SET- 2 (12)

Q (1) Time & space trade-off means using more memory (ie. space) to make an algorithm run in less time or using less memory but taking more time.

Ex→ In merge sort, we take extra space as we make a temp array but it works faster.

Q (2) Application of Multi Dimensional array -

→ ★ Used to store data in table or matrix form
★ Used in mathematical operation like addition & multiplication
★ 2D arrays store pixel values of images
★ 2D arrays represent grids or chess boards

Ex→ Matrix operations :-
2D array to store a 3x3 matrix
int matrix [3][3];

Ex→ Image Processing :-
Grayscale image int image [h][w];

Ex→ Game dev :-
char board [8][8]

Q (3) 1-D array index formula.
In memory, array elements are stored contiguously.

Loc [A[i]) = Bas (A) + (i * w)

where, Bas (A) = Base address
i = Index
w = Size in bytes

Date
Page No.

Ex→ Loc[A[4]] = 1000 + (4 * 4)
B(A) = 1000 = 1016
w = 4 bytes

Significance
→ Enables random access to elements in O(1) time.
→ Helps compilers & system locate elements quickly.

9) Algorithm (Linear Search)
(1) Start from the first element of the array.
(2) Compare the target element with each element one by one.
(3) If match found, return the index of that element.
(4) If not found, even after checking all elements, return -1.

Pseudocode

```
function LS(arr, target)
  for (i to length of arr - 1)
    if (arr[i] == target)
      return i;
    else
      return -1
```

Advantages
* Easy to understand & implement
* Works on both sorted & unsorted lists
* No extra memory required

Limitations
→ Slow for large dataset O(n).
→ Inefficient when compared to Binary Search on sorted data.

Q.(5) Quick sort algorithm

QS is a divide & conquer algo that sorts an array by partitioning it into subarrays around a pivot

⇨ STEPS:-

(1) Choose a pivot element from the array.

(2) Partition the array so that (Partitioning)
   (a) All elements smaller than pivot go to its left.
   (b) All elements greater than pivot go to its right.

(3) Recursively apply the same process to the left & right subarrays.

(4) The process continues until each subarray has one element.

eg of partitioning — [10, 80, 30, 90, 40, 50, 70]

After parti. ⇒ [10, 30, 40, 50, 70, 80, 90]

where pivot = 70

⇨ Pseudocode

```
int partition (int [] arr, int low, int high)
{  int pivot = arr [high]
        i = low - 1
   for (j = low → j < high)
       if (arr[j] < pivot)
           i++
           swap (arr[i], arr[j])
   swap [arr[i+1], arr[high]
   return (i + 1);
}
```

Date
Page No.

6) Different types of Recursion

(1) Direct Recursion – A function calls itself directly.

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

(2) Indirect Recursion – A function calls itself after calling another function first.

Ex →
```
void funcA (int n){
    if (n>0) funcB(n-1);
}
void funcB (int n){
    if (n>0) funcA(n/2);
}
```

(3) Tail Recursion – The recursive call is the last statement in the function.

* Can be optimized by the compiler.

Ex →
```
int sum(int n, int acc){
    if (n==0) return acc;
    return sum(n-1, acc+n); // Last call
}
```

(4) Non-Tail Recursion – Recursive call is not the last statement.

Ex → factorial function ex in (1)

7) Double Linked List (DLL)

each node contains
- → data – the value stored
- → next – pointer to the next node
- → prev – pointer to the previous node

→ It allows traversal in both directions.

(A) Insertion in DLL

1) Create a new node.

2) Adjust the prev & next pointers of the node & its neighboring nodes

Before ins. 10 ⟷ 20 ⟷ 30

After ins. 10 ⟷ 20 ⟷ 25 ⟷ 30

Insert 25 after 20.

3) if inserting at

Beginning - Update head pointer
Set next to current head.
Set prev to NULL.

End - Update last node's next pointer
Set prev to last node
Set next to NULL.

Middle - Update pointers of previous & next nodes accordingly.

(B) Deletion in DLL

1) Find the node to delete

2) Adjust the next of the previous node & prev of the next node to bypass the node

3) Free/delete the node

Before del:- 10 ⟷ 20 ⟷ 25 ⟷ 30

10 ⟷ 20 ⟷ 30

Date
Page No.

(Ques 8) (a) iterative solution for calculating Fibonacci numbers 0, 1, 1, 2, 3, 5, 8, 13, 21...

```
int fib (int n) {
  if (n==0){
    return 0;}
  if (n == 1) {
    return 1;}
  int fib0 = 0, fib1 = 1, fibx = 0;
  for (int i = 2; i <= n; i++) {
      fibx = fib0 + fib1;
      fib0 = fib1;
      fib1 = fibx;
  }
  return fibx;
}
```

(b) Removal of Recursion

Recursion can be replaced by loops to save stack memory.

Recursive calls are converted to iterative constructs like 'for' or 'while'.

Example-

```
int fib (int n) {
  if (n==0) return 0;
  if (n==1) return 1;
  return fib (n-1) + fib (n-2);
}
```

problems $O(2^n)$

Iterative approach like above in part (a) avoids recursion overhead.

T.C $\Rightarrow O(n)$

S.C reduces from $O(n)$ stack space to $O(1)$.

(Q.9) Sparse Matrix Storage Using Linked List

1. Sparse matrix $\begin{bmatrix} 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}_{4\times5}$

2. Linked List Representation
each node contains
row, col, data, next
↳ pointer to the next non-zero element

⇨ Code →

```
class Node {
int row, col, data;
Node next;
Node (int r, int c, int d) {
   row = r; col = c; data = d;
   next = null;
  }
 }
```

⇨ LL representation of the example

Node (row = 0, col = 2, data = 3) →
Node (row = 2, col = 1, data = 7) →
Node (row = 3, col = 4, data = 1) → null

diagram
↳ [0, 2, 3] → [2, 1, 7] → [3, 4, 1] → null

⇨ Advantages of LLR
→ Space Efficient
→ Dynamic Size
→ Easy Traversal of Non-Zero elements.
→ Reduces Wasted Storage.

Date
Page No.

9) Role of Quick Sort in Real-Time Applications
Used widely because of its average-case efficiency & in-place sorting.

1. Database Query Optimization —
Sorting records quickly to improve search & retrieval.

2. Network Routing Tables —
Maintaining sorted routing info for fast lookups.

3. Embedded Systems —
Used in memory-constrained environments because no extra array is needed.

4. Real-Time Gaming —
Sorting scores, events or objects efficiently without large memory overhead.

| Quick Sort | Merge Sort |
|---|---|
| ★ T.C → O(nlogn) avg | ★ T.C → O(nlogn) avg |
| ★ T.C → $O(n^2)$ worst | ★ T.C → O(nlogn) worst |
| ★ S.C → O(logn) | ★ S.C → O(n) |
| ★ Not stable | ★ Stable |
| ★ Faster for in-memory array | ★ Slightly slower due to extra memory |
| ★ Uses → | ★ Uses → |
| (★) Real time, memory-sensitive tasks | (★) When stable sorting is required, or linked lists |