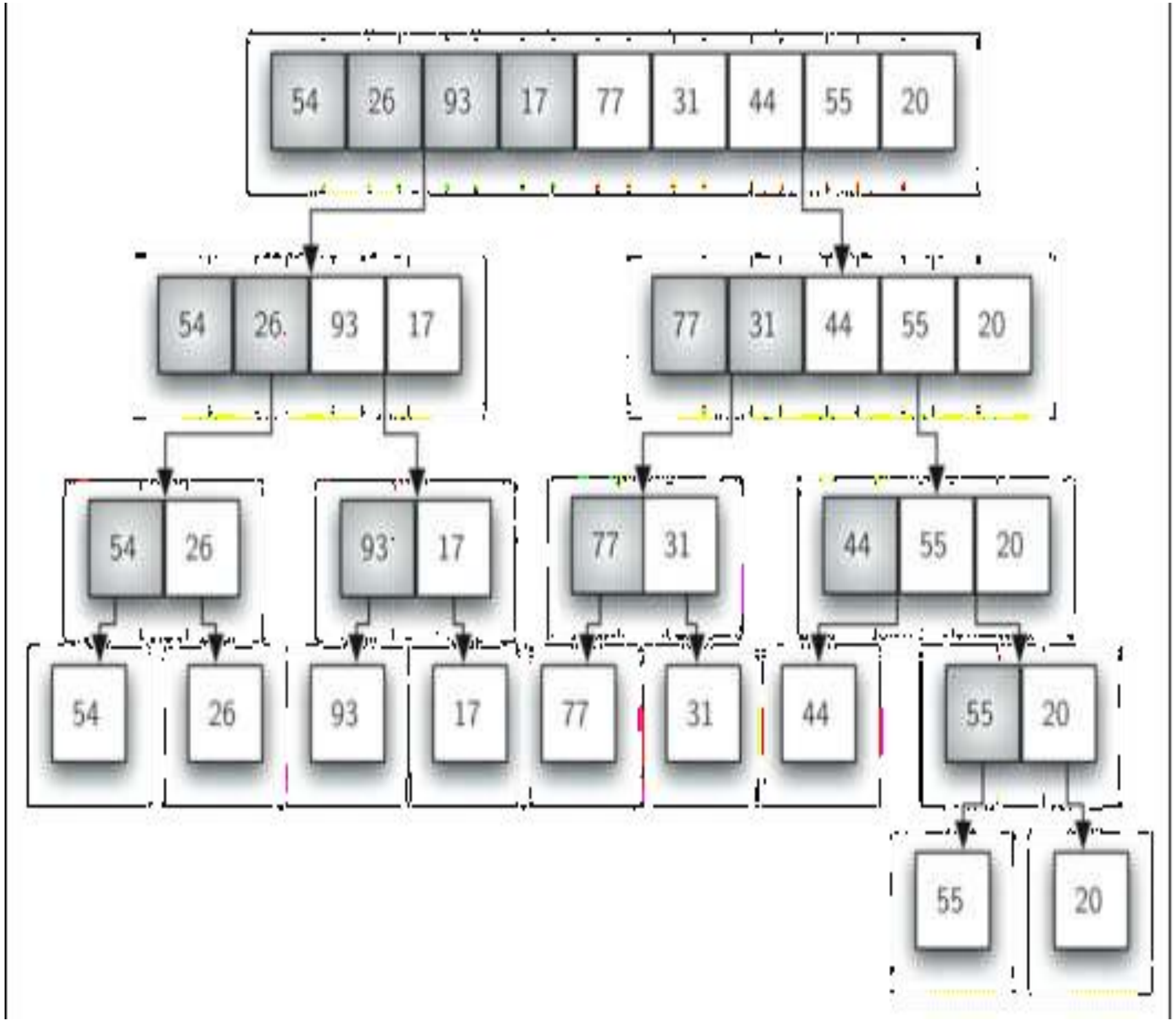
```
QUICKSORT($A, p, r$)
if $p < r$
    $q$ = PARTITION($A, p, r$)
    QUICKSORT($A, p, q - 1$)
    QUICKSORT($A, q + 1, r$)
```

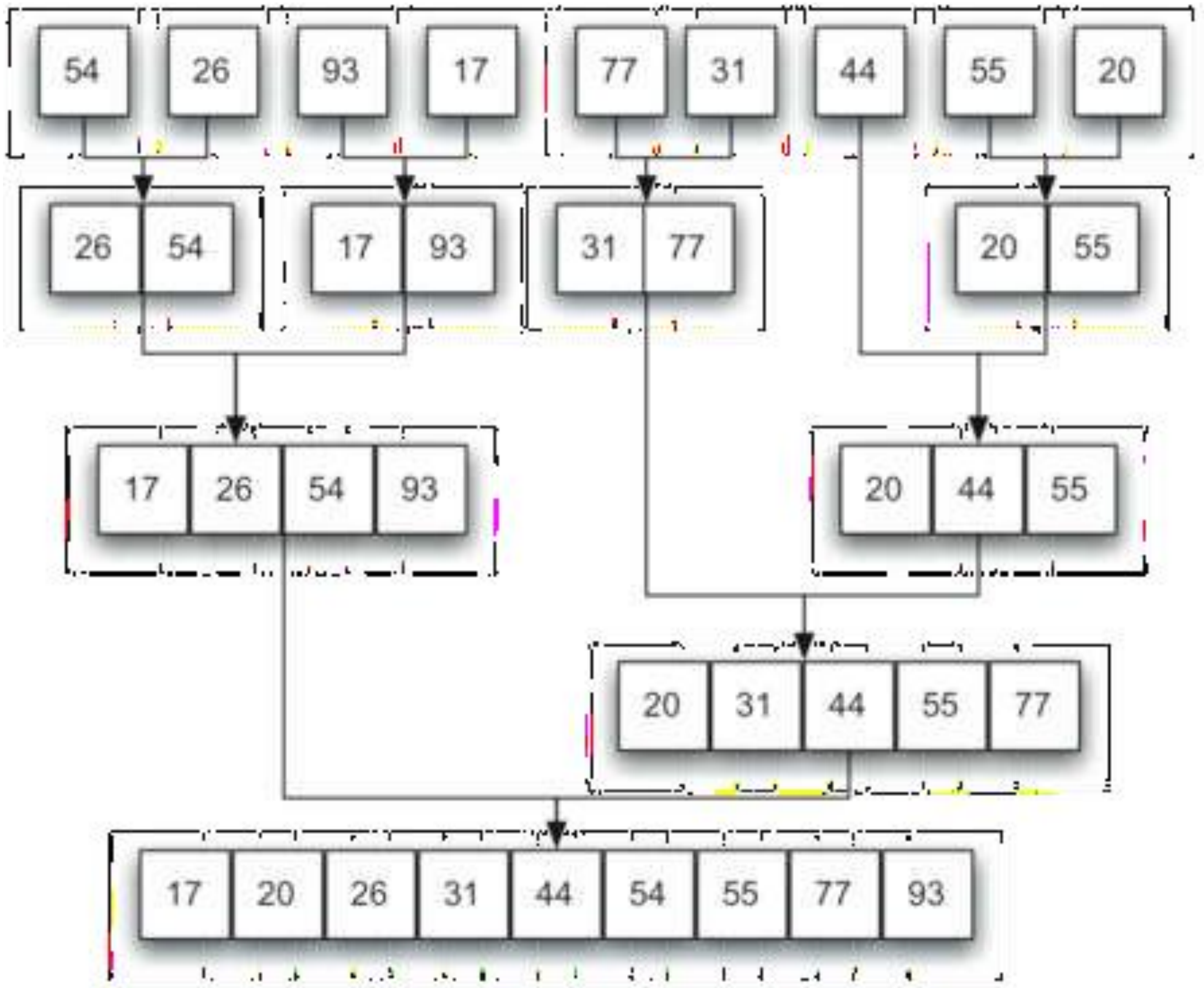
To sort an entire array A, the initial call is QUICKSORT(A, 1, $A.length$).

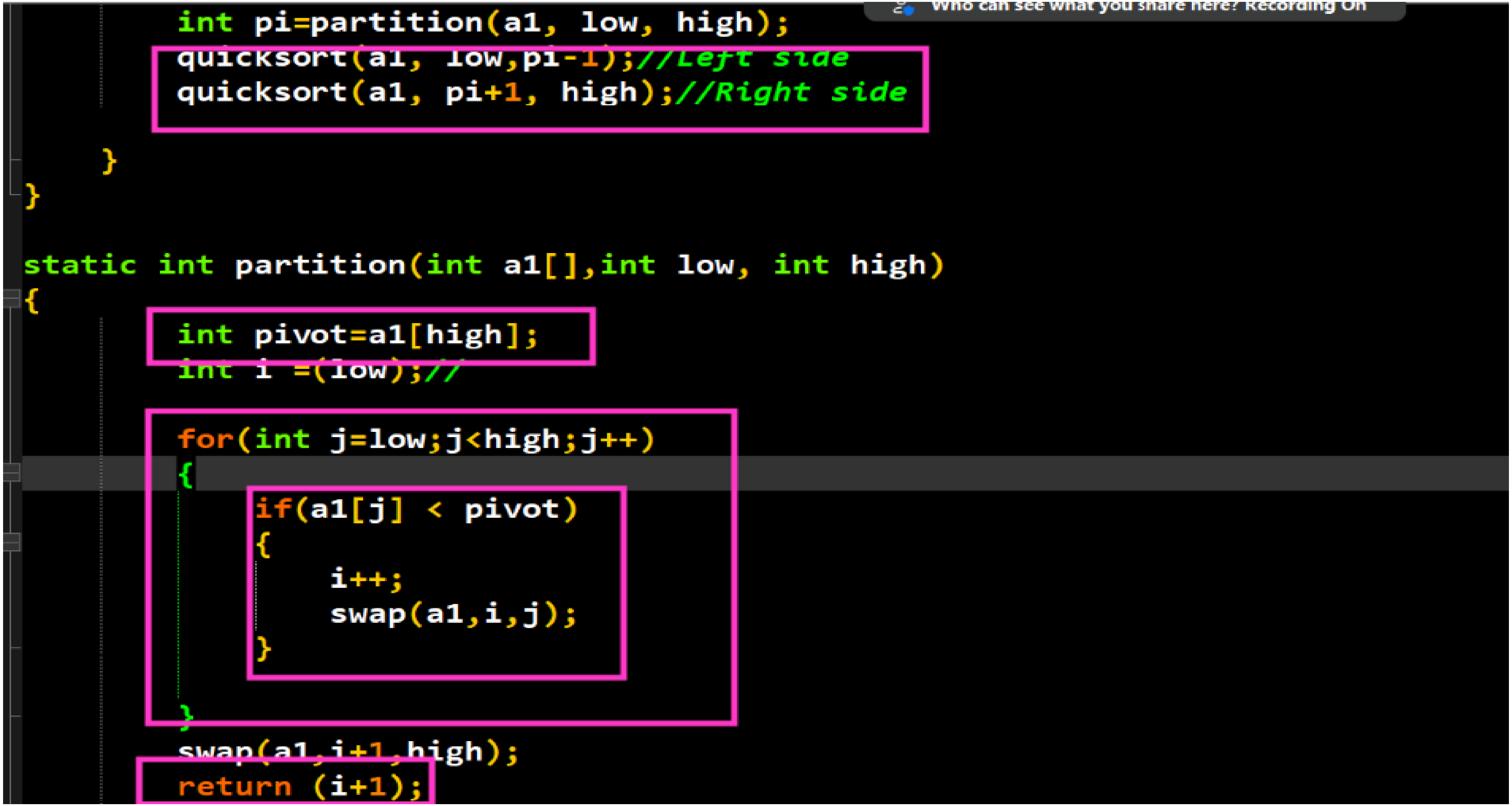
Partitioning the array

The key to the algorithm is the PARTITION procedure, which rearranges the subarray $A[p\,..\,r]$ in place.

