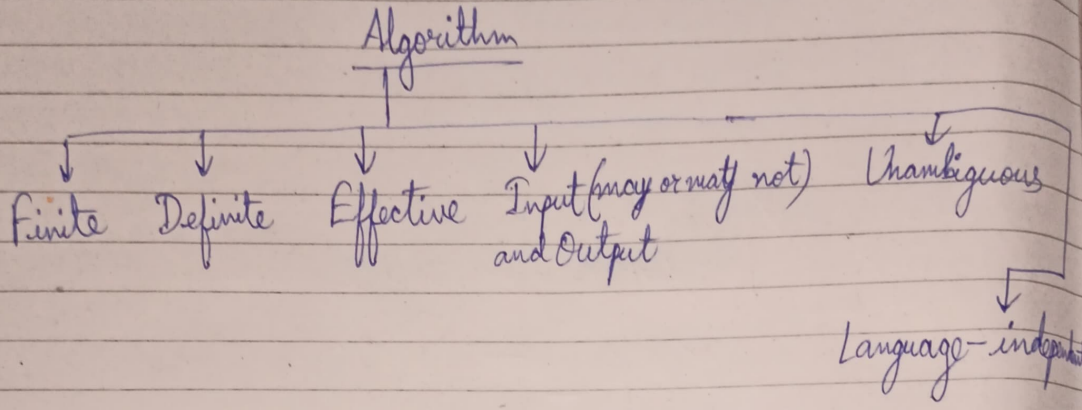
5.3.24

Algorithm

"Algorithm" → Abu Jafar Mohammed ibn Musa al Khowarizmi. (825 A.D.).



a) a priori estimate → Performance (time) measured w.r.t asymptotic notation

b) a posteriori estimate → Time measured as CPU time.

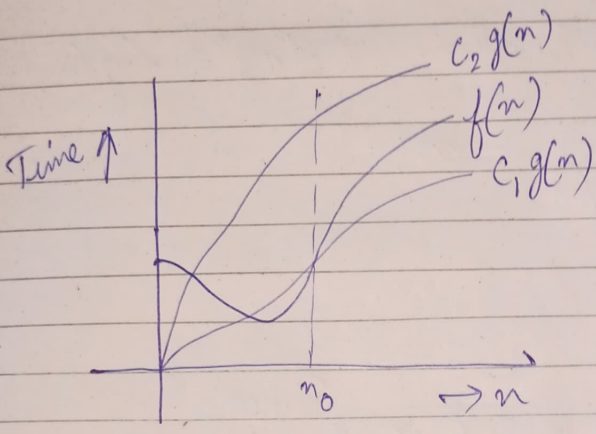
∴ Each program has two parts

- Fixed Part (Independent of input)
- Variable Part (Independent of input)

Books → Horowitz Sahni, ~~Coremen~~ Cormen.

Asymptotic Notation :-

1. $\Theta(g(n)) = \{f(n)$: there exists positive constants c_1, c_2 and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$, for all $n \geq n_0\}$.



$f(n) = \Theta(g(n))$.

$f(n)$ and $g(n)$ are positive functions

2. $O(g(n)) = \{f(n)$: there exists positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$.

3. $\Omega(g(n)) = \{$ "- - - - - - $0 \leq cg(n) \leq f(n)$"$\}$

PT-O

4. $o(g(n)) = \{ f(n) :$ there exists positive constants c and n_0 such that $0 \leq f(n) < cg(n)$ for all $n \geq n_0 \}$.

(small o)

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0.$$

5. $\omega(g(n)) = \{ \text{"} \quad \text{"} \quad \text{"} \quad 0 \leq cg(n) < f(n) \text{"} \}$.

(small omega)

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty.$$

~~Divide and~~

Binary Search, Max Min, Merge Sort, Quick Sort, Strassen's Matrix Multiplication, Tower of Hanoi, Permutation Combination, Factorial, Fibonacci Series.

Functions (Assumption: $f(n)$ and $g(n)$ are non-negative)

1. Transitivity :-

$f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$.

$\Rightarrow f(n) = \Theta(h(n))$.

2. Reflexivity :-

$f(n) = \Theta(f(n))$

$f(n) = O(f(n))$.

$f(n) = \Omega(f(n))$.

3. Symmetry :-

$f(n) = \Theta(g(n))$ iff $g(n) = \Theta(f(n))$.

