Date ___/___/_____

## Set -2.

1. Define time - space tradeoff with an example. 3 M

⇒ A tradeoff is a situation when if one thing increases then another thing decreases. vice-versa.

⇒ It is a way to solve a problem in less time but more memory or use less memory then it will take more time.

⇒ But An ~~ideal~~ Algorithm is that which requires less memory and less time to produce an output.

⇒ But in general, it is not always possible to achieve both.

Example :

```
① for (int i=0; i<10; i++){
      cout << "hello";
  }
```

```
② cout << "hello";
  cout << "hello";
  :
  : 10times
  :
  cout << "hello";
```

⇒ Both will give the same output but 1st one is memory efficient and second one is time efficient.

⇒ It general its very difficult or not always possible to achieve both efficiency.

SM

Date ___/___/_____

2. Discuss the applications of multidimensional arrays with suitable examples. 3 marks

⇒ A mult-D array is an array with more than one index.
arr[row][col], arr[layer][row][col] --- etc.

Applications ⇒

(A) Matrix and Linear Algebra;
⇒ Used to store and perform ~~operations~~ operations on matrixes.
e.g: addition, multiplication, transpose.

(B) Image Processing:
⇒ Images are represented as 2D or 3D arrays.
⇒ Each pixel has a value.
e.g image[3][3] = {{0,255,0},{255,0,180},{0,220,0}}.

(C) Game Development:
⇒ Board games like chess, tic-tac-toe etc.

(D) Scientific and Engineering Applications:
⇒ 3D arrays store data in simulations: temp, pres, or fluid flow in 3D space.

(E) Data Representation:
⇒ Multi-D arrays can represent table or database in memory or records.
e.g Marks of students.

SM

Date ___/___/_____

(F) Sparse Matrix storage:

⇒ Sparse matrix use 2D array for efficint storage.

3. Derive index formula for a 1-D array and explain its significance. 4 M

Let arr[n]

we have to find address of arr[i];

and size of element is W

then

Add E arr[i] = Base address + W × (i − LowerBound)

So

$$\boxed{\text{Add arr}[i] = B + W \times (i - LB)}$$

LB is the lower bound if not specified then LB = 0.

Significant of formula:

1. Direct access

→ Using this formula, we can direct access any element in O(1) time.

2. Memory efficiency

→ knowing the base address and elem size we don't need to store addresses for every elements.

3. Pointer Arithmetic

This formula is basis of pointer arithmetic.

arr+i // points to arr[i]

*(arr+i) // value at arr[i].

SM

Date __/__/____

4. Faundation for Multi-D arrays.

→ This 1-D index formula is extend to 2D and 3D arrays.

4. Advantages and Limitations of Linear search algorithm. 4M

→ Advantages!

1. Simple and easy implement.
2. Works on both sorted and unsorted array.
3. No preprocessing required.

→ Limitations:

1. Inefficient for large arrays O(n) TC.
2. Takes more comparisons than efficient algos like B.S.
3. Sequential access only - Can't skip elems.