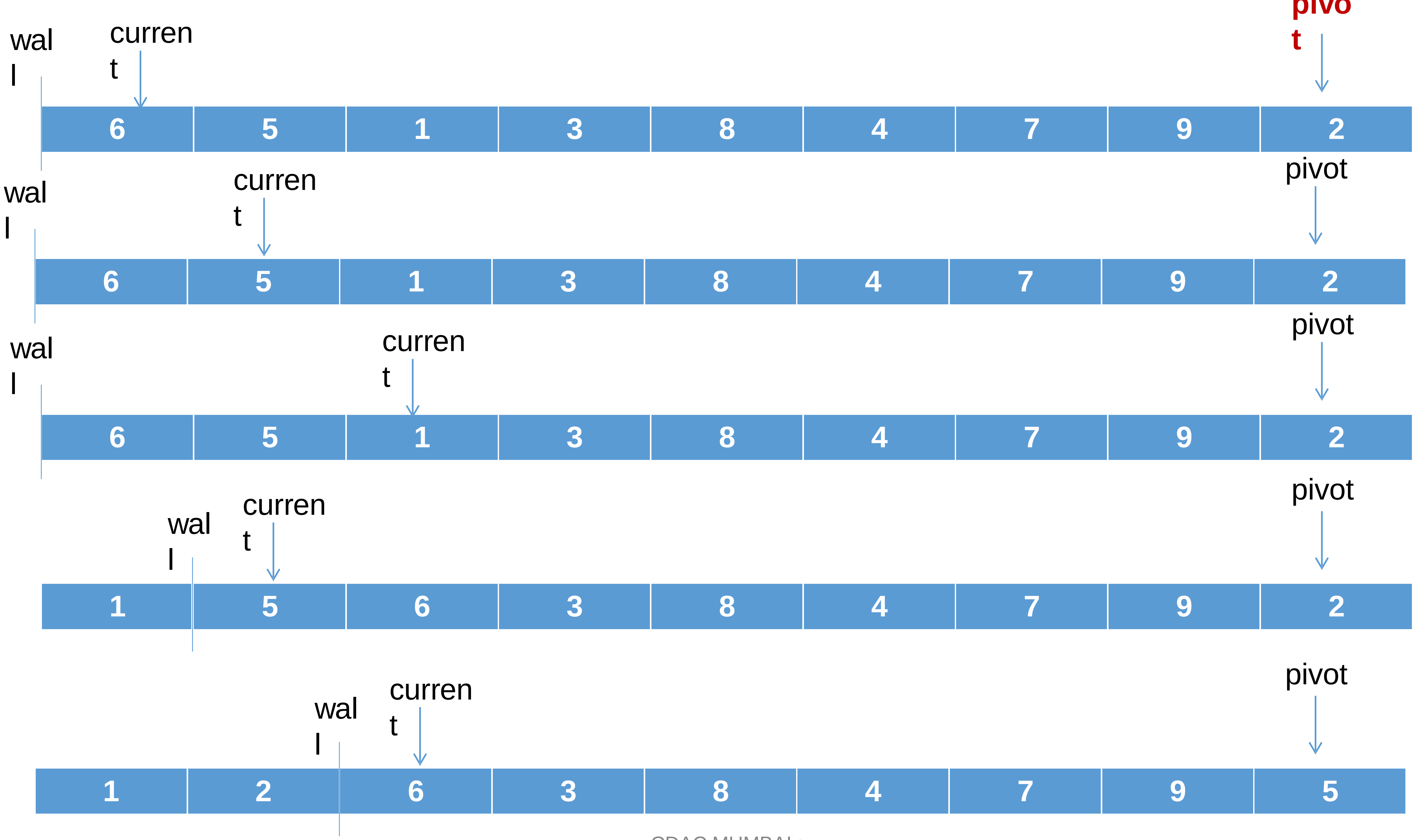
```
PARTITION($A, p, r$)
$x = A[r]$
$i = p - 1$
for $j = p$ to $r - 1$
    if $A[j] \le x$
        $i = i + 1$
        exchange $A[i]$ with $A[j]$
exchange $A[i + 1]$ with $A[r]$
return $i + 1$
```


54 26 93 17 77 31 44 55 20
54 26 93 17
77 31 44 55 20
54 26
93 17
77 31
44 55 20
54
26
93
17
77
31
44
55 20
55
20


54
26
93
17
77
31
44
55
20
26 54
17 93
31 77
20 55
17 26 54 93
20 44 55
20 31 44 55 77
17 20 26 31 44 54 55 77 93



```
        int pi=partition(a1, low, high);
        quicksort(a1, low,pi-1);//Left side
        quicksort(a1, pi+1, high);//Right side

    }
}

static int partition(int a1[],int low, int high)
{
        int pivot=a1[high];
        int i =(low);//

        for(int j=low;j<high;j++)
        {
            if(a1[j] < pivot)
            {
                i++;
                swap(a1,i,j);
            }

        }
        swap(a1,i+1,high);
        return (i+1);
```



wal
l
curren
t
pivo
t

6	5	1	3	8	4	7	9	2

wal
l
curren
t
pivot

6	5	1	3	8	4	7	9	2

wal
l
curren
t
pivot

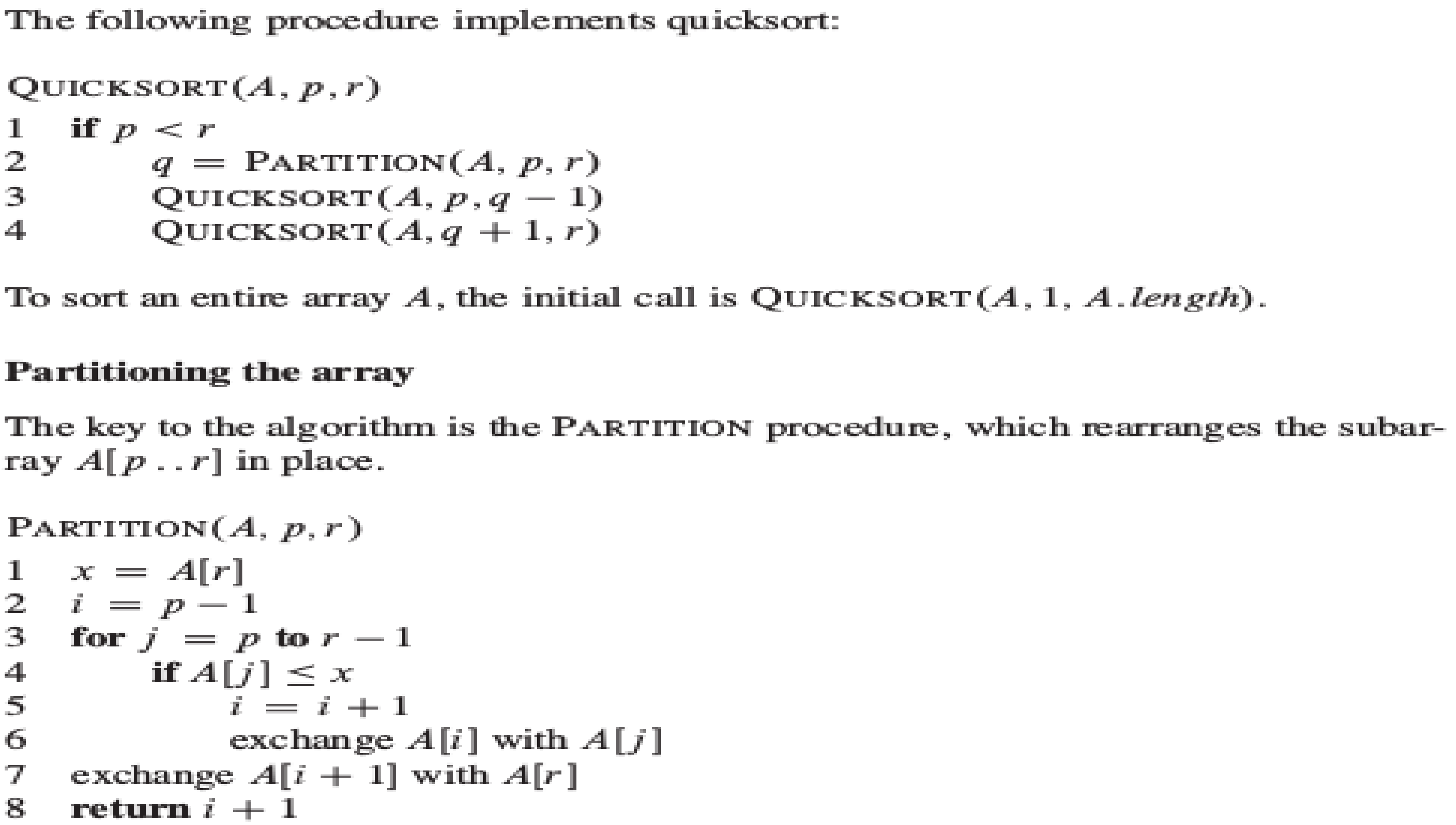
6	5	1	3	8	4	7	9	2

wal
l
curren
t
pivot

1	5	6	3	8	4	7	9	2

wal
l
curren
t
pivot

1	2	6	3	8	4	7	9	5



The following procedure implements quicksort:

