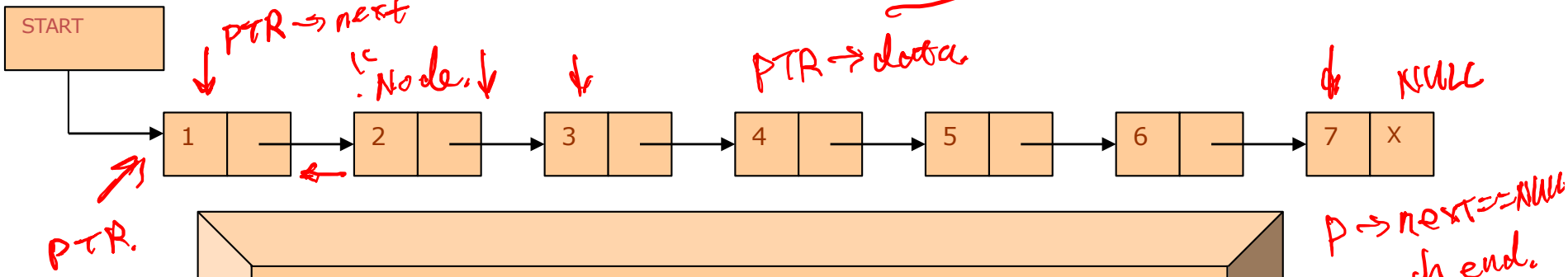
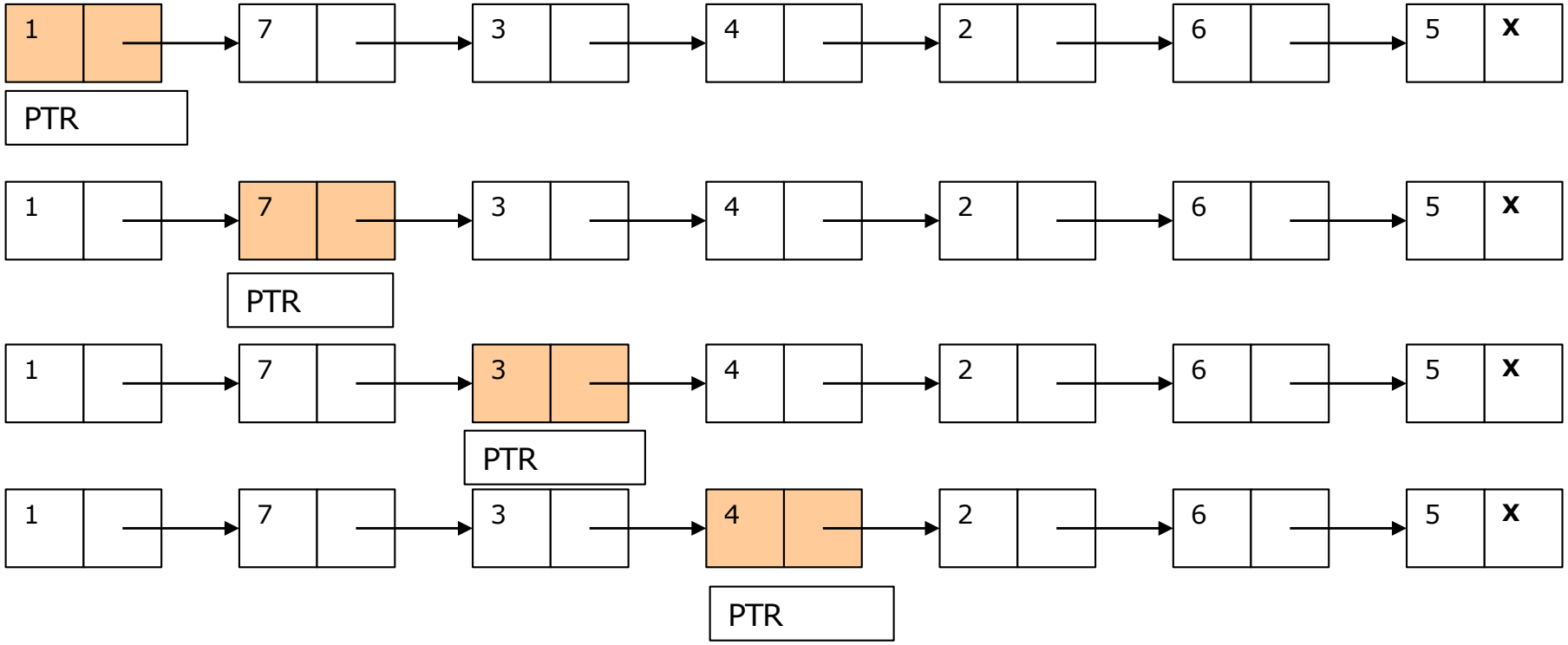
Singly Linked Lists