5. Explain Quick sort algorithm and give an example of partitioning. 5M

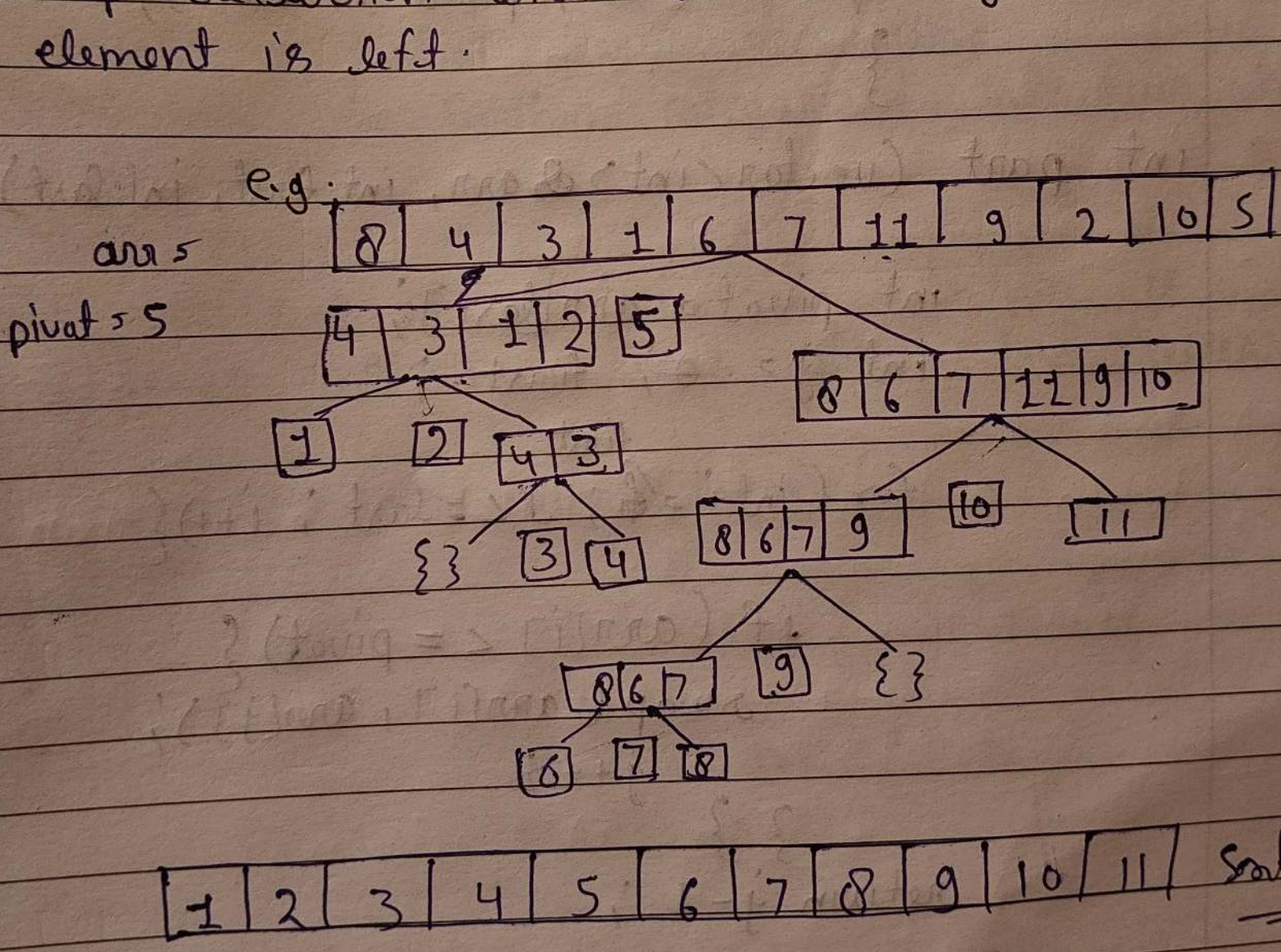
⇒ QuickSort is a sorting algorithm based on the divide and conquer that picks an element as a pivot and partitions the given array around the pivot by placing pivot in its correct position in the sorted array.

Date ___/___/_____

Algorithm!

1. choose a pivot
2. partition the array around pivat.
3. After partition, it is ensured that, all elements that go are greater than pivot are in right and less than pivot elements are left.
3. Recursively call for the two partitioned left and right subarrays.
4. Stop recursion when there is only one element is left.

e.g.



Date __/__/____

Code

```
void quicks (vector<int>& arr, int first, int last){
    if (first >= last) {return ;}
    int pivat = part(arr, first, last);
    quicks (arr, first, pivot-1);
    quicks (arr, pivat+1, last);
    }
int part (vector<int>& arr, int first, int last) {
    int pivot = arr[last];
    int j = first ;
                first
    for (int i = first ; i <= last ; i++){
        if (arr[i] <= pivot) {
            swap ( arr[i], arr[j]);
            j++ ;
        } }
    return j-1 ;
}
```

TC = O(n log n) (avg case)
O(n²) (worst case)

Date __/__/____

6. Discuss the different type of recursion with examples. 5 M

⇒ Recursion is mainly divided into two major types.