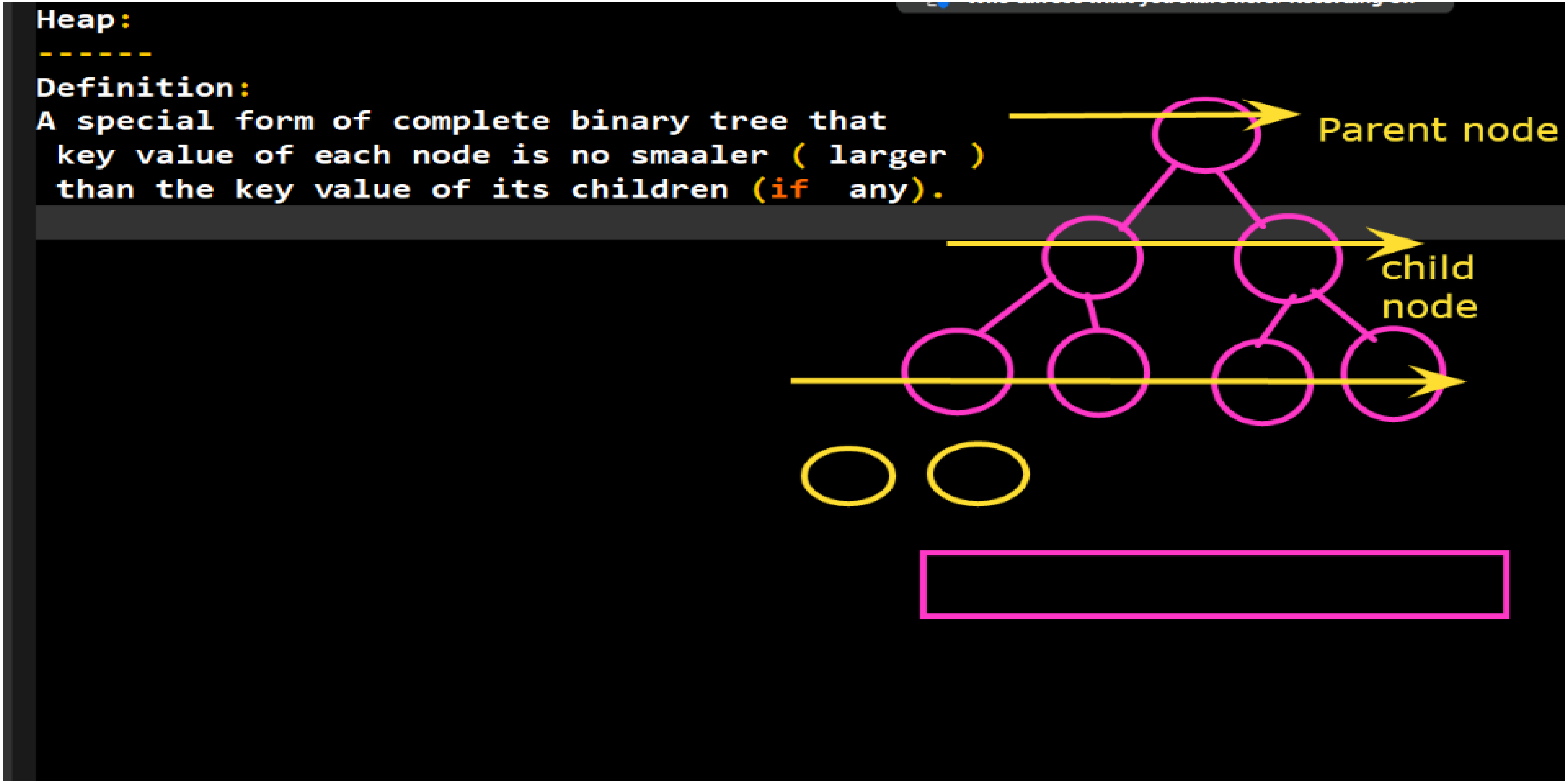
```
QUICKSORT(A, p, r)
if $p < r$
    $q$ = PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

To sort an entire array A, the initial call is QUICKSORT(A, 1, $A.length$).

Partitioning the array

The key to the algorithm is the PARTITION procedure, which rearranges the subarray $A[p \,..\, r]$ in place.

```
PARTITION(A, p, r)
$x = A[r]$
$i = p - 1$
for $j = p$ to $r - 1$
    if $A[j] \le x$
        $i = i + 1$
        exchange $A[i]$ with $A[j]$
exchange $A[i + 1]$ with $A[r]$
return $i + 1$
```



