Characteristics of Algorithm.

Algorithm :- An Algo. is a finite set of instructions that if followed accomplishes the task.

Criteria:-

① Input :- ~~one/many~~ 0/many.

② Output :- one/many.

③ Finiteness :- It should Terminate.

④ Definiteness :- Eng. but not code. // Eg. Add 5 or 6 to x ∴ Faulty!
unambigious 5/0

⑤ Effectiveness :- Should be on pen/pencil & paper.

Steps to Write a Good Algorithm:

① Indentation includes Block Structures.

② The looping constructs have interpretations similar to C & Java.

↳ Eg. for(i=0; i<n; i++) ⟹ for i=0 to n. step size 2.

③ // symbol indicates comments.

④ Multiple Assignments

i=j=e assigns to both i, j the value of expression e.

Variables such as i, j are local to a given procedure. We should not use global variables without explicit statements.

& global i;

Accessing Specific elements can be done in the same way as we do it traditionally.

For ~~anny~~ any attributes no bracketts are required.

We organise compound data into objects which are composed of attributes.

We Pass ~~attrit~~ parameter ~~by a prodi produ~~ through a procedure by value.

A return statement immidiately transfers control back to the point of call in the calling procedure.

The keyword error indicates that an error occured because the conditions were wrong for the procedure to be called.

Algorithm SumofArray (A, n)

// Finding sum of elements in the array

		# no. of times	cost of each inst^n	Total
1	sum = 0	1	$1 \cdot c_1$	1
2	for i = 1 to i = n	n+1	$1 \cdot c_2$	n+1
3	sum = sum + A[i]	n	$1 \cdot c_3$	n
4	print (sum)	1	$1 \cdot c_4$	1
				2n + 3

O(n) //

$c_1 \cdot 1 + c_2(n+1) + c_3(n) + c_4 \cdot 1$

$c_1 + c_2 n + c_2 + c_3 n + c_4$

$(c_2 + c_3) n + (c_1 + c_2 + c_4)$

$a \quad n \quad + \quad b$

$an + b$

$O(n)$ //.

Algorithm :- Sum of Matrices (A, B, C)

```
1  for i = 1 to m ..... A m×n  B m×n ..... m+1
2      for j = 1 to n ..... m(n+1)
3          c[i][j] = A[i][j] + B[i][j] ..... m.n
4          print (C[i][j]); ..... m.n
5      print (new line) ..... m
```

$$m+1 + m(n+1) + mn + mn + m$$

$$3mn + 3m + 1$$

$$= O(mn)$$

Write an algorithm for selection sort & derive its time complexity

Worst Case Scenario.

(Best & Worst Case).

Algorithm selectionSort (A, n)

		# times.	cost
1	for i = 1 to n-1	n.	c_1
2	imin = i	n-1	c_2
3	for j = i+1 to n	$\sum_{i=2}^{n} i = \frac{n^2}{2} + \frac{n}{2} - 1$	c_3
4	If (A[imin] > A[j])	$\sum_{i=1}^{n-1} j$	c_4
5	imin = j	$\sum_{i=1}^{n-1} i$ } $\frac{n^2}{2} - \frac{n}{2}$	c_5
6	temp = A[imin]	n-1	c_6
7	A[imin] = ~~temp~~ A[i]	n-1	c_7
8	A[i] = temp	n-1	c_8

$$n c_1 + c_2(n-1) + c_3\left(\frac{n^2}{2} + \frac{n}{2} - 1\right) + c_4\left(\frac{n^2}{2} - \frac{n}{2}\right) + c_5\left(\frac{n^2}{2} - \frac{n}{2}\right) + c_6(n-1) + c_7(n-1) + c_8(n-1).$$

$\Longrightarrow$ $an^2 + ~~bn + c~~$

$O(n^2)$.

In Best Case ignore c_5

As we assume the ~~array~~ imin is always smaller than j ∴ imin = j never ~~to~~ occurs.

Worst case → for → while loop consider & shifting + termination.

for CS & CG only consider shifting.

rithm Insertion Sort (A, n)

	# times	cost	
For j = 2 to n	n	c_1	$c_1 n$
key = A[j]	n-1	c_2	$c_2(n-1)$
i = j-1	n-1	c_3	$c_3(n-1)$
while (i >= 1 and A[i] > key) ..	$\sum_{i=2}^{n} t_i$	c_4	$c_4(n-1)$
{ A[i+1] = A[i]	$\sum_{i=2}^{n} (t_i - 1)$	c_5	$c_5\left(\frac{n(n+1)}{2} - 1\right)$
i--	$\sum_{i=2}^{n} (t_i - 1)$	c_6	$c_6\left(\frac{n^2+n}{2} - 1\right)$
A[i+1] = key.	n-1	c_7	$c_7(n-1)$

while (i >= 1 and A[i] > key) .

A[i+1] = A[i] .

