Assignment-1

Section-A

1. What are time Complexity and Space Complexity of an algorithm?

ans · Time Complexity measures the amount of time an algorithm takes relative to the input size.

· Space Complexity measures the amount of memory an algorithm uses relative to the input size.

2. How Substitution method works for solving recurrence?

· In the Substitution method, we guess the solution and then prove it using mathematical induction. It involves substituting the guessed form into the recurrence to verify if it satisfies the recurrence.

3. What are the conditions for which master method is not applicable?

· master method is not applicable when:

· The recurrence is not in the standard form:

$$T(n) = aT(n/b) + f($$

· The Subproblem sizes are not equal or division is irregular.

· f(n) is not a polynomial or regular function.

4. What is recurrence? What are the methods to solve a recurrence?

- A recurrence is an equation that defines a function based on its values at smaller inputs.
- methods to solve:
    - Substitution method
    - Recursion Tree method
    - master method

Section-B

1. Write a recursive & non-recursive algorithm for binary search and explain why it is more efficient that linear search.

- Recursive Binary Search:

```
int binarySearch (int arr[], int low , int high,
                        int key){
    if (low > high) return -1;
    int mid = (low + high) /2;
    if (arr [mid] == key) return
binarySearch (arr , low, mid-1 , key);
    else return binarySearch (arr , mid+1, high, key);
}
```

- Non-Recursive Binary Search

```
int binarySearch (int arr[] , int n , int key){
    int low = 0 , high = n-1;
    While (low <= high){
        int mid = (low + high ) /2;
        if (arr [mid] == key) return mid;
        elseif (arr [mid]> key) high = mid - 1;
        else low = mid + 1;
    }
return -1; }
```

·Efficiency : Binary Search has $O(\log n)$ time Complexity, while Linear Search has $O(n)$, making Binary Search faster for large Sorted datasets.

2. Solve the following recurrence using Recursion Tree method.

(a) $T(n) = 2T(n/2) + 1$

- Each level does constant ~~[illegible]~~ work (1).
- Number of levels = $\log n$.
- Total work = $O(\log n)$.

(b) $T(n) = 3T(n/4) + cn^2$

- Tree root work = cn^2
- Each level reduces size to $n/4$, so work reduces faster.
- The dominant work comes from the root level: $O(n^3)$.

Section - C

1. Explain various Asympotic Notations that describe the running time of an algorithm.

- Big O (O) : Upper bound (worst-case).
- Omega (Ω) : Lower bound (best-case).
- Theta (Θ) : Tight bound (average-case).
- Little o (o) : Strictly less than a function.
- Little omega (ω) : Strictly greater than a function.

2. Explain master's method and solve the following recurrence using master method.

- Masters method solves recurrence of the form:

$T(n) = aT(n/b) + f(n)$.

(a) $T(n) = 2T(n/4) + \sqrt{n}$

- $a = 2$, $b = 4$, $f(n) = n^{1/2}$.
- compare $n^{\log_b a} = n^{\log_4 2} = n^{0.5}$ and $f(n)$.
- Both are $n^{0.5}$, thus case 2 of master's theorem applies.
- Solution: $T(n) = O(n^{0.5} \log n)$.

(b) $T(n) = 2T(n/2) + n^3$.

- $a = 2$, $b = 2$, $f(n) = n^3$.
- $n^{\log_b a} = n^{\log_2 2} = n$.
- $f(n)$ dominates n, so case 3 applies.
- Solution: $T(n) = \Theta(n^3)$.