```
Heap:
-------
Definition:
A special form of complete binary tree that
 key value of each node is no smaaler ( larger )
 than the key value of its children (if  any).
```

Heap

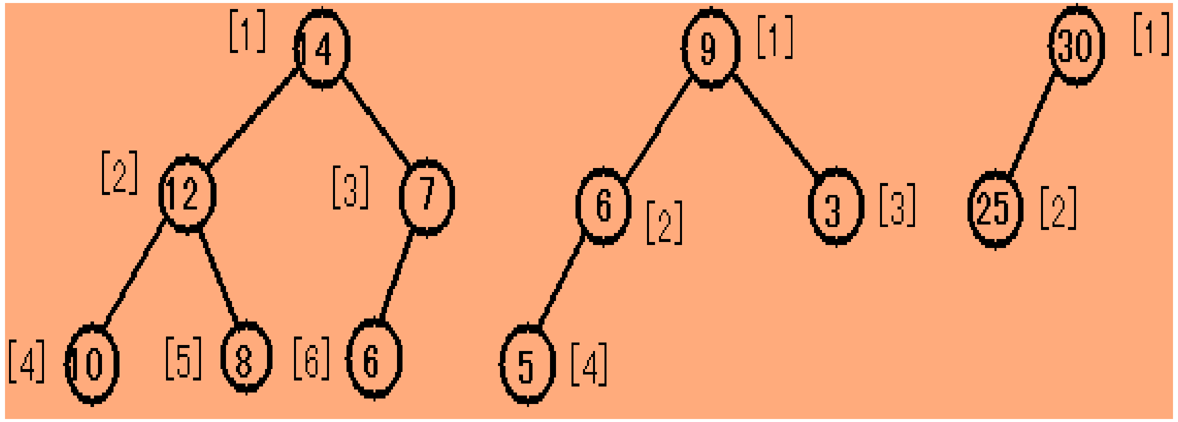
- **Definition in Data Structure**
  - **Heap:** A special form of complete binary tree that key value of each node is no smaller (larger) than the key value of its children (if any).

- **Max-Heap: root node has the largest key.**
  - A *max tree* is a tree in which the key value in each node is no smaller than the key values in its children.
  - A *max heap* is a complete binary tree that is also a max tree.

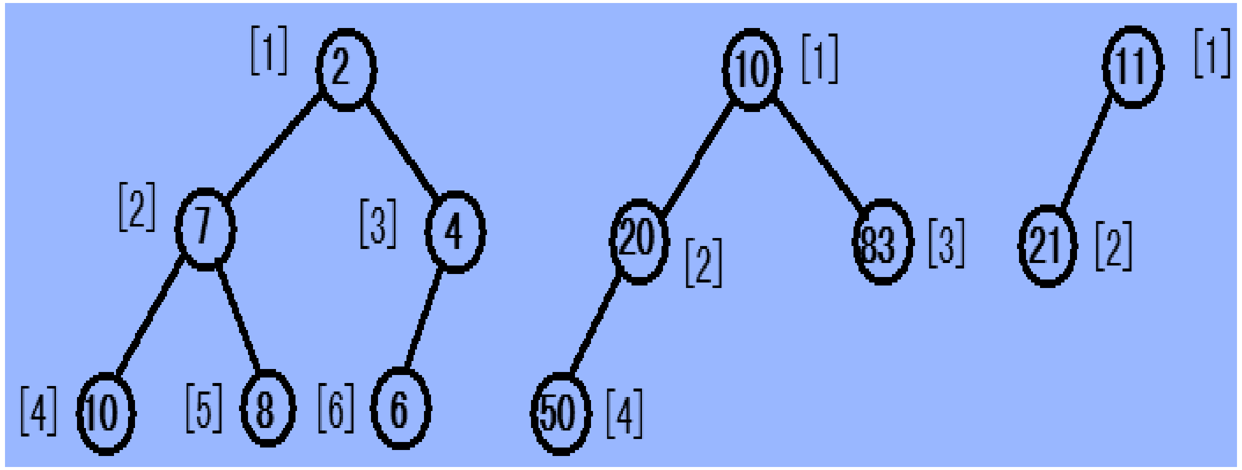
- **Min-Heap: root node has the smallest key.**
  - A *min tree* is a tree in which the key value in each node is no larger than the key values in its children.
  - A *min heap* is a complete binary tree that is also a min tree.

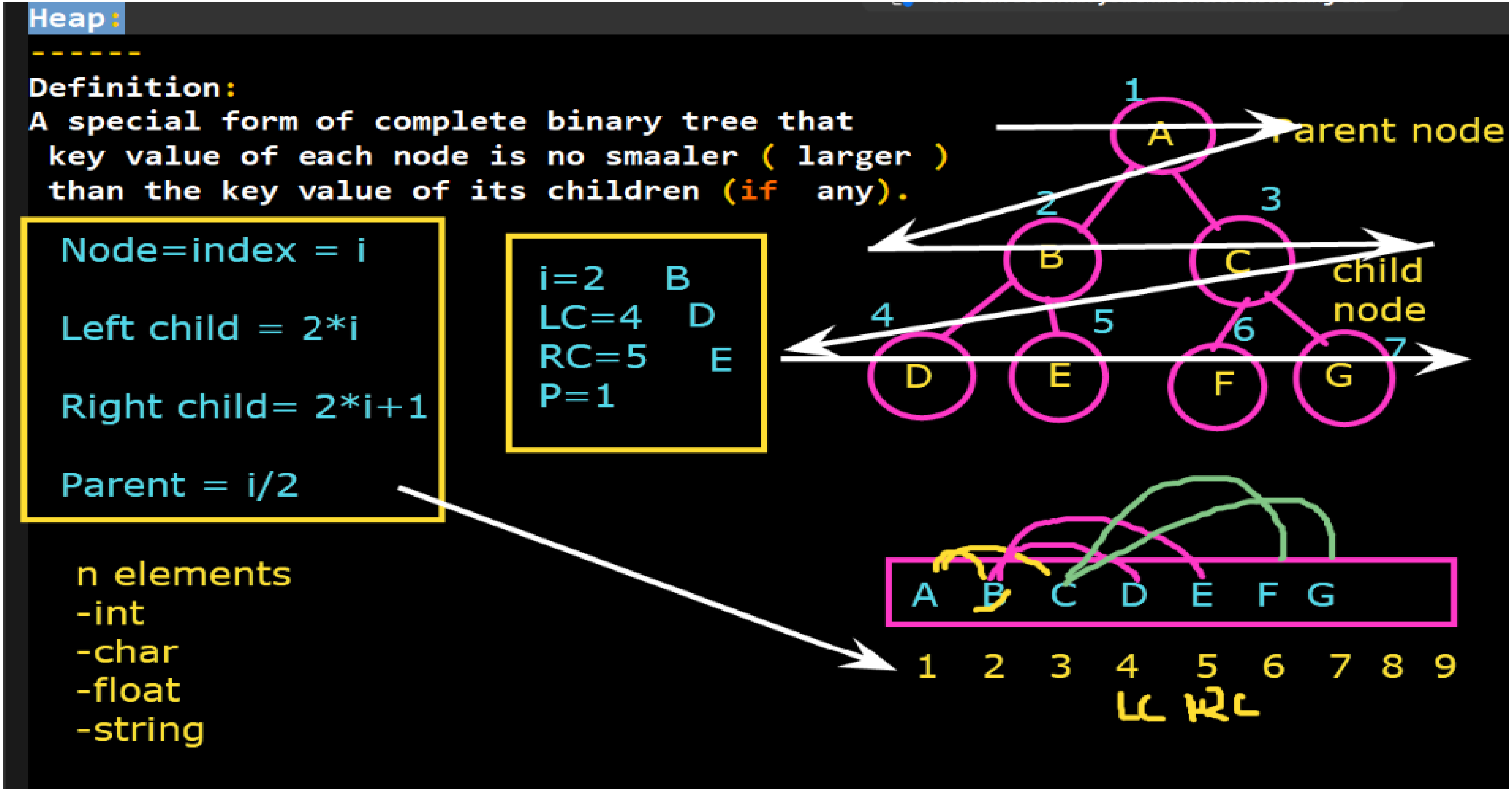
Heap

- **Example:**
  - Max-Heap



  - Min-Heap





Heap:

Definition:

A special form of complete binary tree that
key value of each node is no smaaler (larger)
than the key value of its children (if any).