$\sum_{i=2}^{n} i$ $\sum_{i=1}^{n-1} i$

$\frac{n(n+1)}{2} - 1$ $\frac{n(n-1)}{2}$

$c_1 n + c_2$ $c_4\left(\frac{n^2}{2} + \frac{n}{2} - 1\right)$ $c_5\left(\frac{n^2}{2} - \frac{n}{2}\right)$ $c_6\left(\frac{n^2}{2} - \frac{n}{2}\right)$

$$c_1 n + c_2 n - c_2 + c_3 n - c_3 + c_4 \frac{n^2}{2} + c_4 \frac{n}{2} - c_4 + c_5 \frac{n^2}{2} - c_5 \frac{n}{2} + c_6 \frac{n^2}{2} - c_6 \frac{n}{2} + c_7 n - c_7 .$$

$$\left(\frac{c_4 + c_5 + c_6}{2}\right) n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} + \frac{c_5}{2} - \frac{c_6}{2} + c_7\right) n - (c_2 + c_3 + c_4 + c_7)$$

$an^2 + bn - c$

$O(n^2)$ //.

For Best Case :- (Only consider the termination of the while loop).

$C_4 = n-1$

C_5 & C_6 = 0.

$$C_1 n + C_2(n-1) + C_3(n-1) + C_4(n-1) + C_5 \cdot 0 + C_6 \cdot 0 + C_7(n-1)$$

$$(C_1 + C_2 + C_3 + C_4 + C_7)\, n - (C_2 + C_3 + C_4 + C_7).$$

Average Case :-

The while loop will get executed on an average of n^2 times

$\therefore$ The Average case Time Complexity will be $O(n^2)$.

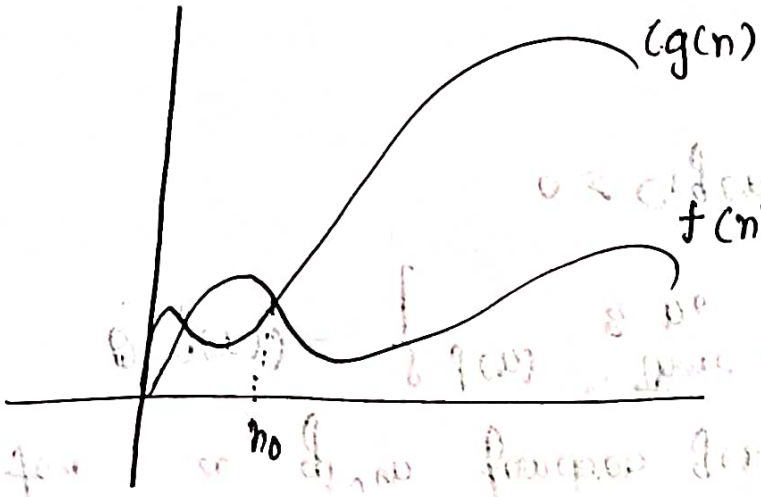
Asymptotic Notation:

O- notation (Big Oh)

For a given function $g(n)$ we denote by $O(g(n))$

$O(g(n)) = \{ f(n)$: There exist possitive constants C & n_0 such that

$0 \leq f(n) \leq g(n)$ for all $n \geq n_0$.

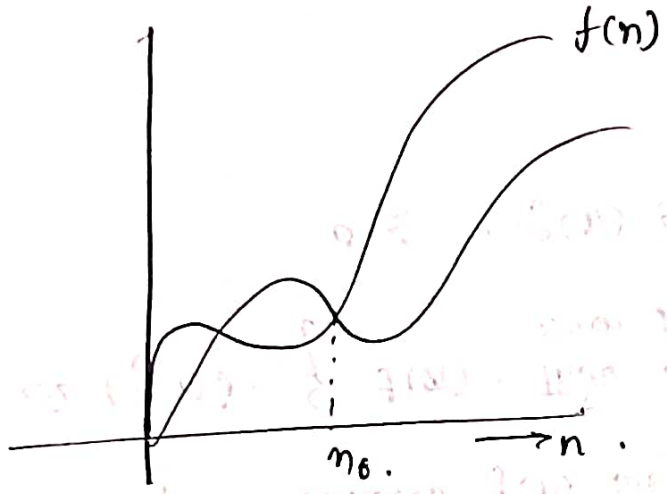


Ω (omega)

