Set - 2

1. Principle in CS where algorithm's performance (time) can be improved at cost of using more memory (space) or its memory usage can be reduced at cost of taking longer to run.

a) High time, Low space
To calculate sinx using taylor series

b) Low time, High space
Pre-calculate all values of Sinx then store them in an array

2. Matrices - 2D array is standard way to represent matrices for operations like addition or multiply.

(i) Image processing - 2D array represent grayscale image while 3-D array represent coloured image (rows, columns, RGB colours)

(ii) Game boards - 2D arrays are used to represent games like chess, tic-tac-toe storing state of each state

3. B - Base address, A[0]
S - Size of one element in bytes
i - index of element we want to access

$$Address(A[i]) = B + (i \times S)$$

It provides O(1) (constant time) random access. To find any element, it performs one addition & one multiply. It doesn't need to iterate through array.

4. Algorithm

LS(arr, target)

For i = 0 to length(arr) - 1

If arr[i] == target then

return i;

END If

END For

Return -1

Advantages -

(i) Very easy to understand & implement

(ii) works for any type of array (sorted or unsorted)

Limitations -

(i) Inefficient - time complexity O(n). In worst case, it checks every element

(ii) Slow for large datasets

5. ~~Algorith~~ Algorithm

(i) Pivot - Choose element from array

(ii) Partitioning - Rearrange all elements so that smaller value element before pivot and rest after pivot element

(iii) Recurse - Recursively apply quick sort algorithm to two smaller sub-arrays.

(iv) Base case - Recursion stops when sub-array has zero or one element i.e. already sorted.

~~Partition~~ Partitioning example

array [4, 7, 2, 1, 5, 3]

pivot = 3

(i) initial i=-1, j=0
(ii) j=0: Array [j] is 4. Do nothing
(iii) j=1: Array [j] = 7. Do nothing
(iv) j=2: Array [j] = 2. Swap [i] & Swap [j]
swap (4, 2)
(v) j=3: Array [j] = 1. Swap array [i] & array [j]
swap (7, 1)
(vi) j=4: Array [j] = 5. Do nothing
(vii) Loop ends
(viii) Final swap pivot(3) with element at i+1 (4)
Swap (4, 3) → [2, 1, 3, 7, 5, 4] (i+1)

6. (i) Direct recursion

```
int factorial (int n) {
  if (n==0) return 1;
return n* factorial (n-1); }
```

(ii) Indirect recursion

```
boolean isEven (int n){
  if (n==0) return true;
return isOdd (n-1); }
boolean isOdd (int n){
  if (n==0) return false;
return isEven (n-1); }
```

(iii) Tail recursion

```
int fact (int n, int acc){
  if (n==0) return acc;
return fact (n-1, n*acc); }
```

(iv) Head recursion

```
void printN(int n){
  if (n>0){
    printN(n-1);
    SOUT(n); }
```

8. a) Fibonacci sequence

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

```
int fibo(int n){
  if (n<=0) return 0;
  if (n==1 ) return 1;
    int a = 0;
    int b= 1;
    int current =0;
  for (int i= 2;i<=n;i++){
   current = a+b;
     a = b;
    b= current; }
    return b; }
```

b) Removal of recursion is process of converting a recursive algorithm into iterative one.

Main reasons -

(i) To prevent stack overflow - deep recursion can use too much memory on call stack causing program to crash.

(ii) To improve performance - Function calls are overhead. Iteration solution is faster

From
```
int fact(int n){
  if (n==0){
    return 1;}
  return n* fact(n-1);}
```

To
```
int fact (int n){
  int result = 1;
  for (i=n; i>0; i--){
    result = result*i; }
  return result; }
```

9. Linked list representation of sparse matrix

(i) Create array where each corresponds to row of matrix

(ii) Each element of this array is head pointer to linked list

(iii) Each node in linked list stores non-zero element from that row -

a) Column index

b) value

c) A next pointer

Demonstration for 4x5 sparse matrix

	col 0	col 1	col 2	col 3	col 4
Row 0	[0,	0,	9,	0,	0]
Row 1	[5,	0,	0,	0,	1]
Row 2	[0,	0,	0,	0,	0]
Row 3	[0,	2,	0,	0,	0]

Benefits of this approach -

(i) Massive space saving - For 1000 x 1000 matrix, 2 D array stores every value whereas linked list only stores 500 nodes for 500 non-zero elements.

(ii) Dynamic size - simply delete or add node without resizing large array

(iii) Efficient operation - We can skip any operation on zero elements