B

4. Transpose Symmetry :-

$f(n) = O(g(n))$ iff $g(n) = \Omega(f(n))$.

$f(n) = o(g(n))$ iff $g(n) = \omega(f(n))$.

P.T.O

Imp

Trichotomy → One property that holds ~~true~~ for real numbers, but not for asymptotic notations.

Sum :-

① If $f(n) = n^2+2n+5$, then prove that $f(n)$ is $O(n^2)$.

Ans) $f(n) = n^2+2n+5$.

$= n^2\left(1+\frac{2}{n}+\frac{5}{n^2}\right)$.

$\leq n^2(1+2+5)$.

$\therefore f(n) \leq 8n^2$. $c=8$.

for $n=1$, $f(n) = 8$

$8n^2 = 8$. True

For $n=2$, $f(n) = 13$.

$8n^2 = 32$ True

For $n=3$, $f(n) = 20$

$8n^2 = 72$ True.

For $n=4$,

H/W

① $f(n) = 2^n + 3n^2 + 4$. Prove $f(n) = O(2^n)$.

② $f(n) = 3n^2 + 5n + 6$. Prove or disprove that $f(n) = O(n^3)$.

~~$f(n) = n^2\left(3 + \frac{5}{n} + \frac{6}{n^2}\right)$~~.

or, ~~$f(n) < n^2$~~.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{3n^2+5n+6}{n^3}$$

$$= \lim_{n\to\infty} \frac{3}{n} + \frac{5}{n^2} + \frac{6}{n^3}$$

$$= 0 \cdot \phi$$

Hence, $f(n) = \Theta(n^3)$.

[But if $g(n) = n^2$,

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 3$.

So, $f(n)$ cannot be $O(n^2)$].

P.T.O

(3) $f(n) = 4n^3 + 3n + 5$. Is $f(n) = \omega(n^3)$?

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{4n^3+3n+5}{n^3}$$

$$= \lim_{n\to\infty} 4 + \frac{3}{n^2} + \frac{5}{n^3} = 4 \neq \infty.$$

$\therefore f(n) \neq \omega(n^3)$.

[But if $g(n) = n^2$, $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$. So, $f(n) = \omega(g)$

6.3.24

Big Oh Ratio Theorem

Let $f(n)$ and $g(n)$ be such that $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ exists, then a function $f \in O(g)$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty$, also including the case in which limit is 0.

Big Omega Ratio Theorem

Let $f(n)$ and $g(n)$ be such that $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ exists, then a function $f \in \Omega(g)$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$, also including the case in which limit is ∞.

Big Theta Ratio Theorem

Let $f(n)$ and $g(n)$ be such that $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ exists, then a function $f \in \Theta(g)$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$, for some constant c such that $0 < c < \infty$.

Prove that

Q) $f(n) = \log n!$. Prove $f(n) = O(n \log n)$.

Ans) $n! = 1.2.3 \ldots n$

$\leq n.n.n \ldots n = n^n$

or, $n! \leq n^n$.

Taking log on both sides,

$\log n! \leq \log n^n$

or, $\log n! \leq n \log n$.

Hence, $f(n) = O(n \log n)$. Proved

P.T.O

Q) $f(n) = \log n!$. Prove that $f(n) = \Omega(n \log n)$.

Ans) $\log n! = \log 1 + \log 2 + \ldots + \log n$.

$= \log n + \log(n-1) + \ldots + \log 2 + \log 1$.

$\geq \log n + \log(n-1) + \ldots + \log(n/2)$. [Taking half of the terms]

~~$\geq \log \frac{n}{2} + \log\left(\frac{n}{2}-1\right) + \ldots$~~

$\geq \log\left(\frac{n}{2}\right) + \log\left(\frac{n}{2}\right) + \ldots + \log\left(\frac{n}{2}\right)$ [Taking all terms as $\frac{n}{2}$]

$\geq \frac{n}{2} \log\left(\frac{n}{2}\right) = \frac{n}{2}(\log n - \log 2)$

$= \frac{n}{2} \log n - \frac{n}{2}$ [$\because \log_2 2 = 1$]

$\therefore \log n! \geq \frac{n}{2} \log n - \frac{n}{2}$.