For a given function $g(n)$ we denote by $\Omega(g(n))$

$\Omega(g(n)) = \{ f(n)$: There exist possitive constants C & n_0 such that

$0 \leq c \cdot g(n) \leq f(n)$ for all $n \geq n_0$

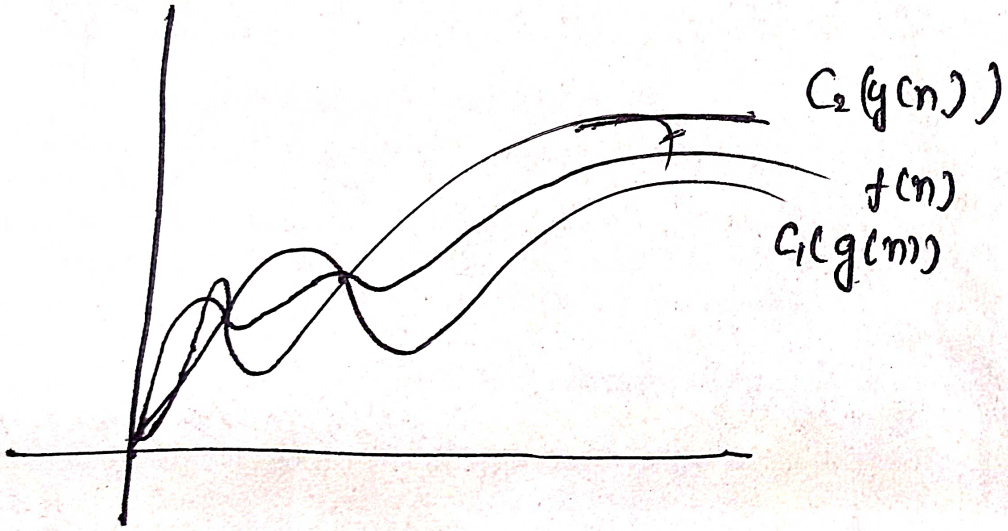


θ Notation (Theta).

For a given function g(n) we denote by θ(g(n))

$\theta(g(n)) = \{ f(n)$: There exist possitive constants C & n_0. such that

$0 \leq C_1 g(n) \leq f(n) \leq C_2 g(n)$ for all $n \geq n_0$.


$C_2(g(n))$
$f(n)$
$C_1(g(n))$

```
BinarySearch (A, low, high, x)
if (low == high)
        if A[low] == x
            return low.
        else
            return false. (-1)                    } (1)

else
    mid = (low + high)/2
    if (A[mid] >= x)
        return mid
    else if (A[mid] > x)
        BinarySearch (A, low, mid-1, X)
    else
        BinarySearch (A, mid+1, high, X)
```

Recurrence Relation.

(1) Masters Theorem.

(2) Recursion Tree.

(3) Substitution Method.

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

- a ↑ no. of subproblem.
- b ↑ ratio of original problem to subproblem.
- $f(n)$ → solution (no. of comparisons made for the terminating condition) i.e (1).

Reccurence relation of Binary Search.

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

Master's Theorem:-

Let $a \geq 1$ & $b > 1$ be constants; let $f(n)$ be a function & let $T(n)$ be defined on the non-negative integers by the reccurence.

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

① if $f(n) = O(n^{\log_b a - \epsilon})$; $\epsilon > 0$.

then $T(n) = \Theta(n^{\log_b a})$

② if $f(n) = \Theta(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \log n)$

③ if $f(n) = \Omega(n^{\log_b a + \epsilon})$; $\epsilon > 0$ & $a < b$.

if $a f\left(\frac{n}{b}\right) \leq c \cdot f(n)$ for some constant $0 < c < 1$

then. $T(n) = \Theta(f(n))$

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

$$a = 1 \qquad b = 2.$$

$n^{\log_b^a} = n^{\log_2^1} = n^0 = 1.$

$f(n) = 1$

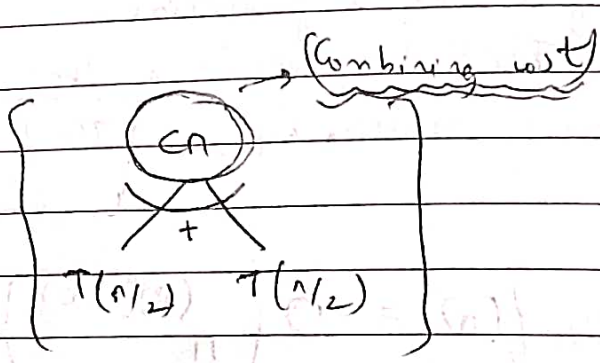
The recurrence relation falls in the the case II.