- A singly linked list is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same data type.
- Traversing a linked list means accessing the nodes of the list in order to perform some processing on them.



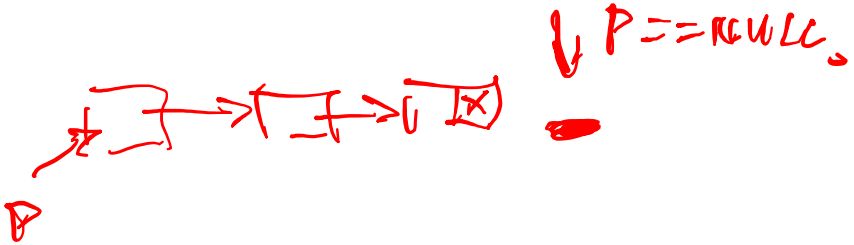

```
ALGORITHM FOR TRAVERSING A LINKED LIST

Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Steps 3 and 4 while PTR != NULL
Step 3:          Apply Process to PTR->DATA
Step 4:          SET PTR = PTR->NEXT
        [END OF LOOP]
Step 5: EXIT
```

Searching for Val 4 in Linked List



Here PTR -> DATA = 4. Since PTR -> DATA = 4, POS = PTR. POS now stores the address of the node that contains VAL.

Search.




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

struct Node* p == START;   O(1)
// traverse the linked list
while ( p != NULL )        O(n)
{
    if ( p->data == 4 ){
        printf("Found value\n");   O(1)
        return;
    }
    p = p->next;
}
```

$O(n) \cdot O(1) = O(n)$.

Time Complexity

$O(n)$. linear

↑ length of linked list.

→ `if p == NULL`

`printf("Not Found\n");` → $O(1)$

`return;`

Searching a Linked List

Searching a linked list means to find a particular element in the linked list.