Node=index = i

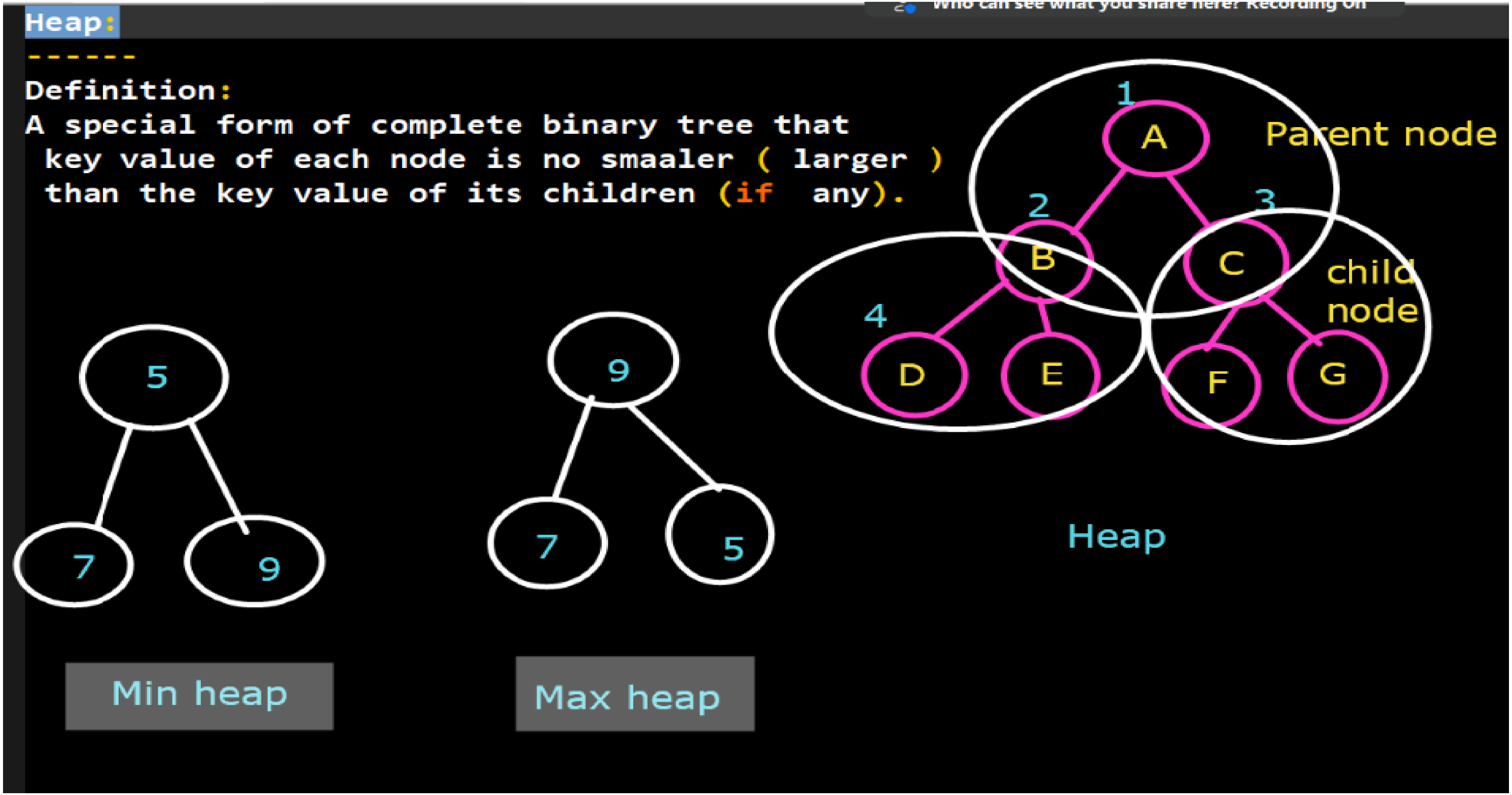
Left child = 2*i

Right child= 2*i+1

Parent = i/2

i=2 B
LC=4 D
RC=5 E
P=1

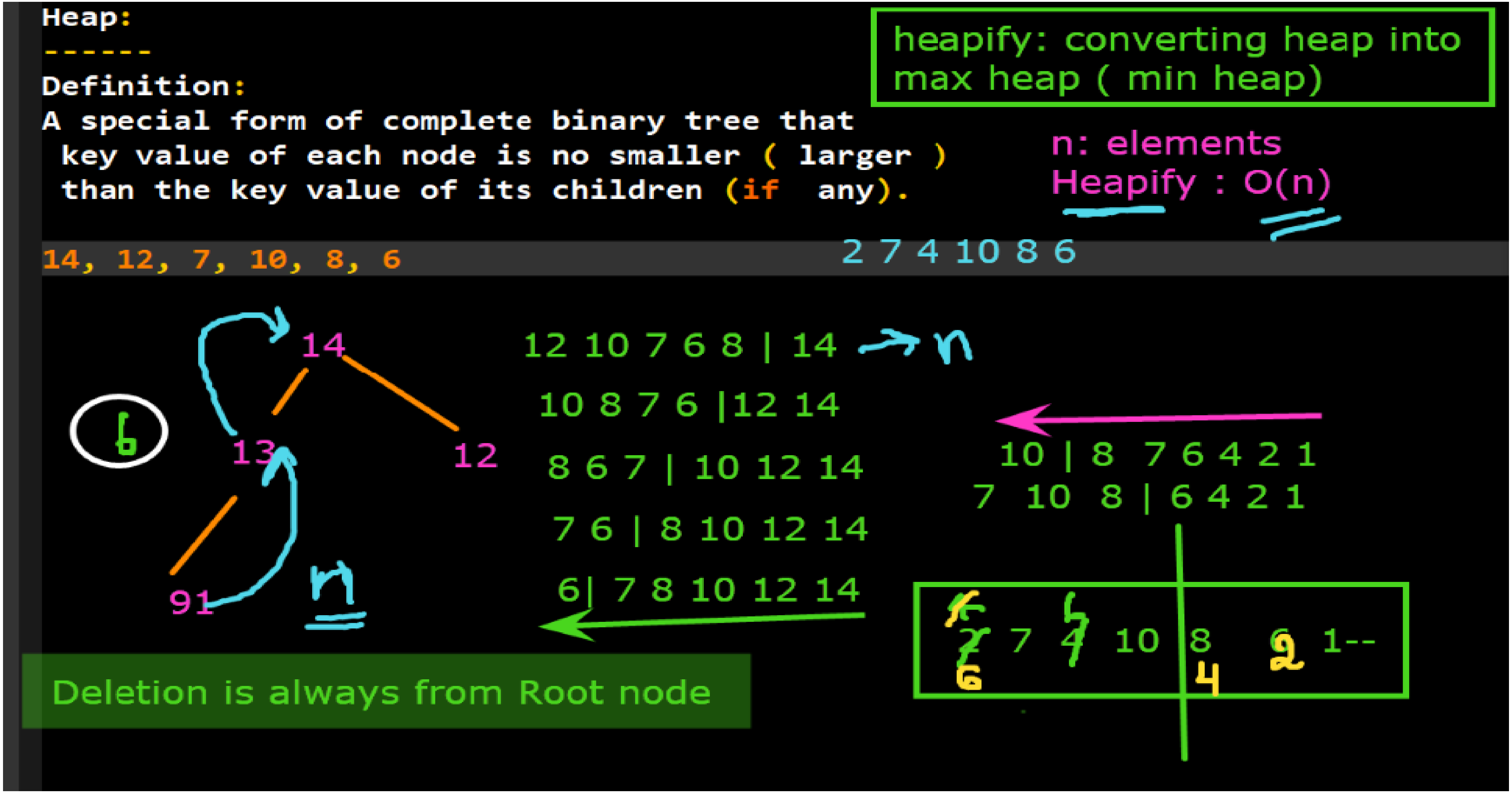
n elements
-int
-char
-float
-string



Heap:

Definition:

A special form of complete binary tree that
 key value of each node is no smaaler (larger)
 than the key value of its children (if any).



Heap:

Definition:

A special form of complete binary tree that key value of each node is no smaller (larger) than the key value of its children (if any).

heapify: converting heap into max heap (min heap)

n: elements

Heapify : O(n)

14, 12, 7, 10, 8, 6

2 7 4 10 8 6

12 10 7 6 8 | 14 → n

10 8 7 6 |12 14

8 6 7 | 10 12 14

7 6 | 8 10 12 14

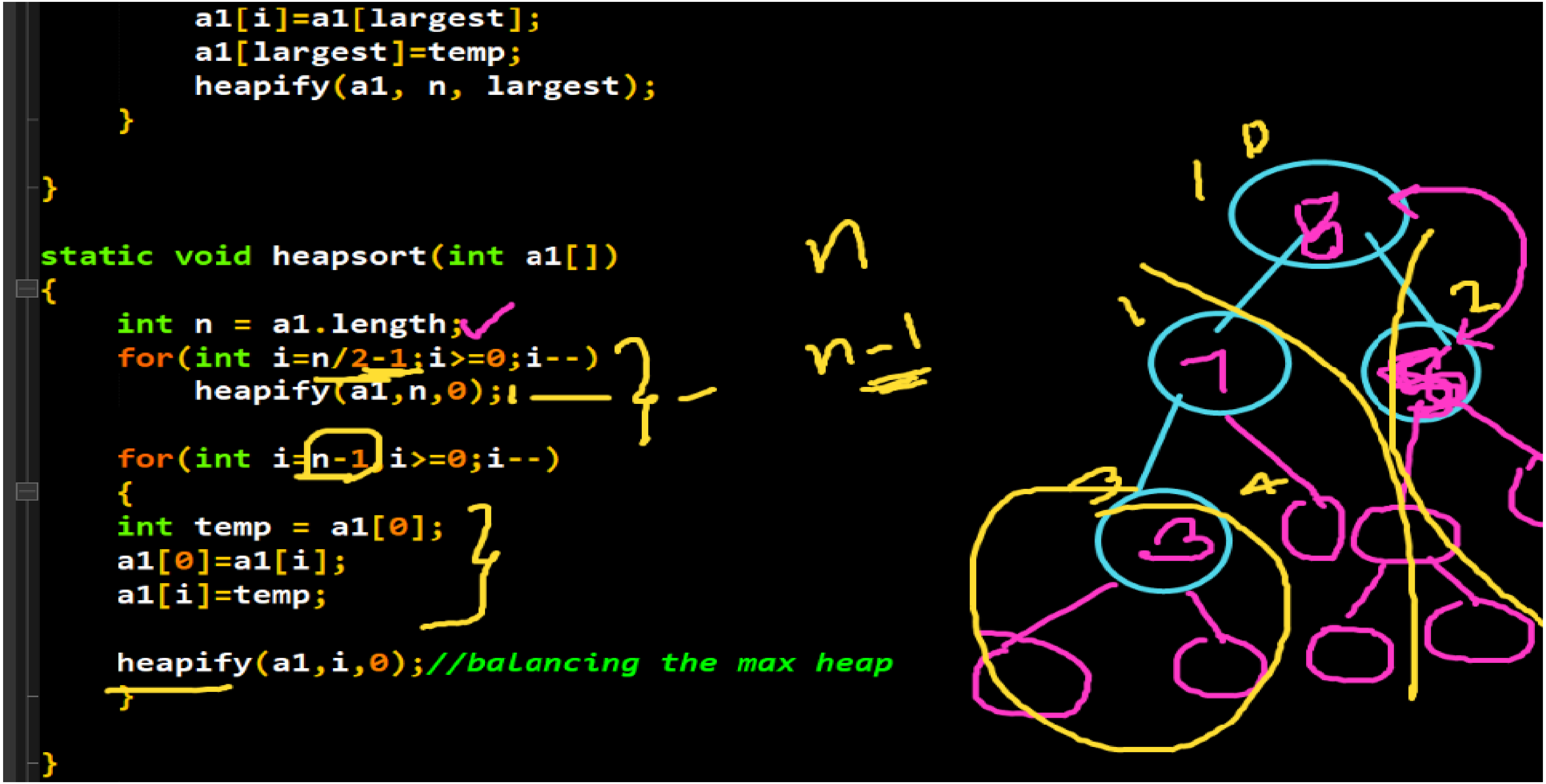
6| 7 8 10 12 14

10 | 8 7 6 4 2 1

7 10 8 | 6 4 2 1

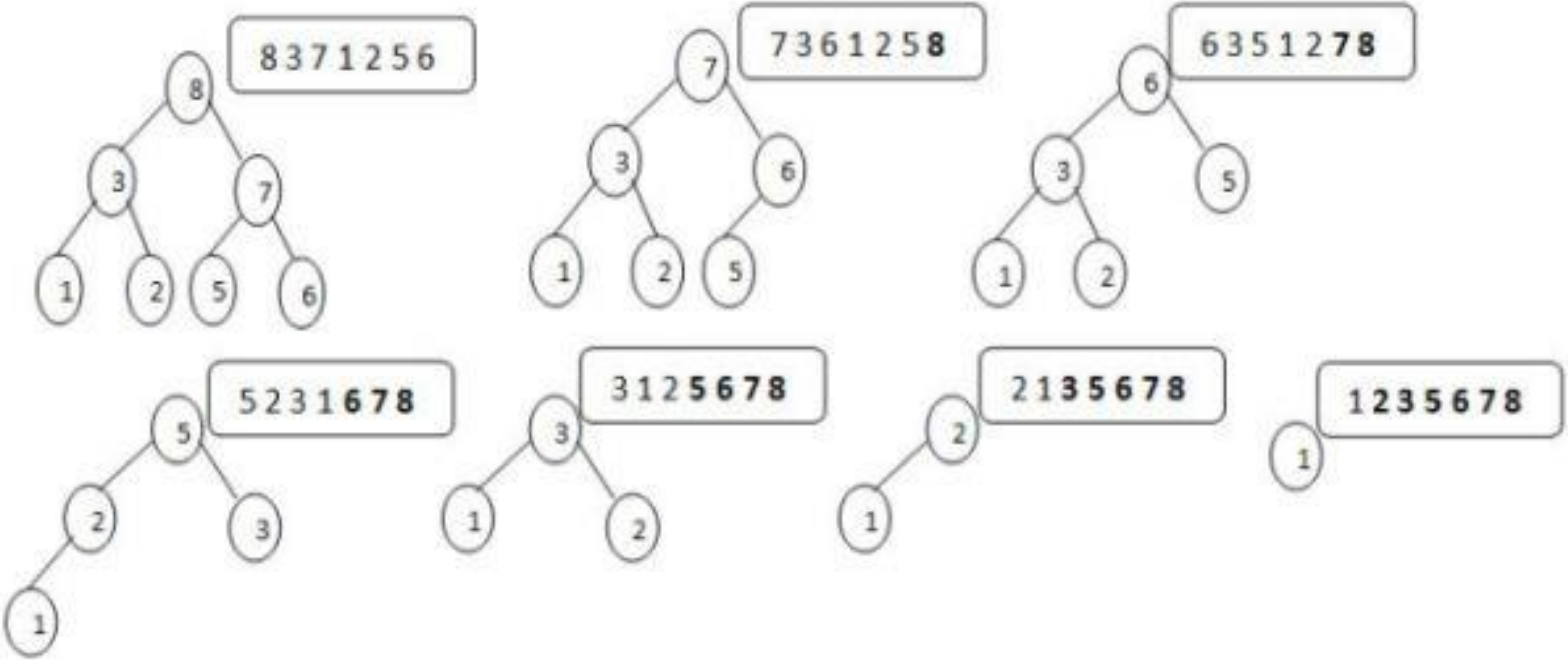
~~1~~ ~~2~~ 6 7 ~~4~~ 5 10 | 8 4 ~~6~~ 2 1--

Deletion is always from Root node

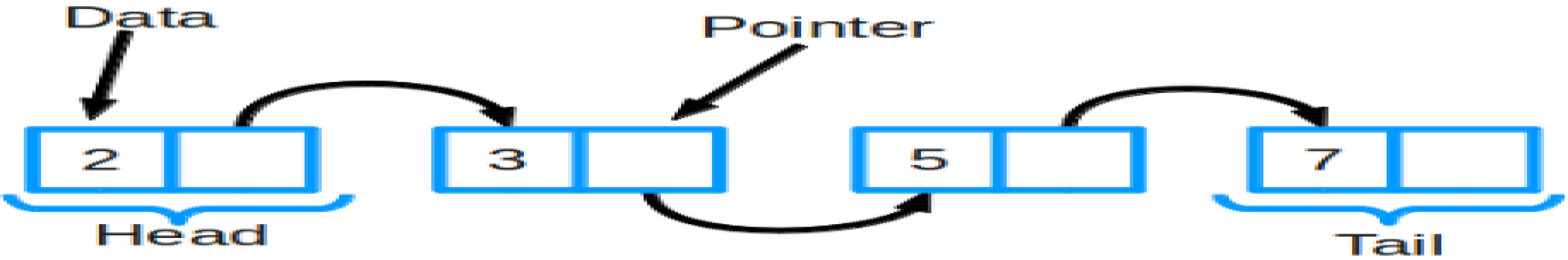


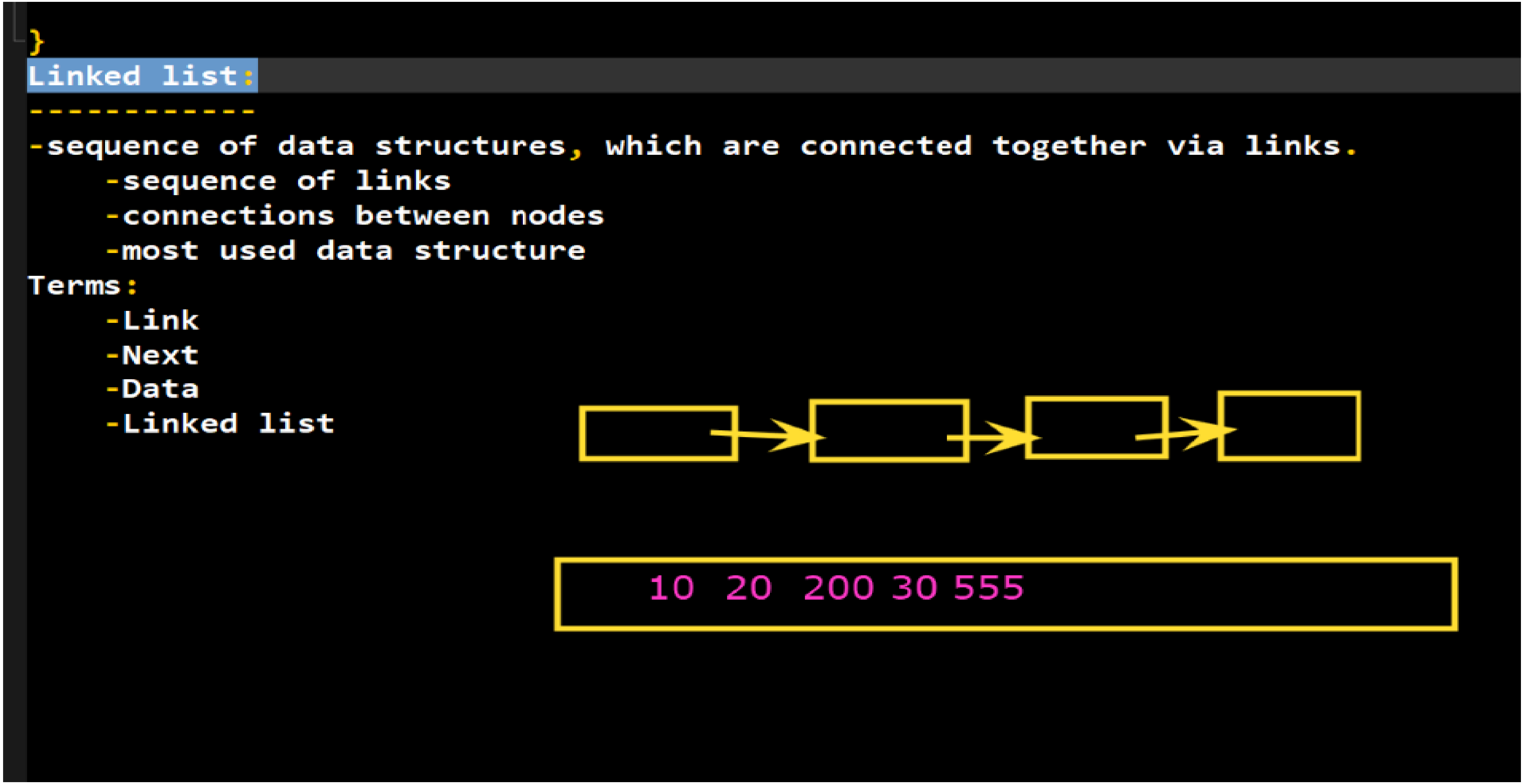
```
            a1[i]=a1[largest];
            a1[largest]=temp;
            heapify(a1, n, largest);
        }