$$\therefore T(n) = \Theta\left(n^{\log_b^a} \log n\right)$$

$$\boxed{T(n) = \Theta(\log n).}$$

Recursion Tree :-

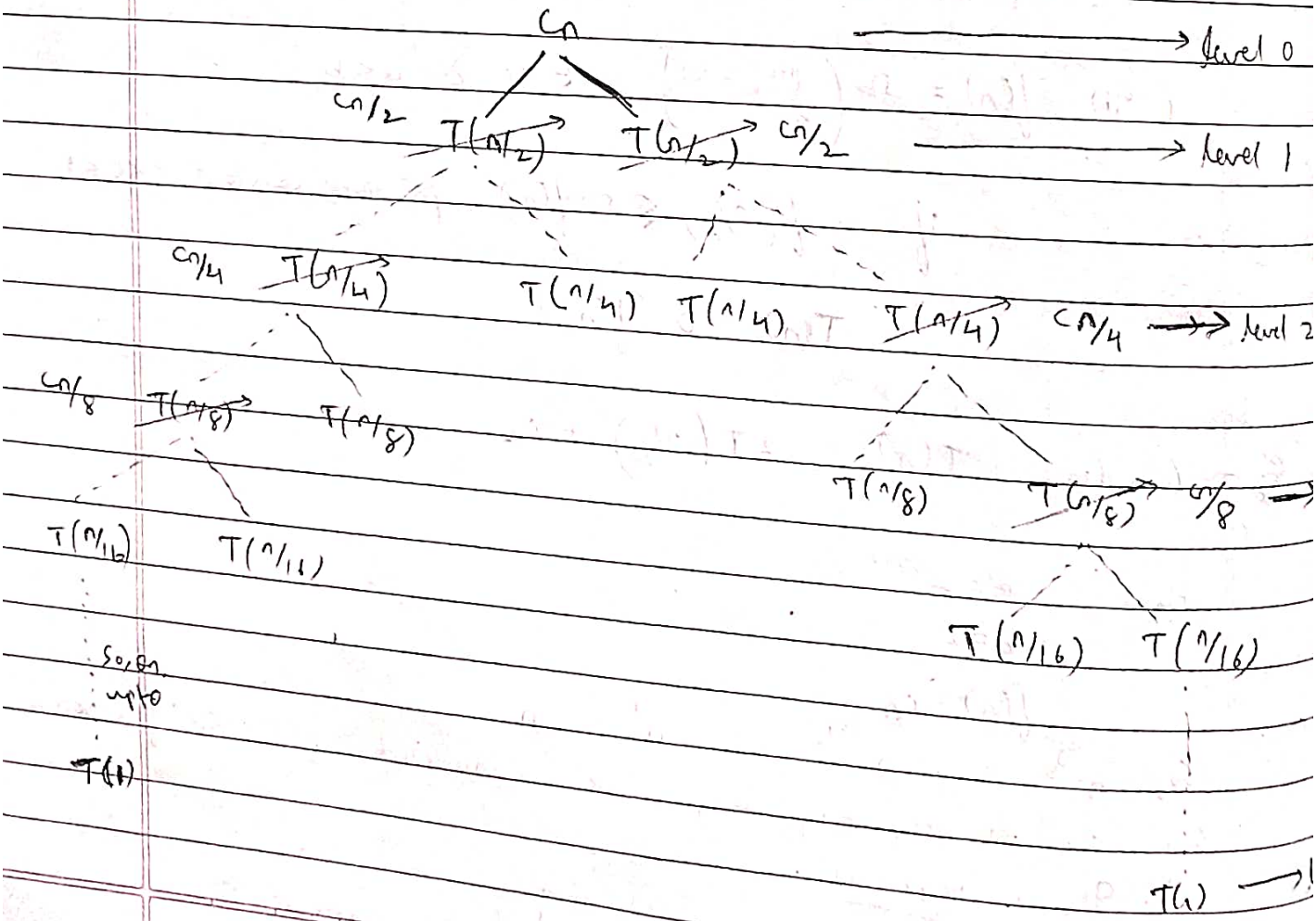
Basically, $2T(n/2)$ means we are "DIVIDING" the problem into "2" sub-problems where each is "half" of the original problem.

So, $T(n) =$ 

To get $T(n/2)$, substitute "$n/2$" in $T(n) = 2T(n/2) + cn$

So, $T(n/2) = 2T(n/4) + cn/2$

So,



Level	No. of terms at each level	cost at each level
0	1	cn
1	2	cn
2	4	cn
3	8	cn
4	16	cn
⋮		
x	2^x	$2^x \cdot T(1)$

Total cost $= x \cdot cn + 2^x \cdot T(1)$

From observation,

$T\left(\frac{n}{2^x}\right) = T(1) \Rightarrow \therefore \frac{n}{2^x} = 1 \Rightarrow n = 2^x$

