DSA-II - L - 1 - 23 sep

Before MID

Time complexity

Devide and conquer. (DnC)

Greedy

DP. → (Maximization/minimization) Optimization

After MID:

Graph → Traversal
→ Sorting
→ Bipartite
→ MST
→ SSSP

Hashing

String

NFC

Data Structure & Algorithm

Linear

Non-Linear

Algorithm → set of steps to figure out results from given inputs.

① 1st element define

② Last ,,

③ every element except 1st one must have a previous element

④ every element ex. last one must have a next element.

Algorithm analysis steps

1st → Time complexity *

→ space/memory (C.A. O.S.) complexity.

→ correctness *
  ↳ Halt
  ↳ correct output.

→ Readability

→ Optimality

Readability (Documentation) → S.E.

Exm 1: n = 100.

```
1. int sumN (int n){          → 1
2.     int sum = 0            → 1
3.     for(int i = 1; i <= n; i++){   → n+1 = 101
4.         sum += 1;          → n = 100
5.     }
6.     return sum;            → 1
7. }
```

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

$= 2n + 3 = O(n)$

~~O → assimptotic notation~~

~~Ω → Best~~

Assimptotic notation

- O → worst case
- Ω → Best case.
- θ → average case.

$T(n) = 5n^4 + 2n + 3$

$n=1 \Rightarrow 5 + 5 = 10$

$n=2 \Rightarrow 80 + 7 = 87$

$n=5 \Rightarrow 3125 + 13 = 3138$

$n=10 \Rightarrow 50000 + 23 = 50023$

$n=100 \Rightarrow 500000000 + 203 = 500000203$

$\therefore T(n) = O(n^4)$.

$1 < \log_2(n) < \sqrt{n} < n < n\log_2 n < n\sqrt{n}$

$< n^2 < \cdots < 2^n < n!$

```
int search(int arr[]; int n; int key){
    for(int i=0; i<n; i++){      → n+1
        if (arr[i] == key){      → n
            return i;            → 1/0
        }
    }
    return -1;                   → 0/1
}
```

2	5	9	7	8	3	1

Best case: O(1)

worst case: O(n)

Best

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

$= 2n+2$

$= O(n)$

worst

$T(n) = 1+1+1 = 3$

$= \Omega(c)$

$= \Omega(1)$

DSA II - L-2 - 26 sep

Exm: 3

```
1 int function(int n){
2   int wordSum = 0;          → 1
3   for(int i=0; i<=n; i++){  → n+1
4       wordSum += i;         → n
5   }
6.  if(wordSum%2 == 0){       → 1
7.      return wordSum;       → 1
8.  }
9.  for(int i=1; i<=n; i++){  → n+1
10      wordSum *= i;         → n
11. }
12. return wordSum;           → 1
13. }
```

worst case:

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

$= 4n + 5$

$= O(n)$

Best case:

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

$= 2n + 4$

$= \cancel{O(n)}\ \Omega(n)$

~~Ω~~ Uper bound

Tight "

Lower "

$T(n) = 2n^5 + 3n^2 + 9n + 2$

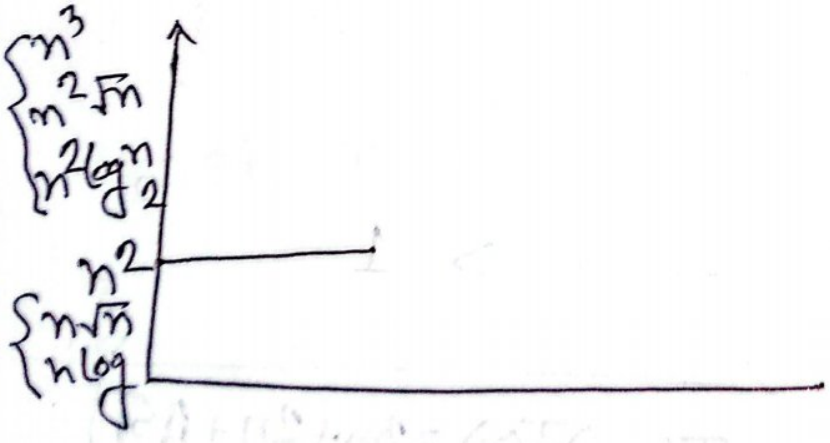
$= 2n^5 + 3n^2 + 9n + 2 \leq (2+3+9+2)n^5$

$\leq 16n^5$

for all n, $2n^5 + 3n^2 + 9n + 2 \leq 16n^5$

$\Rightarrow O(n^5)$

$T(n) = 2n^2 + 3n + 5 = O(n^2)$



average case close to worst case.