Hence, $f(n) = \Omega(n \log n)$. Proved

~~Q) $f(n) = \log n!$~~

Q) $f(n) = \log n!$. Prove that $f(n) = \Theta(n \log n)$.

Ans) First, prove that $f(n) = O(n \log n)$.

Then, prove that $f(n) = \Omega(n \log n)$.

$f(n) = O(n \log n)$, $f(n) = \Omega(n \log n)$

$\Rightarrow f(n) = \Theta(n \log n)$. Proved

Q) If $f(n) = a_m n^m + a_{m-1} n^{m-1} + a_{m-2} n^{m-2} + \ldots + a_1 n + a_0$, then prove that $f(n) = O(n^m)$.

Ans) $|f(n)| \leq |a_m \cdot n^m| + |a_{m-1} \cdot n^{m-1}| + \ldots + |a_1 n| + |a_0|$

[$\because$ Some terms can be negative, $f(n) \leq |f(n)|$.]

$$\text{or, } f(n) \leq n^m \left(|a_m| + \left|\frac{a_{m-1}}{n}\right| + \left|\frac{a_{m-2}}{n^2}\right| + \ldots + \left|\frac{a_1}{n^{m-1}}\right| + \left|\frac{a_0}{n^m}\right| \right)$$

$$\text{or, } f(n) \leq n^m \left(|a_m| + |a_{m-1}| + |a_{m-2}| + \ldots + |a_1| + |a_0| \right)$$

[$n \geq 1$]

$\leq c \cdot n^m$, where c is a constant.

P.T.O

or, $f(n) \leq c \cdot n^m$, for all $n \geq n_0$,

where c is $(|a_m| + |a_{m-1}| + \ldots + |a_1| + |a_0|)$ positive

$\therefore f(n) = O(n^m)$.

Important :-

① $n! = \omega(2^n)$

② $n! = \Theta(n^n)$

③ $n! = \Omega(2^n)$.

④ $\log n = \theta(n)$.

⑤ Show $k \ln k = \theta(n)$ implies $k = \theta\left(\frac{n}{\ln n}\right)$

⑥ $\log(\lceil \log n \rceil !) = \theta(\lceil \log n \rceil \log \lceil \log n \rceil)$

⑦ $\log(\lceil \log \log n \rceil !) = \theta(\lceil \log \log n \rceil \log \lceil \log \log n \rceil)$.

7.3.24

Divide and Conquer

Steps :-

1. Divide → Divide the problem into two sub-problems. (Until problem size is 1)
2. Conquer → Solving the problem.
3. Combine → Combining the subproblems/solutions to get the result.

Merge Sort

Divide → Array of 'n' elements will be given ; it is divided into 2 parts with $n/2$ elements ; parts are further divided till there are only single elements in every part.

Conquer → Two parts are compared and sorted in the required order.

Combine → All sorted parts are combined recursively.

Pseudocode :-

```
MERGE_SORT(A, p, r)
1. if p<r
2. then q ← ⌊(p+r)/2⌋
3.      MERGE_SORT(A, p, q)
4.      MERGE_SORT(A, q+1, r)
5.      MERGE(A, p, q, r)
```