```
ALGORITHM TO SEARCH A LINKED LIST

Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Step 3 while PTR != NULL
Step 3:           IF VAL = PTR->DATA
                            SET POS = PTR
                            Go To Step 5
                  ELSE
                            SET PTR = PTR->NEXT
                  [END OF IF]
        [END OF LOOP]
Step 4: SET POS = NULL
Step 5: EXIT
```

Inserting a New Node in a Linked List

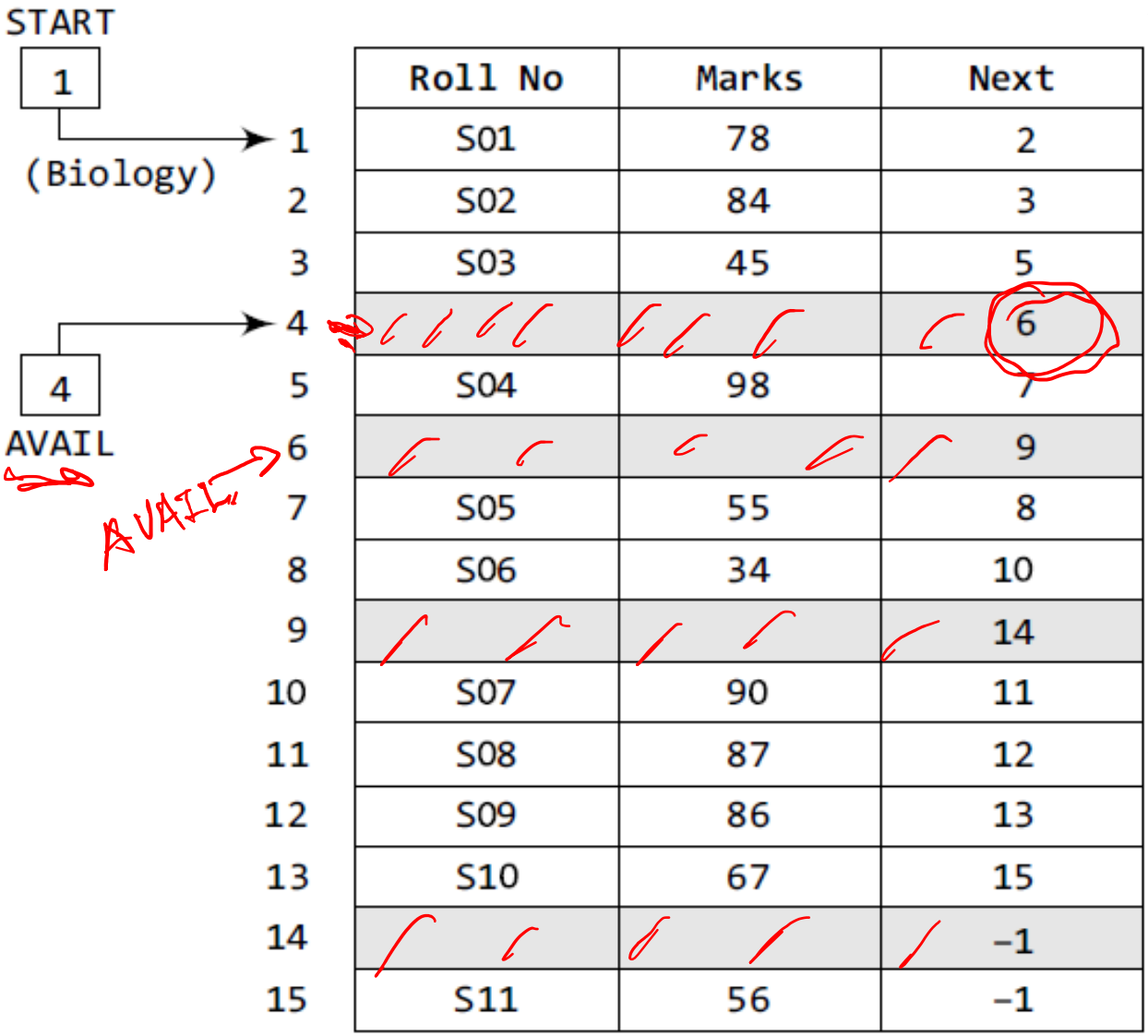
- Four cases
  - Case 1: The new node is inserted at the beginning.
  - Case 2: The new node is inserted at the end.
  - Case 3: The new node is inserted after a given node.
  - Case 4: The new node is inserted before a given node.

- Overflow is a condition that occurs when AVAIL = NULL or no free memory cell is present in the system.
  - When this condition occurs, the program must give an appropriate message.
  - AVAIL is a pointer which stores the address of the first free space.

Memory Allocation and De-Allocation

- If we want to add a node to an already existing linked list in the memory, we first find free space in the memory and then use it to store the information.
- Now, the question is which part of the memory is available and which part is occupied? When we delete a node from a linked list, then who changes the status of the memory occupied by it from occupied to available? The answer is the operating system.
- The computer maintains a list of all free memory cells. This list of available space is called the *free pool*.

Free Pool or Heap



	Roll No	Marks	Next
1	S01	78	2
2	S02	84	3