$\therefore x = \frac{\log n}{\log 2}$

$\therefore \boxed{x = \log_2 n}$

$\therefore$ Total cost $= \log_2 n \cdot cn + 2^{\log_2 n} \cdot T(1)$

$= \log_2 n \cdot cn + n \cdot T(1)$

$= n \cdot c \cdot \log_2 n + n \cdot T(1)$

$= \boxed{O(n \cdot \log_2 n)}$

$\frac{cn}{2^n}$ | $T(1)$

2^n → cost $c_n \cdot T(1)$

2^{n-1} → $\frac{cn}{2} \cdot T(1)$

$c_n + 2^{cn}\left[1+2+3+4+\ldots 2^{n-1}\right]$

4^{n-1}

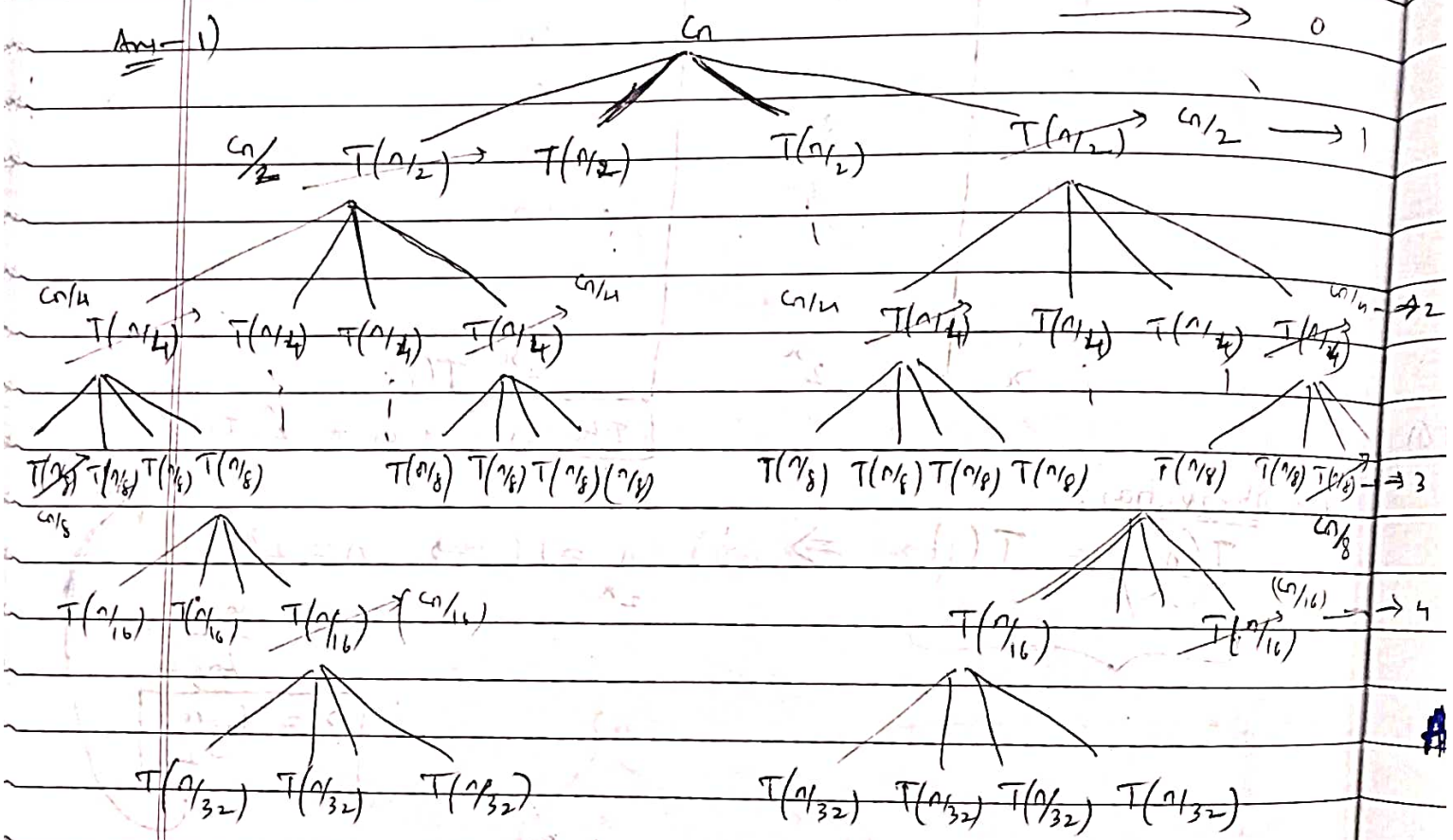
4^{n-1} 2^{n-1} 2^{n-1} cn

Q1) $T(n) = 4T(n/2) + cn$

Q2) $T(n) = 2T(n/2) + cn^2$

Q3) $T(n) = 3T(n/3) + cn^2$

Ans – 1)



Level	No. of terms at each level	cost
0	1	cn
1	4	$2cn$
2	16	$4cn$
3	64	$8cn$
4	256	$16cn$
⋮	⋮	⋮
x	4^x	$4^n T(1)$

$\sum_{i=0}^{x-1} 2^i cn$

Total cost $= (n^2 - 3n + 3)cn + 4^n \cdot T(1)$

$T\left(\frac{n}{2^x}\right) = T(1) \Rightarrow \frac{n}{2^x} = 1 \Rightarrow n = 2^x$