$LB \leq TB \leq UB$

Exm 4:

		ω	B
1.	int function(int n ; int m){	→ 1	1
2.	int sum = 0;		
3.	for(int i = 1 ; i <= n, i++){	→ n+1	n+1
4.	sum + = i;	→ n	n
5.	}		
6.	if (sum % 2)	→ 1	1
7.	return sum;	→ 0	1
8.	else{	→ 1	0
9.	for(int i = 1 ; i <= m ; i++){	→ m+1	0
10.	sum + = i;	→ m	0
11.	}		
12.	}		0
13.	return sum;	→ 1	
14.	}		

$T(n,m) = 2m + 2n + 05$

$= O(m+n)$

$T(n) = 2n +$

$= \Omega(n)$

$T(n,m,o) = 3n^5 + 4n^3 + 9m^6 + 3m^6 + 3n^7$
$+ 8m + 5o^3 + 8o^2 + 9$

↳ exact time function.

$= O(n^7 + m^6 + o^3)$

Exm 5:

```
int func(int n){
    int prod = 1                    ⟶ 1
    for(int i=1; i<=n; i+=2){       ⟶ n/2 + 1
        prod += 2i;                 ⟶ n/2
    }
    return prod;                    ⟶ 1
}
```

$T(n) = 1 + \frac{n}{2} + 1 + \frac{n}{2} + 1$

$= 2\frac{n}{2} + 3$

$= n + 3 = O(n)$.

for(int i = 1 ; i <= n; i = i+k){ → $\frac{n}{k}+1$
  statement. → $\frac{n}{k}$
}

for(int i = n ; i >= 1 ; i = i-k){ → $\frac{n}{k}+1$
  statement → $\frac{n}{k}$
}

Exm 6 :

```
int function (int n){
   int prod = 0;              → 1
   for(int i = n; i >= 1 ; i = i-4){ → n/4 + 1
        prod += i;            → n/4
   }
   return prod.               → 1
}
```

$$T(n) = \frac{n}{4} + \frac{n}{4} + 1 + 1 + 1$$

$$= 2\frac{n}{4} + 3 = \frac{n}{2} + 3.$$

$$= O(n)$$

Binary Search Time Complexity.

$$n \quad \frac{n}{2} \quad \frac{n}{4} \quad \frac{n}{8} \quad \frac{n}{16} \text{ ---} 1$$

$$\frac{n}{2^0} \quad \frac{n}{2^1} \quad \frac{n}{2^2} \quad \frac{n}{2^3} \quad \frac{n}{2^4} \text{ ---} \frac{n}{2^i}$$

$$\frac{n}{2^i} = 1$$

$$\Rightarrow n = 2^i$$

$$\Rightarrow \log_2 n = \log_2 2^i$$

$$\therefore i = \log_2 n.$$

Exm 7

```
int func(int n){
  int sum=0;                          ⟶ 1
  for(int i=1; i<=n; i=i*2){          ⟶ log2(n)+1
    sum+=i;                           ⟶ log2(n)
  }
  return sum;                         ⟶ 1
}
```

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

$$= 2\log_2(n) + 2$$

$$= O(\log_2(n))$$

---

i=	1	2	4	8	16 ----	n
	2^0	2^1	2^2	2^3	2^4	2^{*x}

$2^{\phi x} = n$

$x\phi = \log_2 n.$

#for(int i=1; i<=n; i*=k) → $\log_k(n) + 1$

// statement. → $\log_k(n)$

for(int i=n; i>=1; i/=k) → $\log_k(n) + 1$

// statement. → $\log_k(n)$

Exm 8!

```
int func(int n){
    int sum = 0;        ——— 1
    int k = n;          ——— 1
    for(int i=
    while(k>=1){        → log_5(n) + 1
        sum += k;       → log_5(n)
        k = k/5;        → log_5(n)
    }
    return sum;         → 1
```

$$T(n) = 1+1+\log_5(n)+1+\log_5(n)$$
$$+\log_5(n)+1$$

$$= 3\log_5(n) + 9$$

$$= O(\log_5(n))$$

DSA II – L-3 – 30 Sep

Exm 9:

```
bool isprime(int n){
    for(int i=2; i*i<=n; i++){
        if(n%i==0)
            return false;
    }
    return true;
}
```