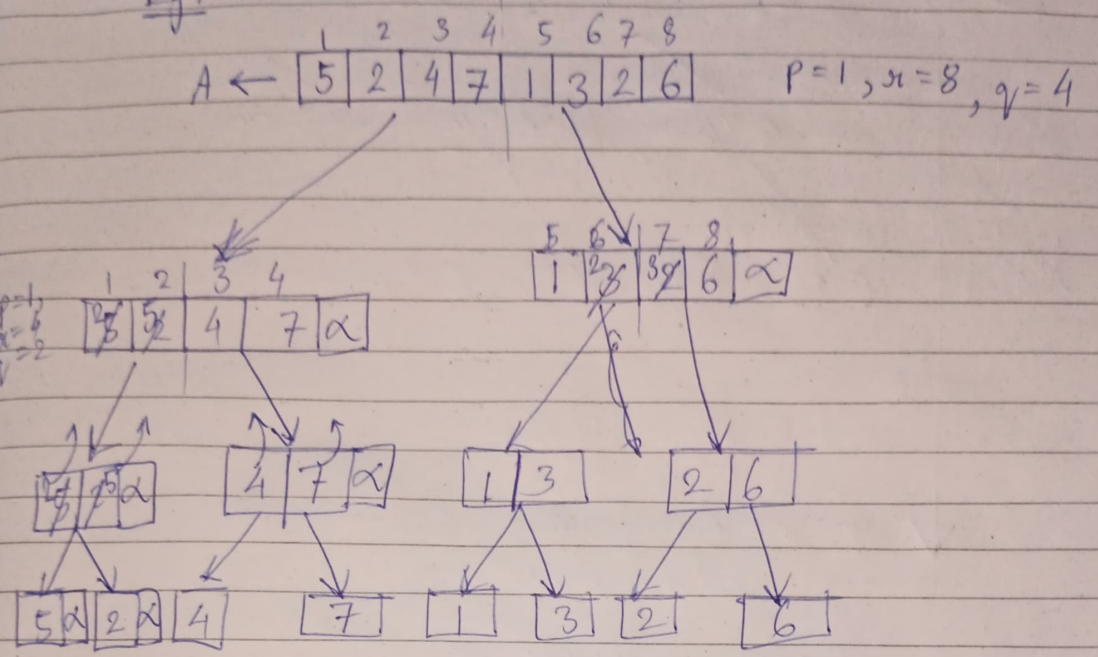
P.T.O

MERGE(A, p, q, r)

1. $n_1 \leftarrow q - p + 1$
2. $n_2 \leftarrow r - q$
3. Create arrays $L[1 \ldots n_1 + 1]$ & $R[1 \ldots n_2 + 1]$
4. for $i \leftarrow 1$ to n_1
5. do $L[i] \leftarrow A[p + i - 1]$
6. for $j \leftarrow 1$ to n_2
7. do $R[j] \leftarrow A[q + j]$
8. $L[n_1 + 1] \leftarrow \infty$
9. $R[n_2 + 1] \leftarrow \infty$
10. $i \leftarrow 1$
11. $j \leftarrow 1$
12. for $k \leftarrow p$ to r
13. do if $L[i] \leq R[j]$
14. then $A[k] \leftarrow L[i]$
15. ~~else~~ $i \leftarrow i + 1$
16. else $A[k] \leftarrow R[j]$
17. $j \leftarrow j + 1$

Sentinel (End-marker of an array) (an element which is not considered, never occurs; by taking modulus in random generation)

Eg :-



∴ Final

↗ Merge Sort is :-

1. ~~Not~~ NOT an in-place sorting algorithm, as it does not operate on a single array.
2. A stable sorting algorithm. ~~as~~

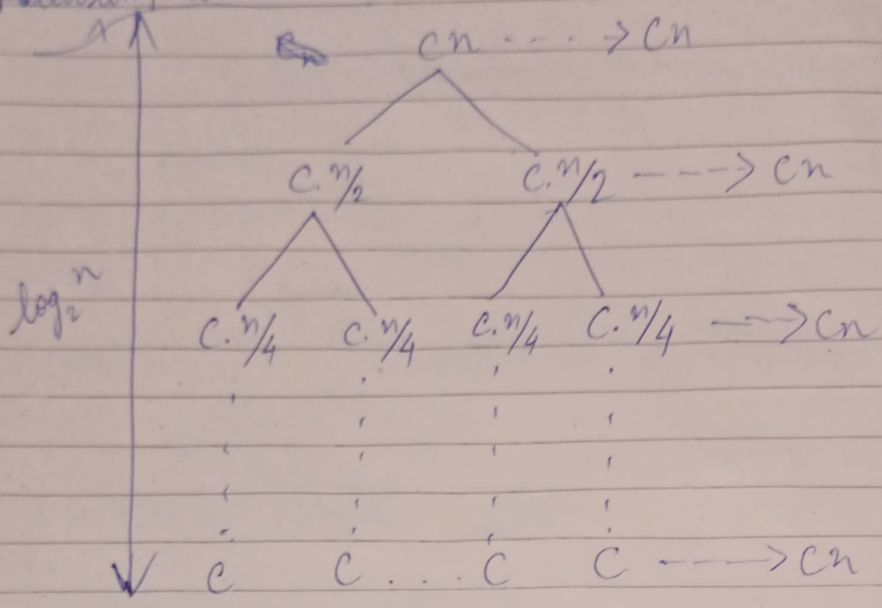
Time Complexity :-

↗ Merge procedure takes linear time, as only a for loop is involved.