$x = \frac{\log n}{\log 2} = \boxed{\log_2 n}$

$c_n + 2c_n[1+2+3$

$c_n + 2c_n\frac{(n-2)(n-1)}{2}$

$c_n[1 + n^2 - 3n + 2]$

$\frac{a(1-r^n)}{(1-r)}$ $\quad$ $\frac{1(2^n-1)}{1}$

$= 2^n - 1$

$(n^2 - 3n + 3)c_n$

So, Total cost $= 4^n \cdot T(1) + \sum_{i=0}^{n-1} 2^i \cdot c_n$

$= 4^{\log_2 n} \cdot T(1) + c_n(2^n - 1)$

$= n^2 \cdot T(1) + c_n(2^{\log_2 n} - 1)$

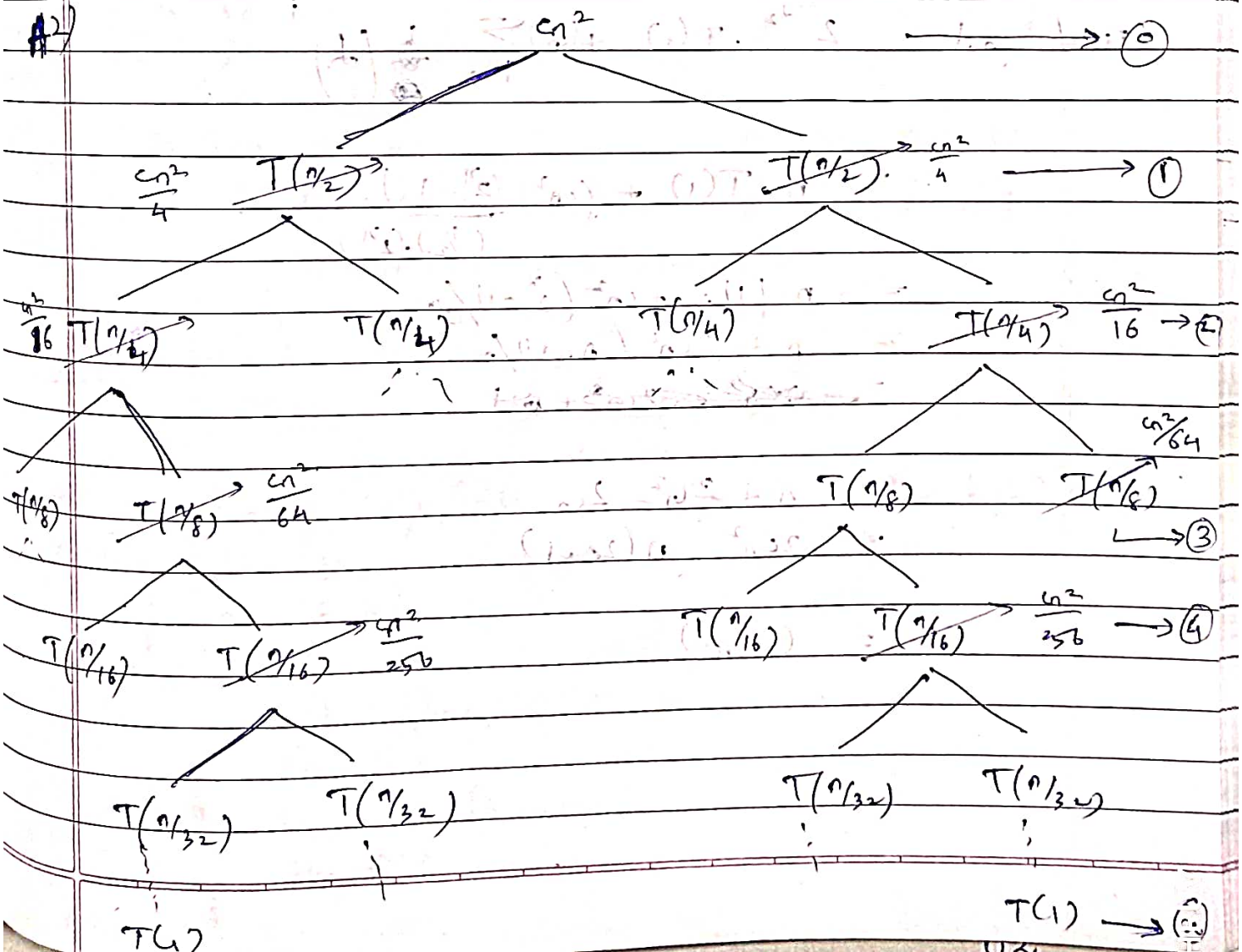
$= n^2 \cdot T(1) + c_n(n-1)$

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

$= n^2 \cdot (1) + cn^2 - cn$

$= (c+1)n^2 - cn$

$= O(n^2)$



Level	No. of terms at each level	cost
0	1	cn^2
1	2	$cn^2/2$
2	4	$cn^2/4$
3	8	$cn^2/8$
4	16	$cn^2/16$
:		
:		
:		
x	2^x	$2^x \cdot T(1)$

$$\text{Total} = \sum_{i=0}^{x-1} \frac{cn^2}{2^i} + 2^x \cdot T(1)$$

$$T\left(\frac{n}{2^x}\right) = T(1) \implies \frac{n}{2^x} = 1 \implies \boxed{x = \log_2 n}$$

$$\text{Total cost} = 2^{\log_2 n} \cdot T(1) + (cn^2) \sum_{i=0}^{x-1} \left(\frac{1}{2}\right)^i$$