$\sqrt{n}$ ⓐ

$\sqrt{n}-1$

ⓑ 1/0 ⓦ

ⓦ 0/1 ⓑ

wr ~~$T(n) = \sqrt{n}+\sqrt{n}+1+1 = 2\sqrt{n}+2 = O(\sqrt{n})$~~

$T(n) = \sqrt{n}+\sqrt{n}-1+1 = 2\sqrt{n} = O(\sqrt{n})$

bst

~~$T(n) =$~~

$T(n) = 1+1+1+0 = $ ~~$O(0)$~~ $\Omega(1) = \Omega(c)$

Exm - 10

B ω

```
void func1(int n){
  for(int i=1; i<=n; i++){        → n+1   n+1
      func2();                    n  |  n(log2 n)
  }
}
```

func() { Best case: Ω(1)
worst " : O($\log_2 n$)

worst → $T(n) = n+1+(\log_2 n)n$.

~~$= n(\log_2 n + 1) + 1$~~

$= O(n \log_2 n)$.

Best → $T(n) = n+1+n = \Omega(n)$

Exm 11

```
int m,
void func (int n){
int sum=0;
for(int i=1; i<=n; i++){          → n+1
    for(int j=1; j<=m; j++){      → n(m+1)
        sum+=i+j;                 n×m
    {
    sum+=i                        → n
}
}
```

$$T(n,m) = n+1+nm+n+n\times m+n$$

$$= O(n\times m)$$

Exm 12!

```
void function(int n){
  for(int i=n; i>=1; i--){                    ——— n+1
    for(int j=2; j<=n; j=j*3){                -> n(log3 n)
      print                                   -> n(log3 n - 1)
    }
    print                                     -> n
  }
}
```

$T(n) = n+1 + n(\log_3 n) + n(\log_3 n - 1) + n$

$= O(n \log_3 n)$

$f(n) = \frac{n(n+1)}{2}$

$f(n+2) = \frac{(n+2)(n+2+1)}{2}$

$= \frac{(n+2)(n+3)}{2}$

$5+6+7+\cdots+(n+2)$

$$\frac{(n+2)(n+3)}{2} - \frac{4(4+1)}{2}$$

Exm 13

```
void function(int n){                    ——— 1
  sum = 0
  for(int i=1; i<=n; i++){               ——— n+1
    for (j=i; j<=n; j++)                 ——— (n+1)(n+2)/2 - 1
      stat.                              ——— n(n+1)/2
    }
    St. → n
}
return - → 1
```

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

$$= O(n^2)$$

DSA-2-L-4-30th

exm 14!

	Best	worst
int function(int n, int m){		
int sum = 1;	1	1
for(int i=0; i<n; i++){	n+1	n+1
for(int j=0; j<=i; j++){	$\frac{(n+1)(n+2)}{2}-1$	$\frac{(n+1)(n+2)}{2}-1$
print	$\frac{n(n+1)}{2}$	$\frac{n(n+1)}{2}$
sum+=i+j;	$\frac{n(n+1)}{2}$	$\frac{n(n+1)}{2}$
}		
if(sum%2==0){	n	n
for(j=1; j*j<=m; j++){	0	$\frac{n}{2}*\sqrt{m}$
sum	0	$\frac{n}{2}*\sqrt{m}$
}		
}		
}		
return sum;	1	
}		