1. Direct Recursion
2. Indirect Recursion

1. Direct Recursion
When a function calls itself from within itself is called direct recursion.
It can be further categorized into four types:

i) Tail Rec
⇒ If the last statement in the function is recursive call then, it is called tail recursion.

```
void printN(int n) {

    if (n<=0) { return; }

    cout<<n<<" ";
    printN (n-1);
}

// printN (5)
// 5 4 3 2 1
```

Date __/__/____

ii) Head Rec

⇒ If the recursive call is ~~not~~ the first ~~last~~ statement of function then it is called ~~to~~ head recursion.

```
void printN (int n){

    if (n<=0) { return ; }

    printN (n-1);
    cout<< n<<" ";
    }

// printN(5)
// 1 2 3 4 5
```

iii) Tree Rec

When a function calls itself more than once in each call.

```
void fun (int n) {

    if (n==0) { return ; }

    cout <<n<<" ";
    func(n-1);
    func(n-1);
}
```

Date __/__/____

iv) Nested Rec
When a function passes a recursive call as parameter. Means Recursion inside recursion.

```
~~voi~~ int func(int n){
        if (n<=1) return 1;
        return func(func(n-1));
        }
```

2. Indirect Rec
When a function calls another function and that function calls the first one again.

```
void A(){
    cout<<"A";
    B();
}
```

```
void B(){
    cout<<"B";
    A();
}
```

Date __/__/__

7. Explain insertion and deletion operations in doubly linked lists with diagrams. 8 M.

⇒ Insertion

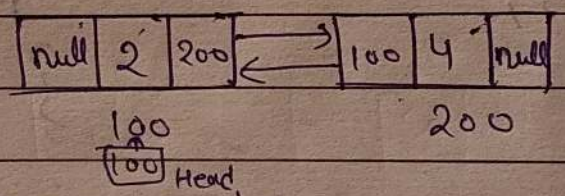
We can insert a node at

i) begining :

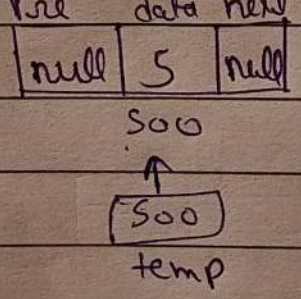
→ Algo :

① Create a new node
② Make newNode -> next = head
③ Set newNode->pre = NULL;
④ Update head -> prev = newNode
⑤ Finally move head = newNode.

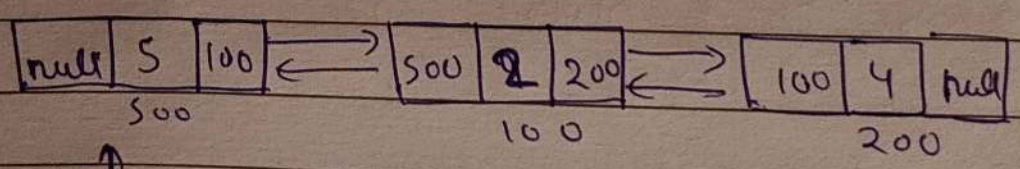
Let a Linked List




add a new node




now temp -> nex = head
head -> prev = temp ; head = temp
then it will be

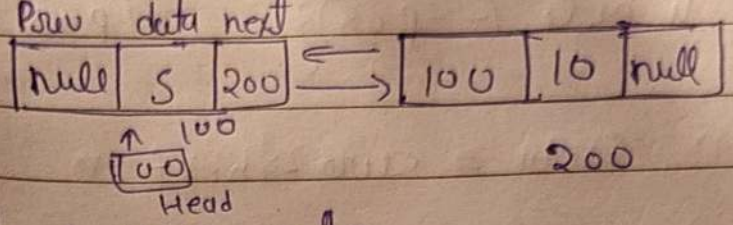



SM

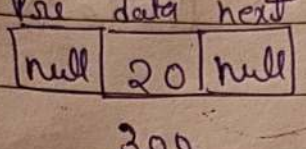
Date __/__/____

ii) Ending :