3	S03	45	5
4			6
5	S04	98	7
6			9
7	S05	55	8
8	S06	34	10
9			14
10	S07	90	11
11	S08	87	12
12	S09	86	13
13	S10	67	15
14			-1
15	S11	56	-1

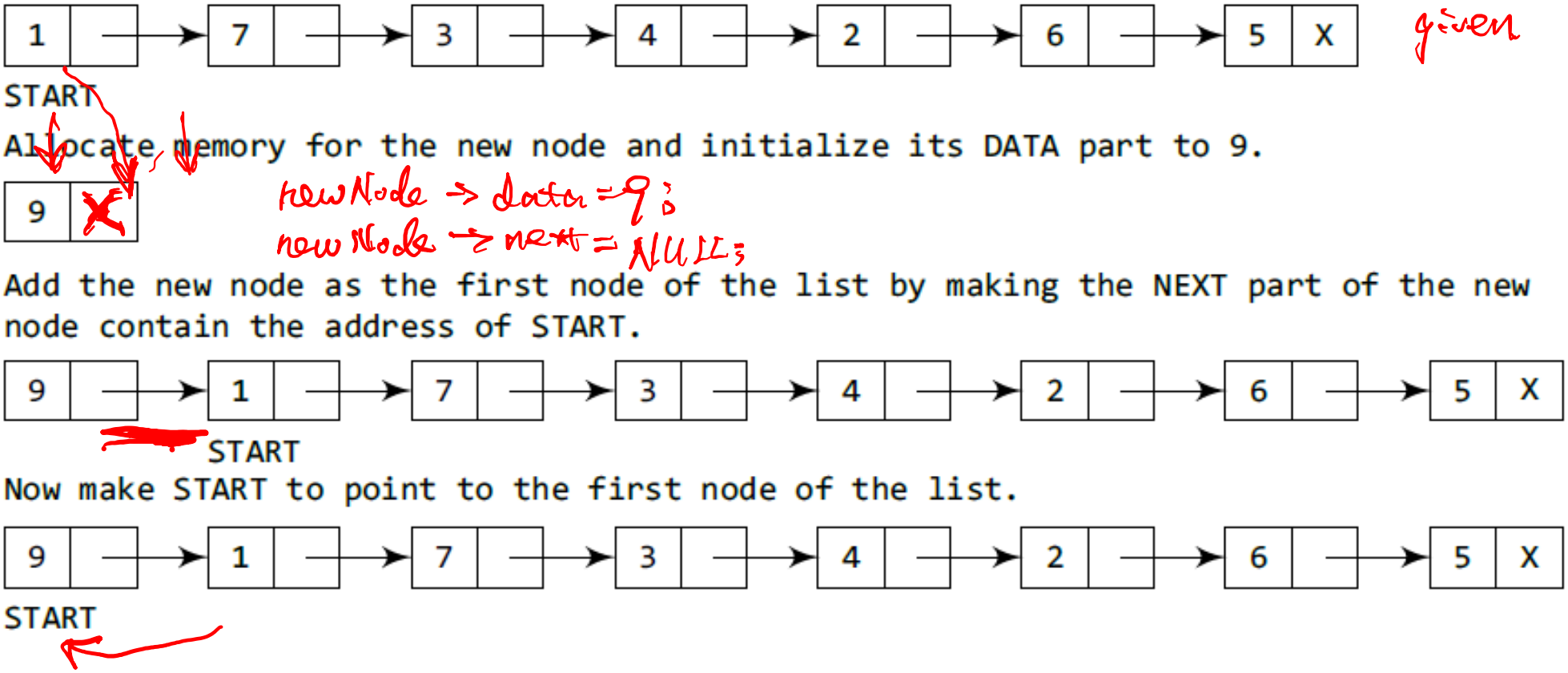
- For the free pool (which is a linked list of all free memory cells), we have a pointer variable AVAIL which stores the address of the first free space.
- Now, when a new student's record is added, the memory address pointed by AVAIL will be taken and used to store the desired information.
- After the insertion, the next available free space's address will be stored in AVAIL.
- For example, when the first free memory space, 4, is utilized for inserting the new node, AVAIL will be set to contain address 6.

// overflow C. Language.

```
struct Node * newnode = (struct Node*) malloc (sizeof(struct Node));
  if (newNode == NULL).
   { printf("Memory overflow");
     return;
   }
```

Inserting a Node at the Beginning

Suppose we want to add a new node with data 9 and add it as the first node of the list.



Inserting a Node at the Beginning

```
ALGORITHM TO INSERT A NEW NODE IN THE BEGINNING OF THE LINKED
LIST

Step 1: IF AVAIL = NULL, then
                Write OVERFLOW
                Go to Step 7
        [END OF IF]
Step 2: SET New_Node = AVAIL
Step 3: SET AVAIL = AVAIL->NEXT
Step 4: SET New_Node->DATA = VAL
Step 5: SET New_Node->Next = START
Step 6: SET START = New_Node
Step 7: EXIT
```

// given. newNode.

newNode -> data = 9;

newNode -> next = NULL;

// step 2 :

newNode -> next = START;

// step 3 : Update header

START = newNode;

① START == NULL, empty linked list.

√.

Time Complexity

O(1).