~~T(n)~~

worst: $T(n,m) = 1 + n + 1 + \frac{(n+1)(n+2)}{2} - 1 + \frac{n(n+1)}{2}$

$+ \frac{n(n+1)}{2} + n + (\sqrt{m} + 1)n + n\sqrt{m}$

$+ 1$

$= O(n^2 + n\sqrt{m})$

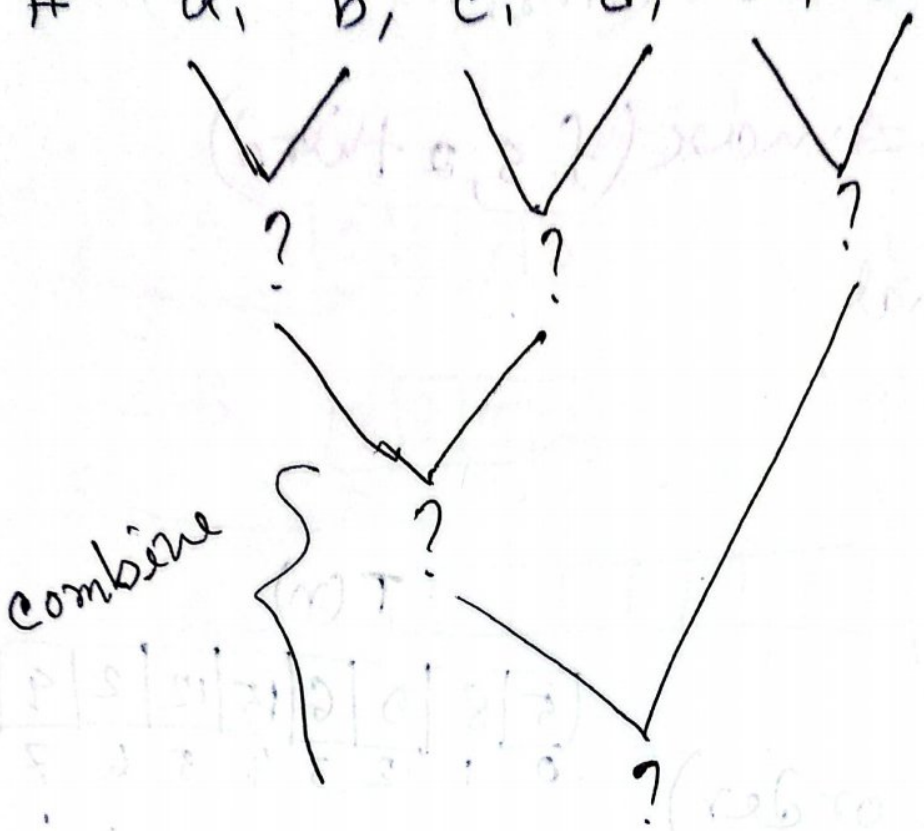
Best: $T(n) = \Omega(n^2)$

Divide & Conquer (DnC)

3 Steps

- ↳ Divide
- ↳ Conquer
- ↳ Combine (merge)