}

static void heapsort(int a1[])
{
        int n = a1.length;
        for(int i=n/2-1;i>=0;i--)
            heapify(a1,n,0);

        for(int i=n-1;i>=0;i--)
        {
        int temp = a1[0];
        a1[0]=a1[i];
        a1[i]=temp;

        heapify(a1,i,0);//balancing the max heap
        }

}
```

Example:- The fig. shows steps of heap-sort for list (2 3 7 1 8 5 6)

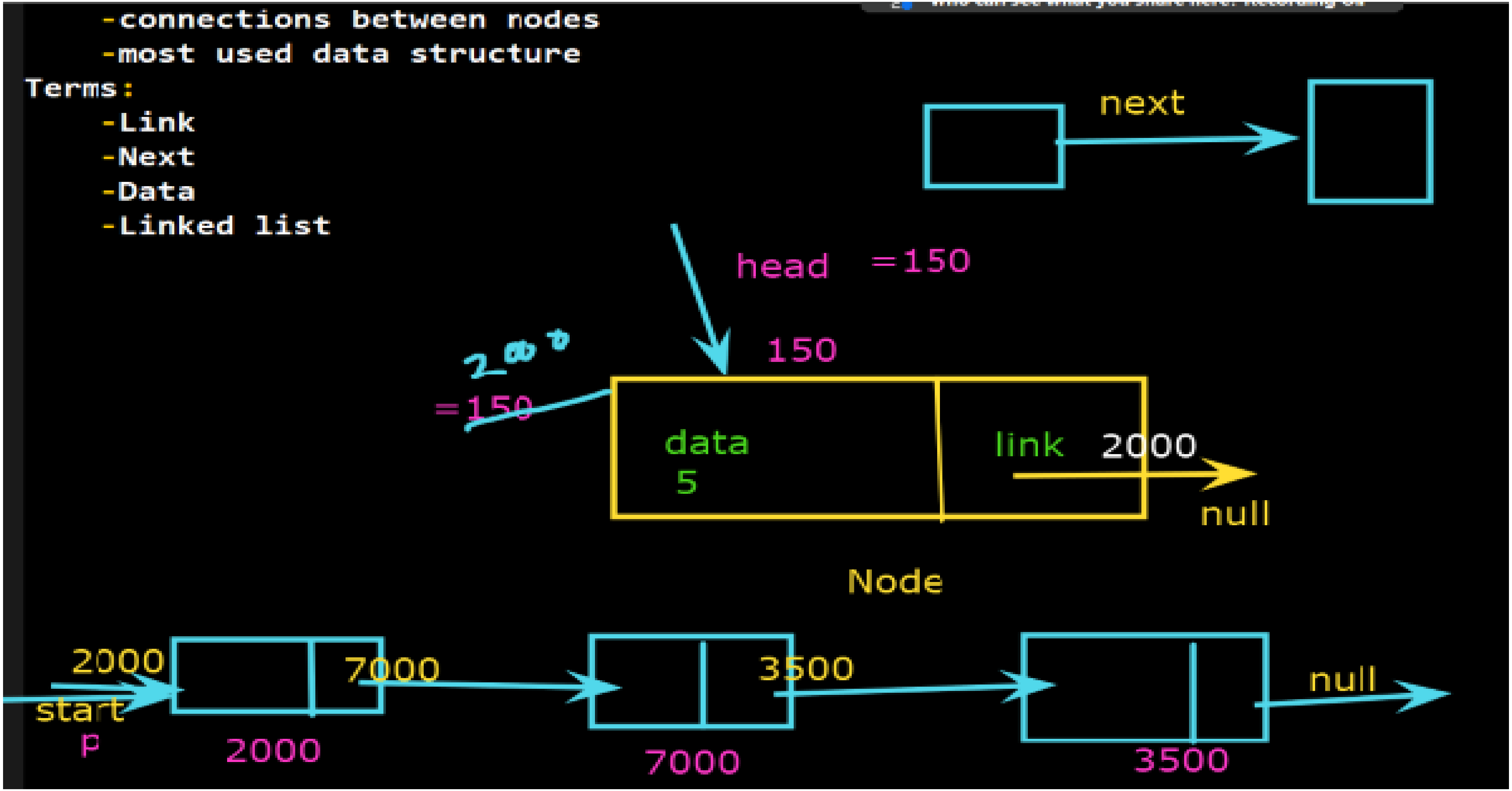


Linked list

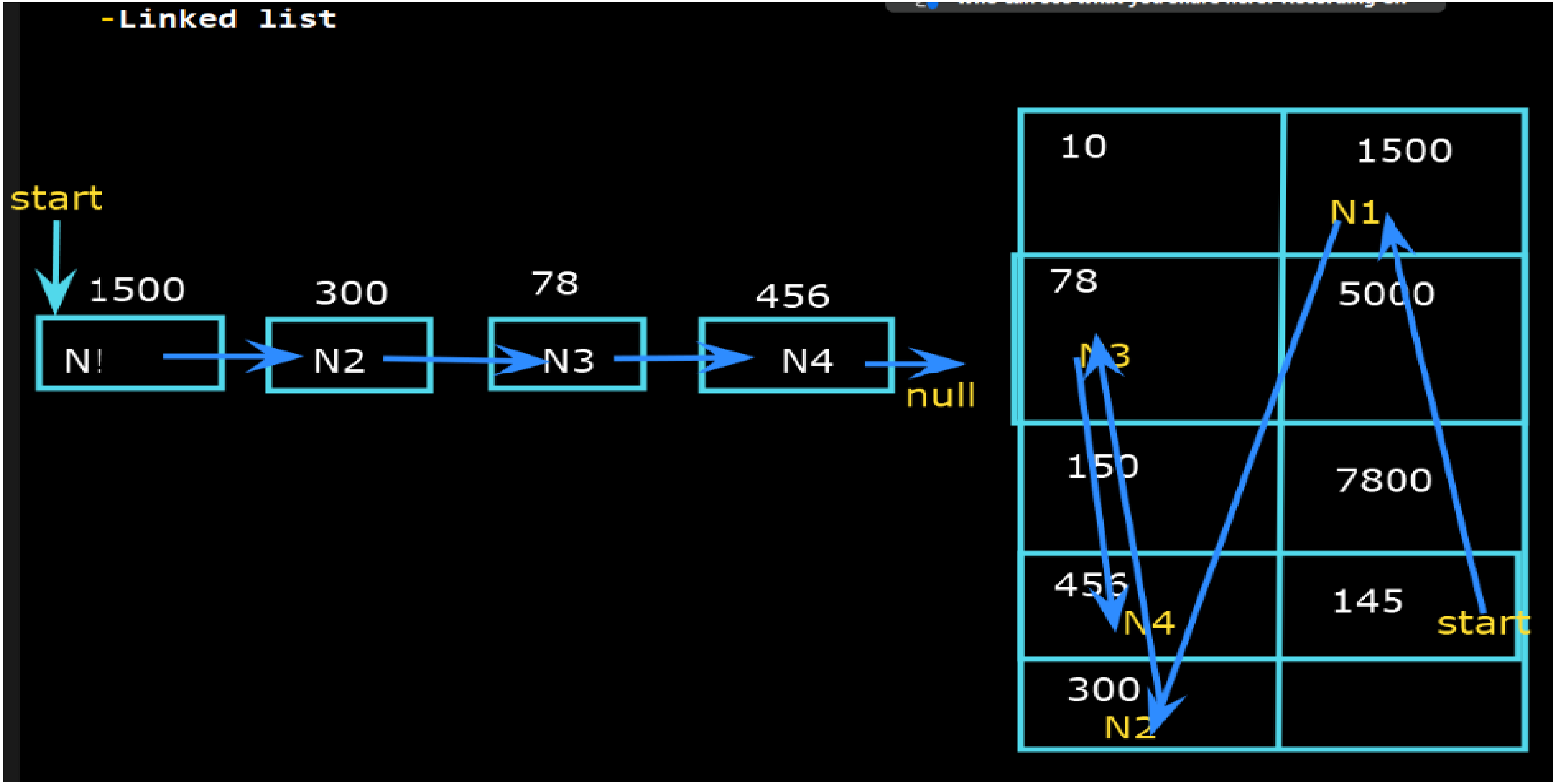




```
}
Linked list:
-------------
-sequence of data structures, which are connected together via links.
    -sequence of links
    -connections between nodes
    -most used data structure
Terms:
    -Link
    -Next
    -Data
    -Linked list
```