Let a Linked list

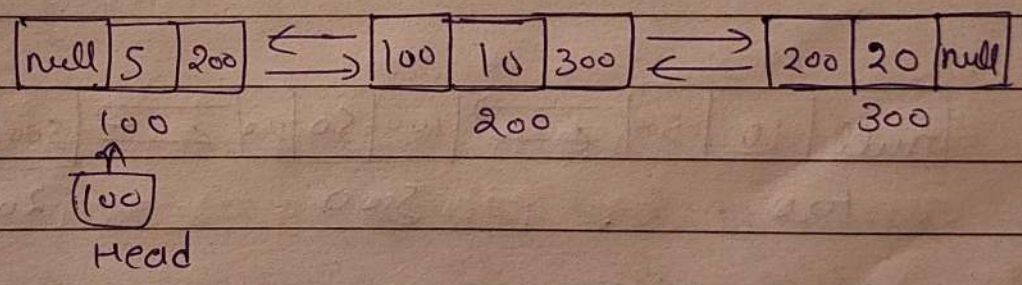



now add a new node



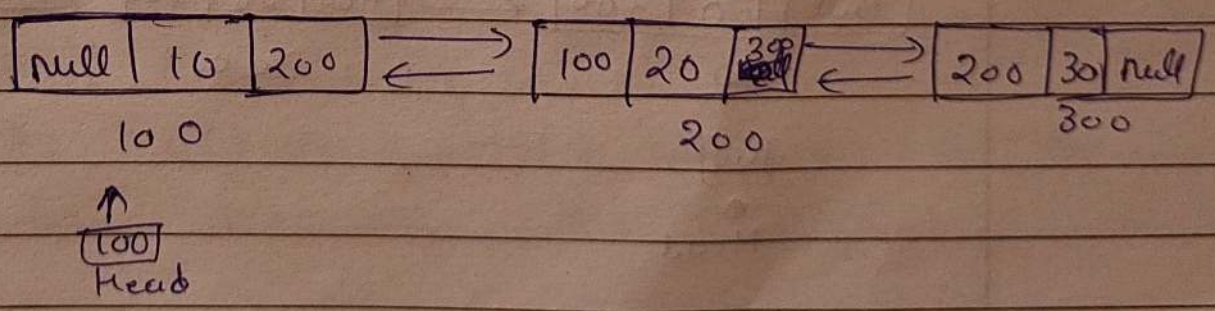

```
Node* curr = head;
while (curr->next != null)
  { curr = curr->next; }
Node * temp = new Node (5);
curr->next = temp;
temp->pre = curr;
```