a, b, c, d, e, f → Devide

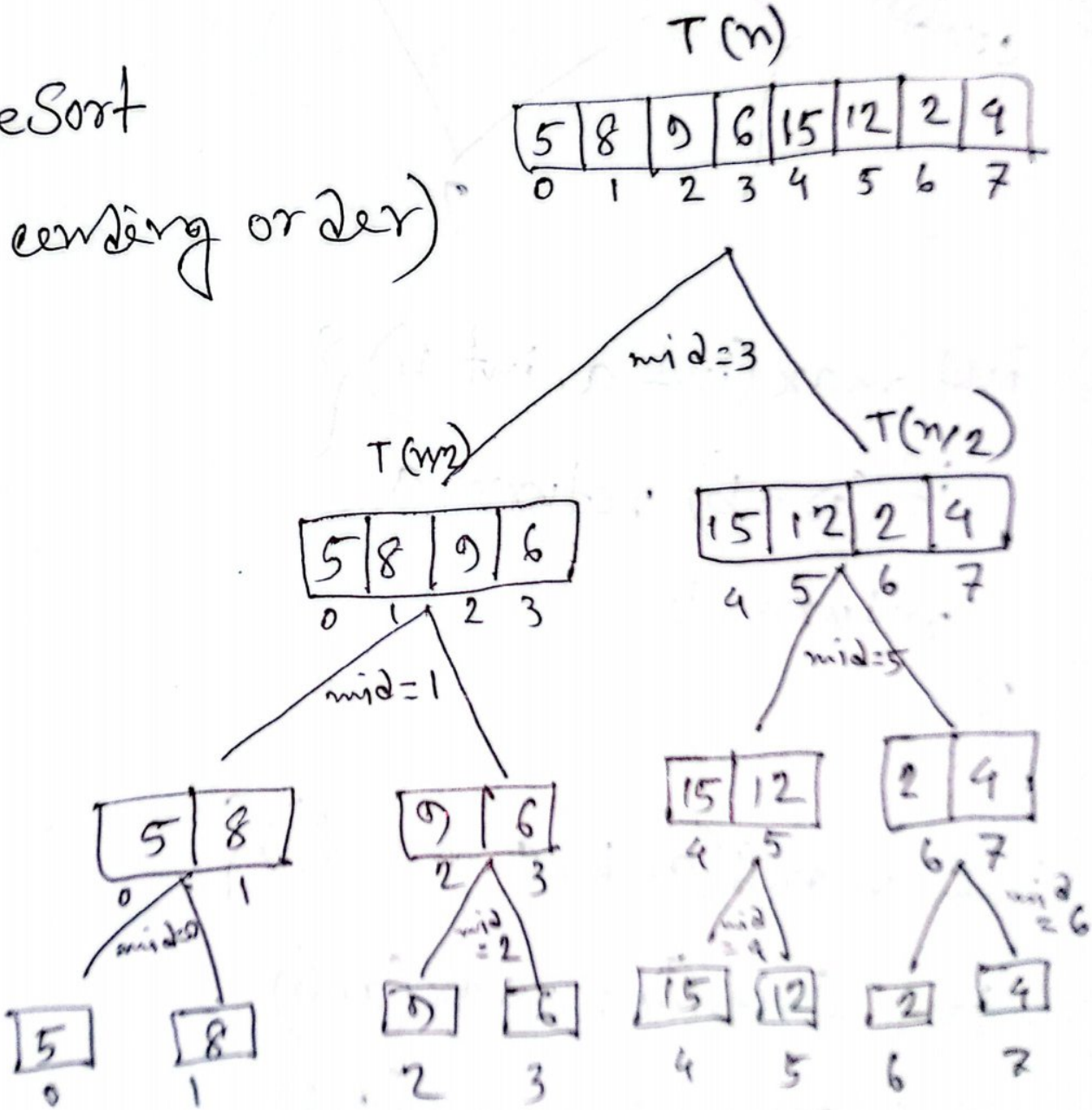


→ Conquer

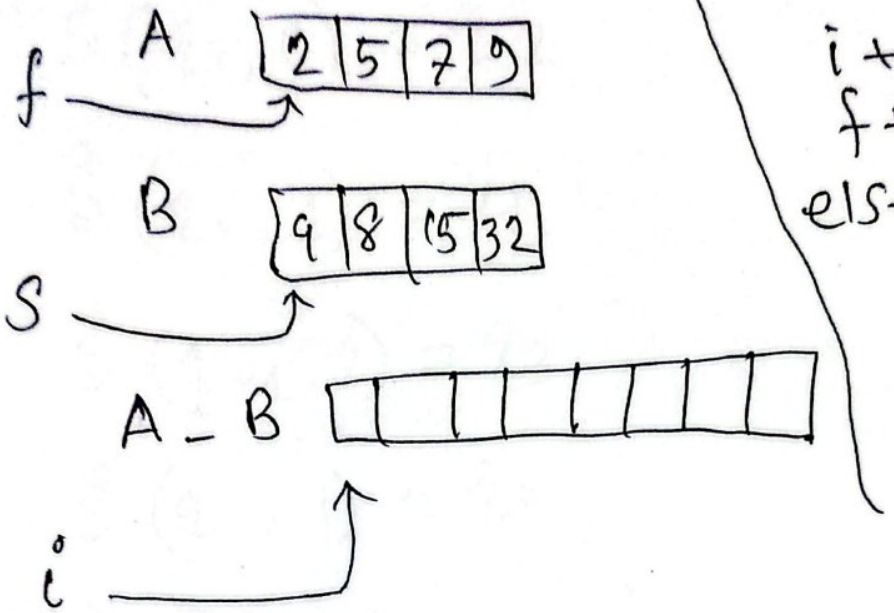
```
int max(int a, int b){
  if(a>=b) return a
  return b
}

int max6(a, b, c, d, e, f,){
  int first = max(a, b)
  int second = max(c, d)
  int third = max(e, f)
```

```
int f.s = max(first, second);
int final = max(f.s, a third)
return final
}.
```

#mergeSort
(ascending order)



merge = $O(n)$




```
if(A[f] <= B[s])
    A_B[i] = A[f]
    i++
    f++
else
    A_B[i] = B[s]
    i++
    s++
```

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + n$$

$$= 2T\left(\frac{n}{2}\right) + n$$

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + n & ; \text{ when } n > 1 \\ 1 & ; \text{ when } n = 1 \end{cases}$$