```
    -connections between nodes
    -most used data structure
Terms:
    -Link
    -Next
    -Data
    -Linked list
```

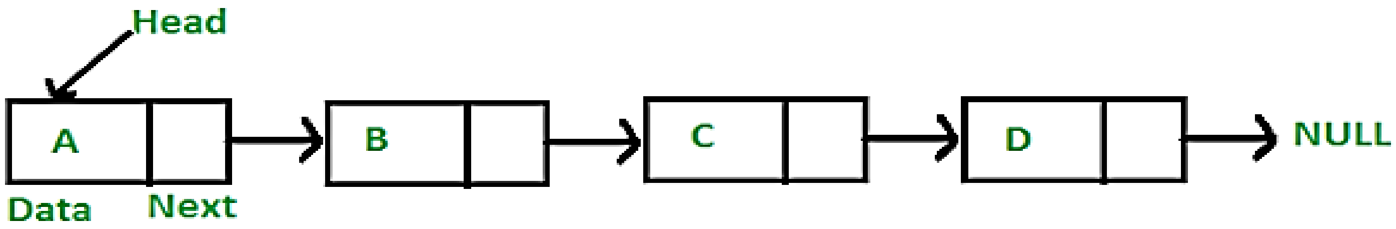


-Linked list

start
1500
N!
300
N2
78
N3
456
N4
null
10
1500
N1
78
N3
5000
150
7800
456
N4
145
start
300
N2

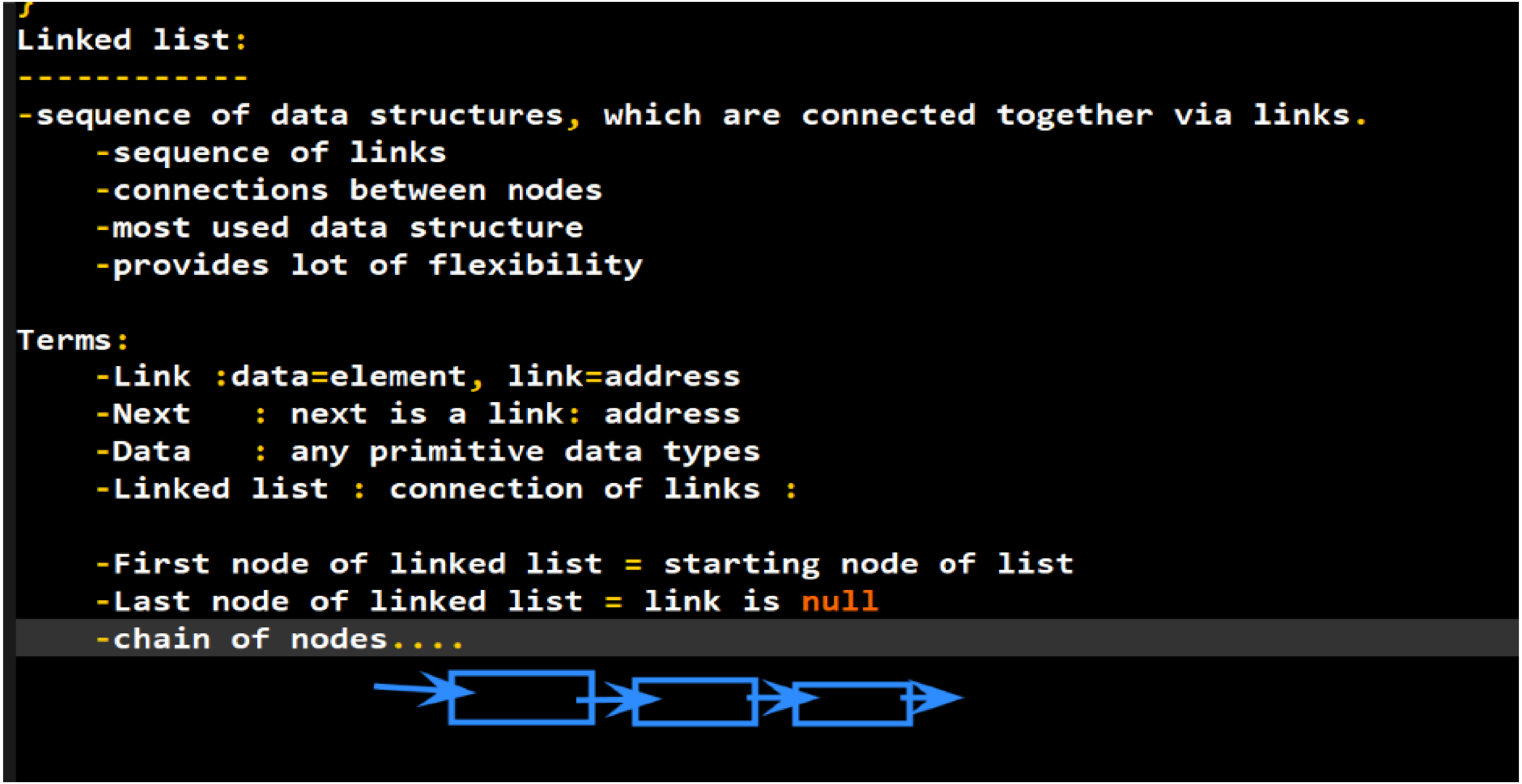
Linked List Representation

- Linked list can be visualized as a chain of nodes, where every node points to the next node.

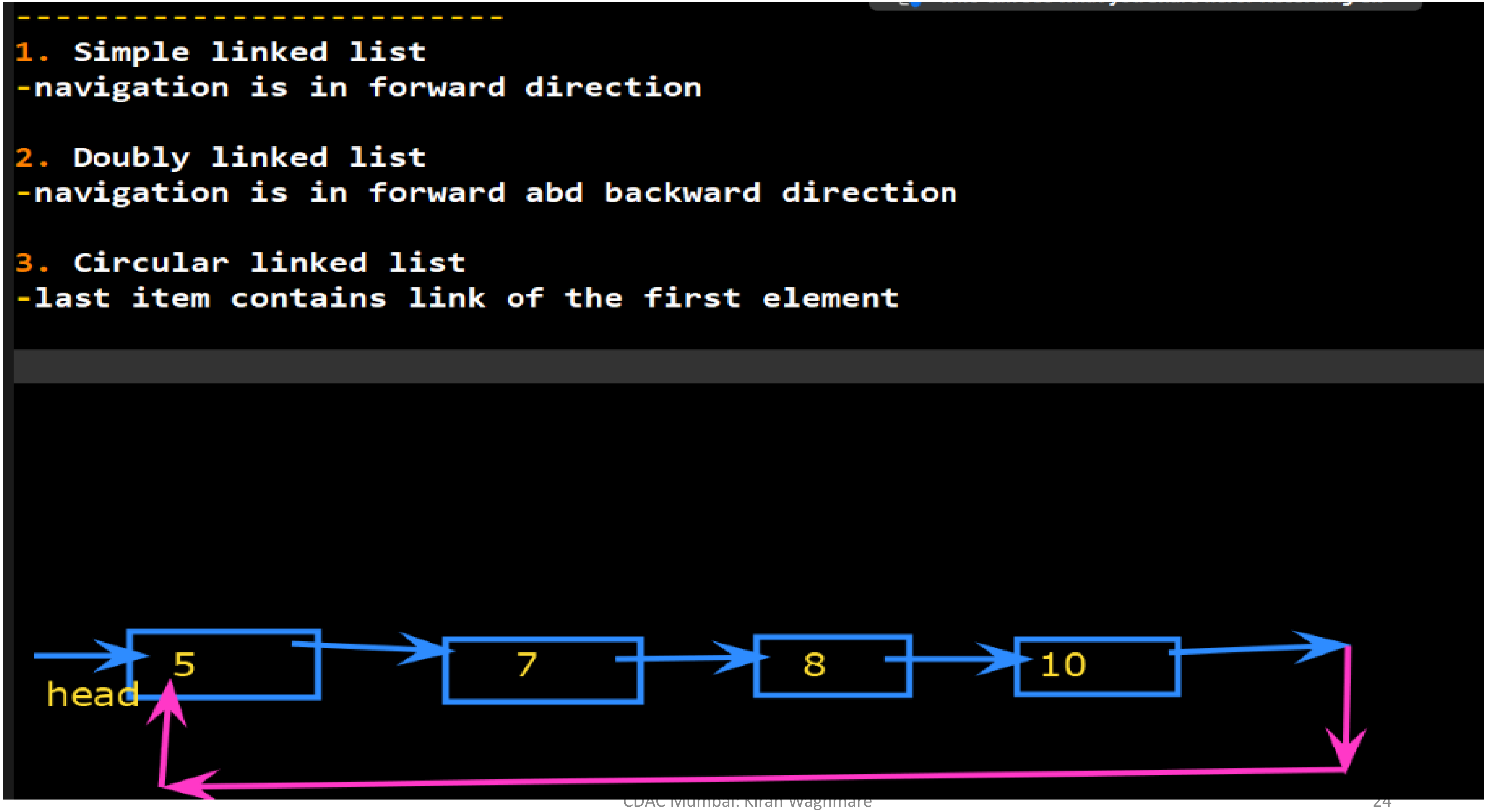


- As per the above illustration, following are the important points to be considered.

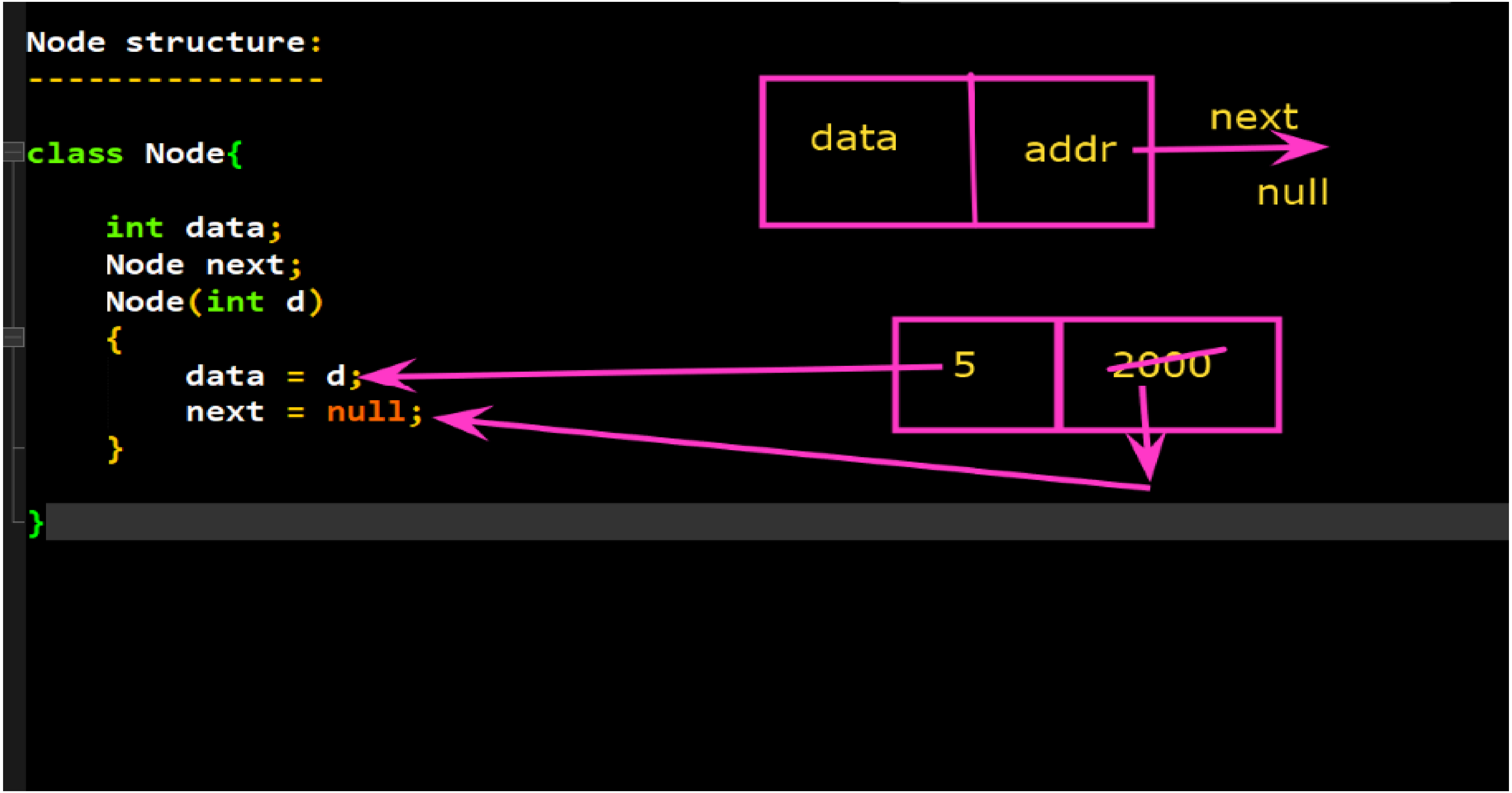
1. Linked List contains a **link element** called **first.**
2. Each link carries a **data field(s)** and a **link field** called **next.**
3. Each link is **linked with its next link** using its **next link.**
4. **Last link carries a link as null** to mark the end of the list.



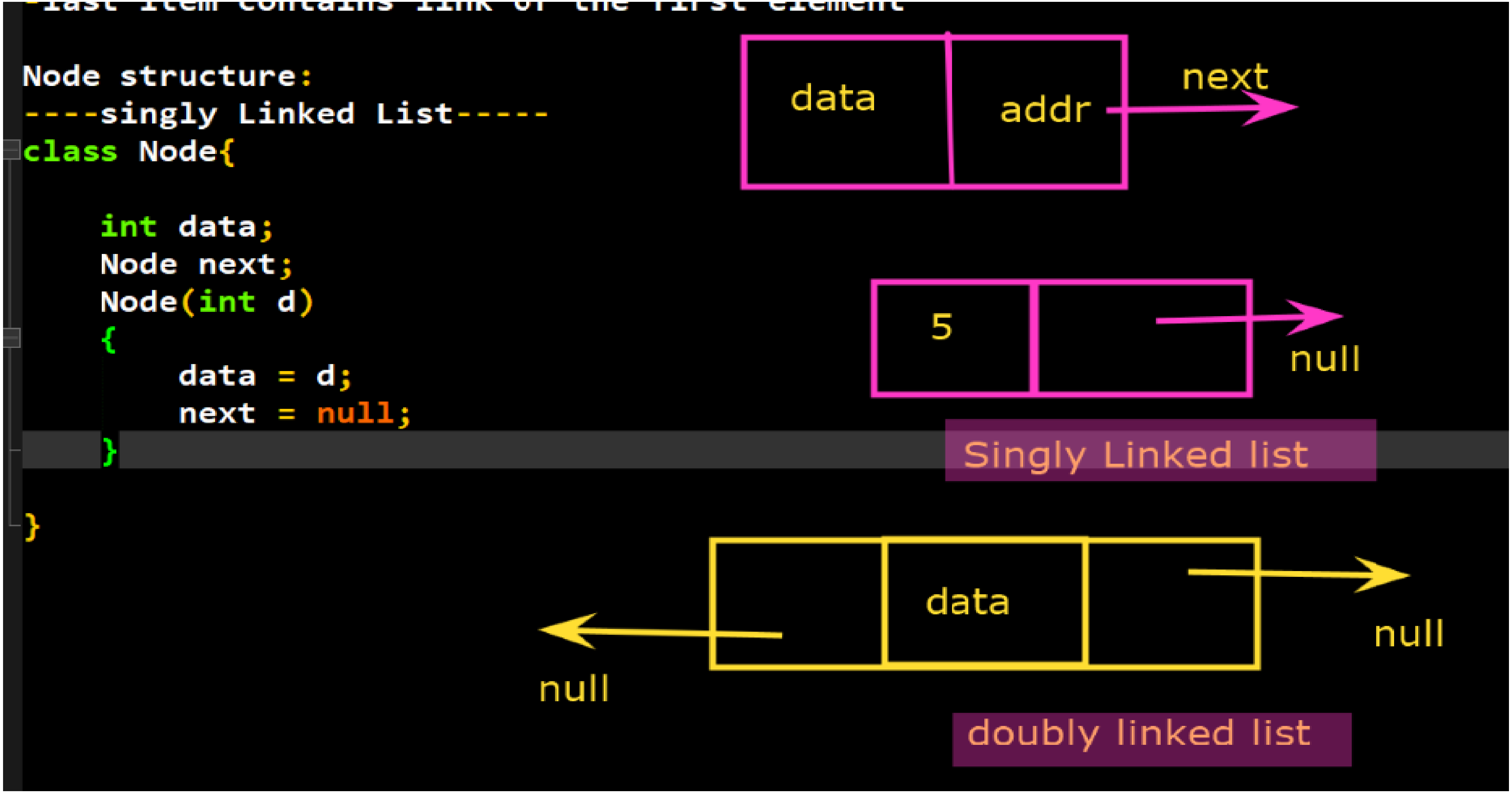
```
}
Linked list:
------------
-sequence of data structures, which are connected together via links.
    -sequence of links
    -connections between nodes
    -most used data structure
    -provides lot of flexibility