So now




(ii) At specific pos

Let a Linked List



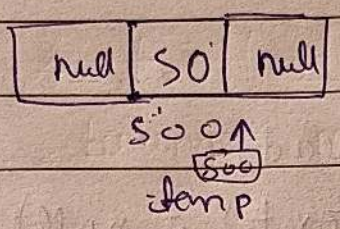

⇒ insert at 2nd pos. value = 50

SM

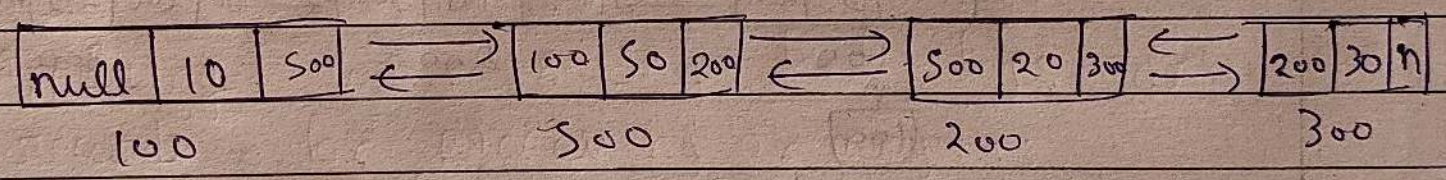
Date ___/___/_____

⇒ So

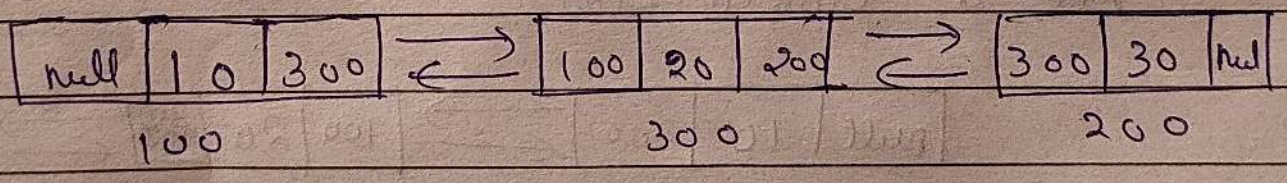
```
Node * curr = head;

while (--pos){          // pos = 1 then
    curr = curr->next
}

Node * temp = new Node (50);
```

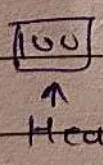



```
temp -> next = curr -> next;
temp -> pre = curr;
curr -> next = temp;
temp -> next -> pre = temp;
```




## 2. Deletion :




⇒ now we have to delete first node.

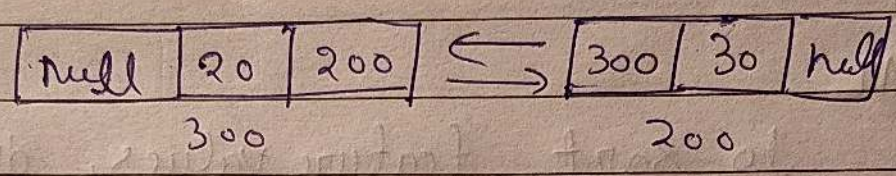
SM

Date ___/___/_____

```
Node * temp = head;

head = head -> next
delete temp;
head -> prev = NULL;
```

this will move head to next node then delete temp.

so




8(a) Write an iterative sol for calculating fibbonacci Numbers. 4M

Set 5 Q 6

8(b) Explain removal of recursion.

Set 4 Q 6 4M

Q.9. Discuss the role quick sort in real time applications. Compare it with merge sort in terms of efficiency. 10M

⇒ Quick sort Applications:

i) System - Level Programming (C/C++)

⇒ Used in the built-in qsort() function of the C standard libraray for sorting arrays.

Date __/__/____

ii) Database and search engines
⇒ Used to sort data efficiently before binary search, indexing or joining tables.

iii) Big data processing
⇒ Helps in parallel sorting on large data bases

iv) AI/ML
⇒ Used to sort feature values, distances or probabilities during model training and prediction.

v) Real Time Systems
Performs fast in place sorting where low memory usage ~~is~~ is critical.

vi) File and log sorting
Applying when sorting text, logs or records for processing or merging.

Date __/__/____

# Comparison Quick v/s Merge sort.

| Features | Quick sort | Merge sort |
|---|---|---|
| Approach. | Divide & conquer (partition around pivot) | Divide & conquer partition ~~around~~ mid & merge |
| Worst TC | $O(n^2)$ (when pivot selection poor) | $O(n \log n)$ (always stable) |
| Avg/Best case | $O(n \log n)$ | $O(n \log n)$ |
| Space comp | $O(\log n)$ → In place (recursion stack only) | $O(n)$ extra space for merging. |
| Stability | Not stable | stable (maintains order) |
| Use | when memory limited and data set is mostly unsorted | When stable sort needed or external sorting (like files) |