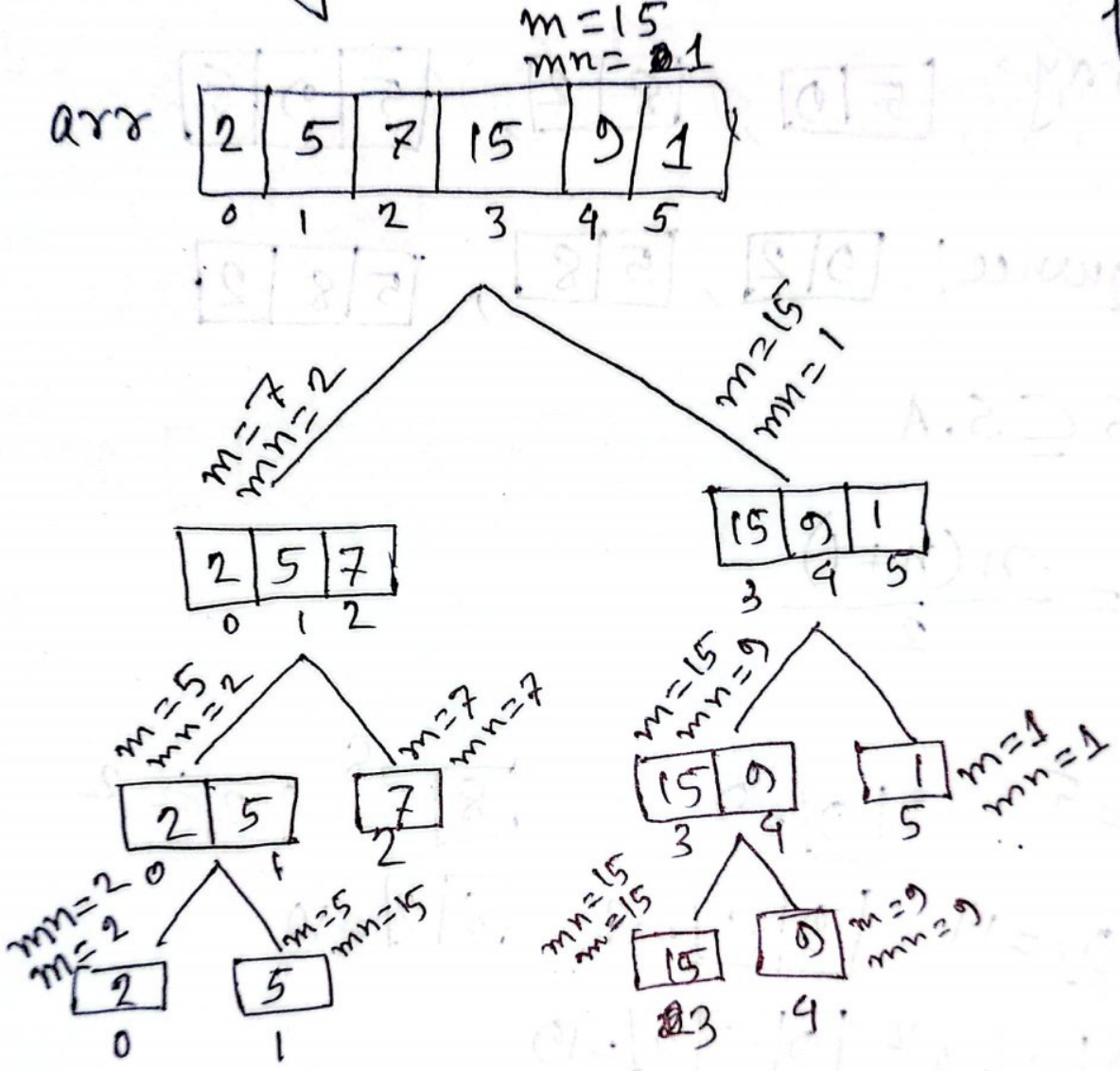
→ conquer time complexity.

DSA II - L-5 - 7 Oct

Min Max Problem

using DnC approach.

m = max
mn = min



$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + C$$

$$= 2T\left(\frac{n}{2}\right) + C$$

$$\therefore T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + C ; & n > 1 \\ 1 ; & n = 1 \end{cases}$$

#Maximum Sum Subarray (DnC)