$$= n \cdot T(1) + (cn^2)\frac{(2^x - 1)}{(1/2)\cdot(2^x)}$$

$$= n \cdot (1) + 2cn^2 (2^x - 1)/(2^x)$$

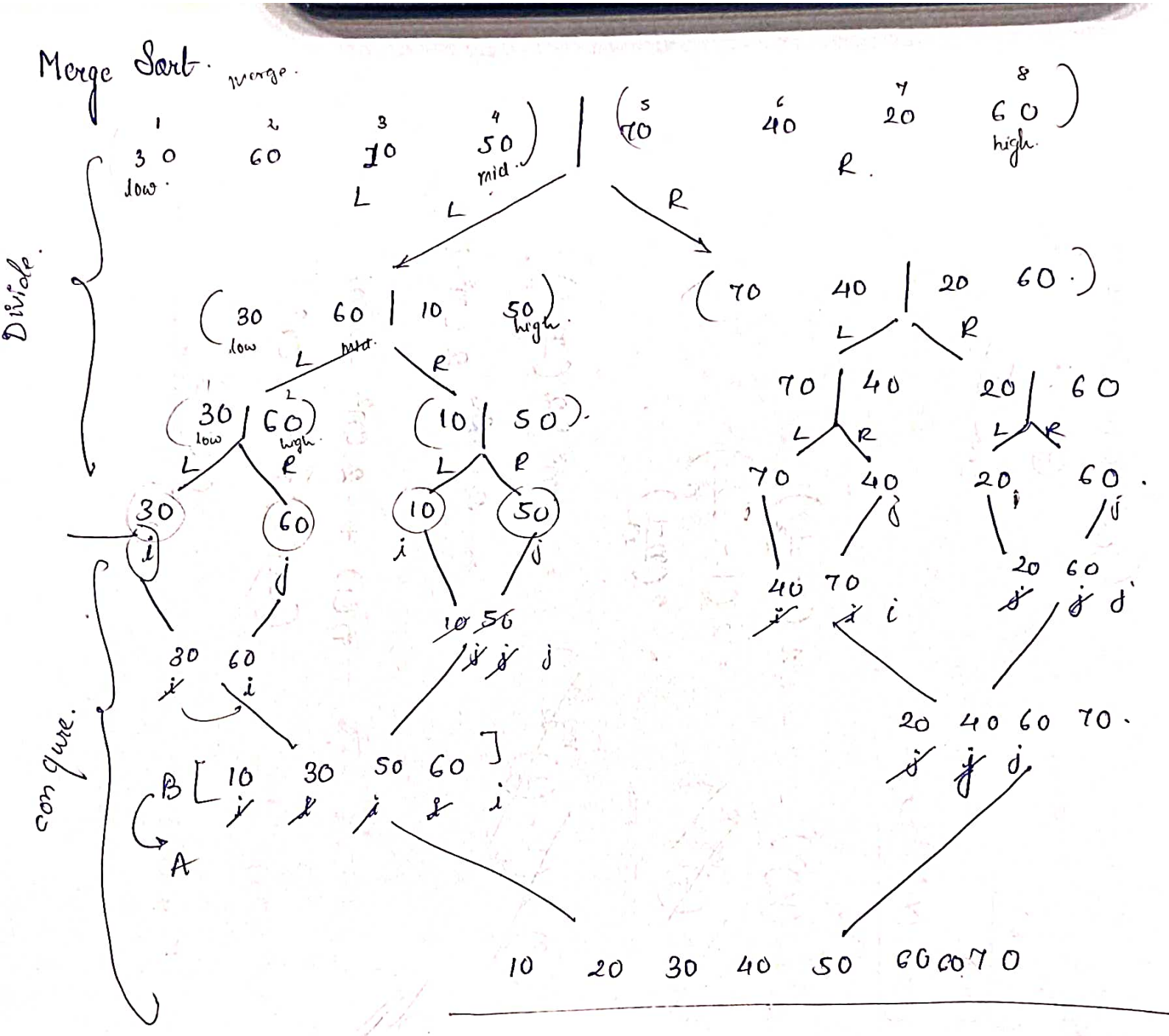
$$= n + 2cn^2 (n-1)/n$$

$$= n + 2cn^2 - 2cn$$

$$= 2cn^2 - n(2c-1)$$

$$= O(n^2)$$

Merge Sort.


merge.
1 2 3 4 5 6 7 8
30 60 10 50 70 40 20 60
low.
mid.
high.
L
R
Divide.
conquer.
(30 60 | 10 50)
(70 40 | 20 60)
30 | 60
10 | 50
70 | 40
20 | 60
40 70
20 60
B [10 30 50 60]
A
10 20 30 40 50 60 60 70

```
MergeSort (A, low, high)
{
    if (low < high)
    {
        mid = (low + high)/2 ..... // Left
        MergeSort (A, low, mid) ..... // Right   // Left
        MergeSort (A, mid+1, high) ..... // Right
        Merge (A, mid+1, high) ..... combine.
        Merge (A, low, mid, high)
    }
}

Merge (A, low, mid, high)
{
    i = low
    j = mid+1
    k = low .  // temporary array
                                        // B is a temporary array.
    while (i ≤ mid && j ≤ high)
    {
        if A[i] ≤ A[j]
            B[k] = A[i]
            i++
        else
            B[k] = A[j].
            j++
        k++ }
    if (i > mid)
        if (j ≤ high)
            while (j ≤ high)
                B[k] = A[j]
                j++
                k++
```

if j > high.

~~if i > high if i~~

while i ≤ mid

B[k] = A[i]

i++

k++

for i = low to high.

A[i] = B[i]

}

Merge Sort time Complexity.

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

$a = 2 \quad b = 2 \quad f(n) = Cn$.

$n^{\log_2^2} = \underline{n}$

$f(n) = Cn$.

$\therefore T(n) = O(n \log n)$ .... case II

1	2	3	4	5	6	7	8
70	10	50	80	20	60	30	40.

pivot

20 10 30 < 40 < 70 50 80 60.

3 ≅ 4 4.

$T(n) = 2T(\frac{n}{2}) +$ ◯ ← Partition.

```
Quick Sort : (A, low, high)
    if  low < hight
            p = partition (A, low, high)
            Quicksort (A, low, p-1)
            Quicksort (A, p+1, high)
```

```
Partition (A, low, high)
    pivot = A[high]
    i = low
    j = low
    while (i < high)
    {       if A[i] < pivot
                swap (A[i], A[j])
                j++
        i++
    }
    swap (A[j], A[high])
```

```
swap (int a, int b)
{
    int temp;
    temp = a
    a = b
    b = temp.
}
```

n.