Terms:
    -Link :data=element, link=address
    -Next    : next is a link: address
    -Data    : any primitive data types
    -Linked list : connection of links :

    -First node of linked list = starting node of list
    -Last node of linked list = link is null
    -chain of nodes....
```



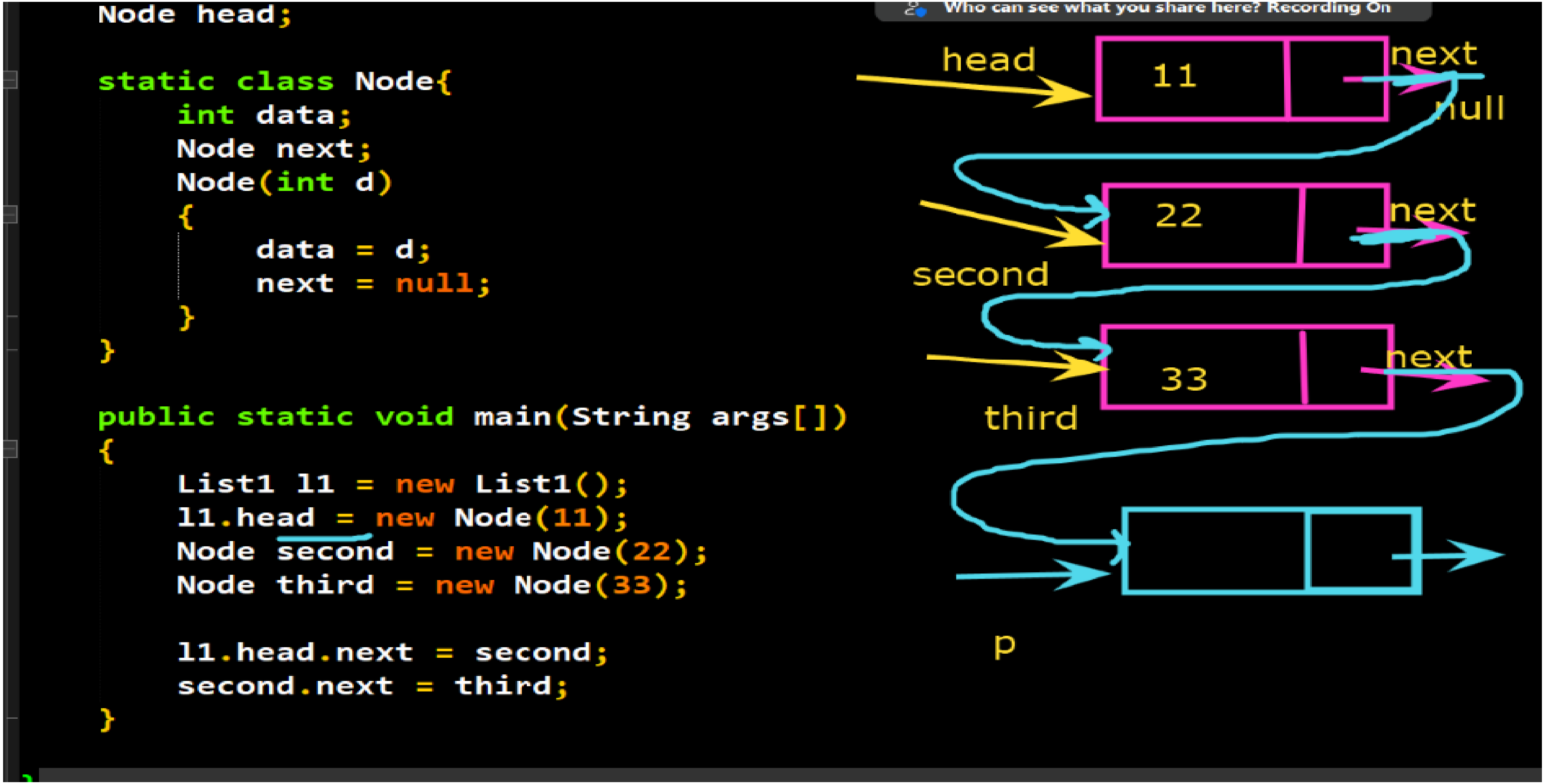
```
--------------------------
1. Simple linked list
-navigation is in forward direction

2. Doubly linked list
-navigation is in forward abd backward direction

3. Circular linked list
-last item contains link of the first element
```



```
Node structure:
---------------

class Node{

    int data;
    Node next;
    Node(int d)
    {
        data = d;
        next = null;
    }

}
```



```
-last item contains link of the first element

Node structure:
----singly Linked List-----
class Node{

    int data;
    Node next;
    Node(int d)
    {
        data = d;
        next = null;
    }

}
```



```
Node head;

static class Node{
    int data;
    Node next;
    Node(int d)
    {
        data = d;
        next = null;
    }
}

public static void main(String args[])
{
    List1 l1 = new List1();
    l1.head = new Node(11);
    Node second = new Node(22);
    Node third = new Node(33);

    l1.head.next = second;
    second.next = third;
}
```