P.T.O

Recursion

Recursion Tree Method :-



Time = Cost × Height

$= c.n.\log_2 n$.

$\approx O(n.\log_2 n)$.

Substitution Method :-

$$T(n) = \begin{cases} \theta(1) & ; \text{if } n \leq c \\ a \cdot T(n/b) + D(n) + C(n) & ; \text{otherwise} \end{cases}$$

- a : No. of subproblems
- n/b : Size of problem
- D(n) : Divide (Constant)
- C(n) : Combine (Linear)

or, $$T(n) = \begin{cases} c & ; \text{if } n=1 \\ 2.T(n/2) + cn & ; \text{otherwise} \end{cases}$$

Since, the array is divided into 2 parts (each part has $T(n/2)$)

$$T(n) = 2T(n/2) + cn$$

$$= 2\left(2T(n/2^2) + c \cdot n/2\right) + cn$$

$$= 2^2 T(n/2^2) + 2 \cdot cn$$

$$\vdots$$

$$= 2^k T(n/2^k) + k \cdot cn$$

At k^{th} state, $n/2^k = 1$. $\left[T(n/2^k) = T(1)\right]$

$$\Rightarrow n = 2^k$$

$$\Rightarrow \log_2 n = k.$$

$$\therefore T(n) = n \cdot c + \log_2 n \cdot cn$$

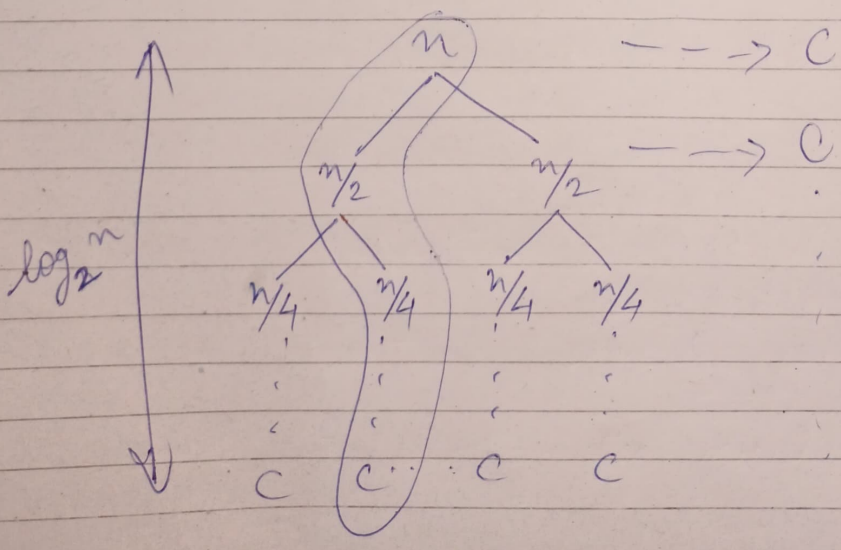
$$\approx O(n \log_2 n)$$

P.T.O

Binary Search

```
binsearch (arr, x, low, high)
    if low > high
        return False
    else
        mid = ⌊(low + high)/2⌋
        if x == arr[mid]
            return mid
        else if x > arr[mid]
            return binsearch (arr, x, mid+1, high)
        else
            return binsearch (arr, x, low, mid-1)
```

Recursion Tree Method :-



→ In the Divide part (state), the problem is divided into subproblems ($\log_2 n$ state). In Conquer state, we are matching with the key value (Const. time). No Combine state.

Substitution Method :-

$$T(n) = T(n/2) + C$$

$$= T(n/2^2) + 2C$$

$$\vdots$$

$$= T(n/2^k) + k \cdot C .$$

$n/2^k = 1 \quad [T(n/2^k) = T(1)]$.

or, $\log_2 n = k$.

$\therefore T(n) = C + \log_2 n \cdot C$.

$\approx O(\log_2 n)$.