∴ Best Case Time Recurrence Relation. (Average Case).

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

~~$O(n/2)$~~ $O(n \log n)$.

For worst case:- ⟶ The pivot element is at the end. (correctly placed)

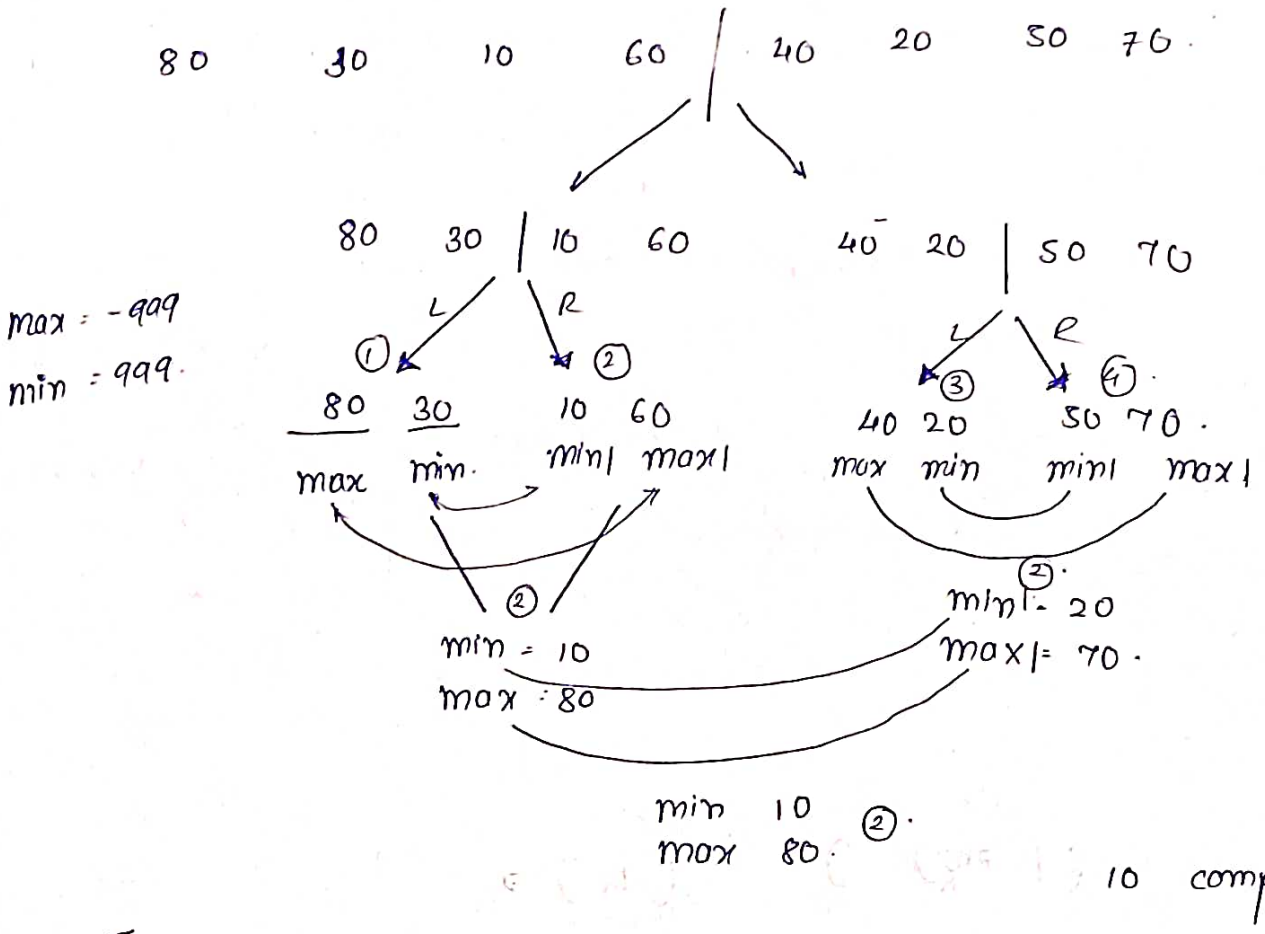
$$T(n) = T(n-1) + cn.$$

Eg 10 20 30 40 50 | 60.

$$T(6) = T(5) + cn$$

$O(n^2)$.

OR
Array might be sorted in Descending order.



10 comparisons.

Time Complexity O(n).

Algorithm Maxmin (A, i, j, max, min)

```
if i == j
    max = min = A[i]          // For only one element.

else if i = j - 1
        if A[i] < A[j]
            min = A[i]        // For two elements.
            max = A[j]
        else
            max = A[i]
            min = A[j].
```

0 .

else

mid = (i+j)/2

Maxmin(A, i, mid, max, min.)

Maxmin(A, ~~mid~~ mid+1, j, max1, min1)

if max1 > max.

max = max1

if min1 < min.

min = min1

} Ambigious in Practical Code.

Time Complexity

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

$n^{\log_2 2} = n = n$

$f(n) = 2n^0$

$\therefore \quad \theta(n)$ // (Case 1).