PT·O

Probable Questions :-

1. Define Algorithm. ~~Write~~
2. Write the properties of algorithm.
3. What do you mean by the term 'asymptotic'?
4. Short notes :-
   a) Asymptotic Notation
   b) Properties of " "
   c) ~~Big~~ $O, \Omega, \Theta, o, \omega$
5. Define Ratio Theorem for O, Ω, Θ.
6. ~~All numerical problems on complexity and proofs.~~
6. Numericals and proofs on asymptotic notations and recurrence relations.
7. Short notes :-
   a) Recurrence Relations
   b) Substitution Method
   c) Recurrence/Recursion Tree Method
   d) Master Theorem.
8. Proof of Master Theorem
9. For all algorithms, practice pseudocode/algorithm, example with step-by-step execution of the

algorithm.

10. Time Complexity and Space Complexity Analysis;
Best Case, Worst Case and Average Case Scenarios.

11. Why is Best Case time complexity of Insertion Sort linear?

12. For all sorting and searching algorithms, calculate no. of movements (swapping for sorting) and no. of comparisons.

13. Why Quick Sort is called the best sorting algorithm.

14. Name some sorting algorithms which are not based on comparisons.

15. Why ~~can is the Wor~~ comparison-based algorithms having time complexity $O(n \log n)$? (Coremen)

16. Explain the time taken by Merge Sort in each of the steps — Divide, Conquer and Combine.
(Same question for all types of algorithms based on Divide and Conquer).

17. Short Note on Divide and Conquer techniques.

P.T.O

Maxmin Algorithm

- An array with n elements in random order → Input.
- Finding Max. and Min. elements using Divide and Conquer approach → Objective.

Divide → Divide the array into 2 parts.

~~(Stops when~~

- ~~Given n~~ Stop when subproblem size is 2; one will be max. and the other will be min.
- Compare max. and min. at each stage; set them.

Conquer → Comparing.

Combine → Combining the subarrays to calculate max. and min.

A[x...y].

Algorithm:-

MaxMin(x, y)

if $y - x \leq 1$ then

return (max(A[x], A[y]), min(A[x], A[y]))

else

(max1, min1) = MaxMin(x, $\lfloor (x+y)/2 \rfloor$)

(max2, min2) = MaxMin ($\lfloor (x+y)/2 \rfloor + 1$, y).

return (max(max1, max2), min(min1, min2))

$T(n) = 2T(n/2) + 2$; for $n > 2$ → Since there are 2 comparisons → for max. and for min.

$= 1$; for $n = 2$.

$= 0$; for $n = 1$. (Max. = Min.).

Using Substitution Method :-

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

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

$= 2^2 \cdot T(n/2^2) + 2^1 + 2^2$

$\vdots$

$= 2^k \cdot T(n/2^k) + (2^1 + 2^2 + \ldots + 2^k)$

P.T.O

$$= 2^k \cdot T\left(\frac{n}{2^k}\right) + \frac{2 \cdot (2^k - 1)}{2 - 1}$$

$$= 2^k \cdot T\left(\frac{n}{2^k}\right) + 2^{k+1} - 2.$$

$$\frac{n}{2^k} = 2$$

$$\Rightarrow n = 2^{k+1}$$

$$\therefore T(n) = 2^k + 2^{k+1} - 2.$$

$$= \frac{n}{2} + n - 2.$$

$$\approx O(n).$$

No. of Comparisons in Maxmin Algorithm :-

8.3.24 DAA Lab Work - 1

a) Implement bubble sort, selection sort, insertion sort for a random set of data ($10, 10^2, 10^3, 10^4, 10^5, 10^6$ — input). Measure CPU-time. Plot a graph for i/p VS CPU-time for each of the above mentioned sorting algorithms.

Lab Copy :-

1. Problem Statement.
2. Algorithm.
3. Program Code.
4. I/P & O/P Plot.
5. Analysis of time and space complexity.
6. Discussion (PO mapping).

P.T.O

Bubble Sort

10 → CPU Time = 0.000004 sec.
100 → " = 0.000101
1000 → " = 0.005934
10000 → " = 0.200599
100000 → " = 22.293266
1000000 → " = 2295.008238.

Selection Sort

10 → CPU time = 0.000005 sec.
100 → " = 0.000067
1000 → " = 0.004559
10000 → " = 0.096155.
100000 → " = 8.234350
1000000 → " = ~~529.307562~~ ~~996.497510~~
826.847238

Insertion Sort

10 → CPU time = 0.000004 sec.
100 → " = 0.000033
1000 → " = 0.002742
10000 → " = 0.063410
100000 → " = 5.054821
1000000 → " = ~~529.307562~~ 506.714044