5	9	8	2

Subarray: | 5 | 9 |, | 8 | 2 |, | 5 | 9 | 8 |

vs

Subsequence: | 9 | 2 |, | 5 | 8 |, | 5 | 8 | 2 |

S.S $\subseteq$ S.A

$$S.A. = \frac{n(n+1)}{2}$$

5	= 5		9	= 9		8	= 8		2	= 2

5	9	= 14		9	8	= 17		8	2	= 10

5	9	8	= 22		9	8	2	= 19

5	9	8	2	= 24

4 3 2 1

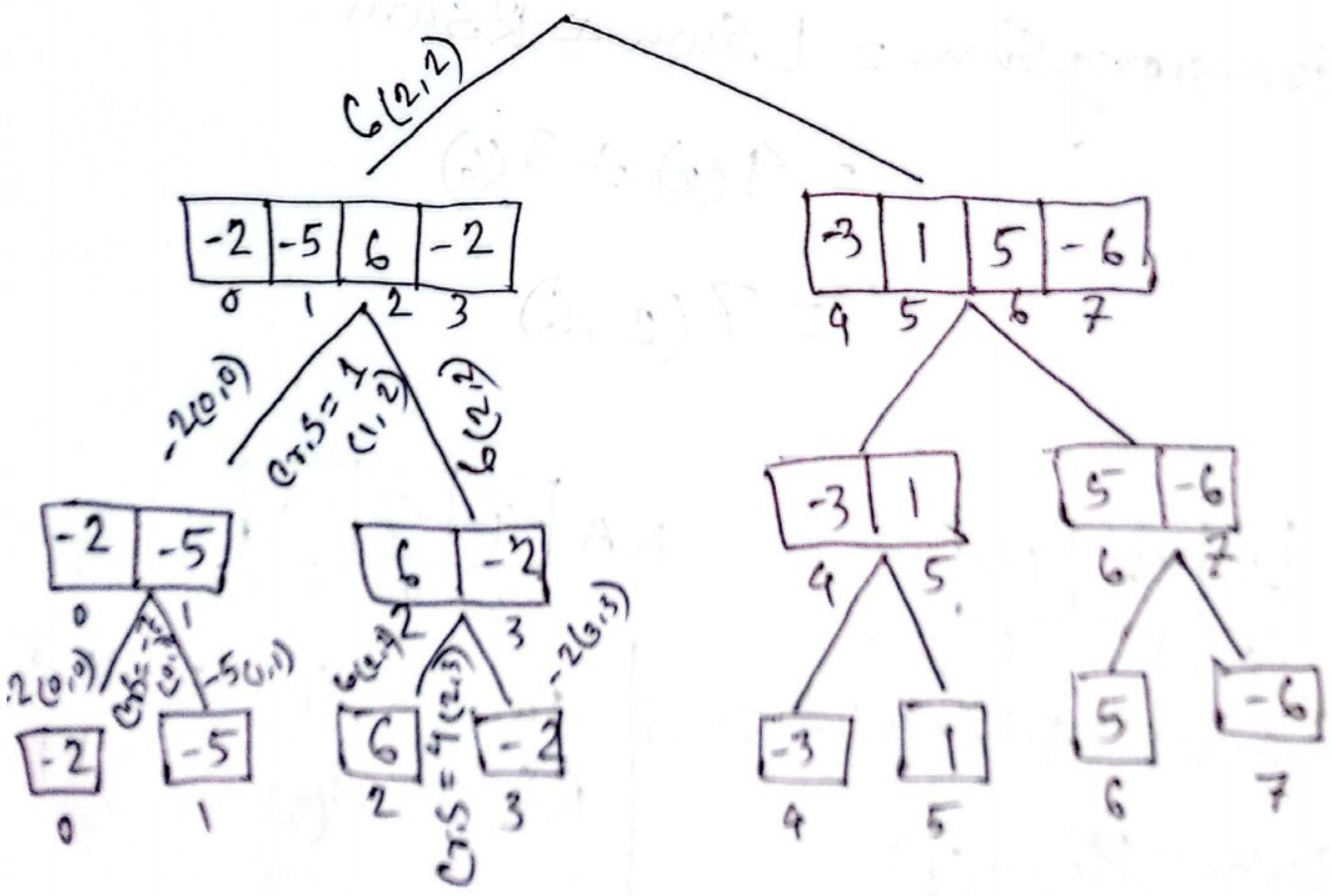
Kadan's Algo : $O(n)$

| -10 | 17 | -15 | 59 |
0 1 2 3

Maximum Sum Subarray = | 17 | -15 | 50 | = 52
1 2 3

$O(n^3) \xrightarrow[\text{(prefix sum)}]{\text{opt}} O(n^2) \xrightarrow{\text{DnC}} O(n \log_2 n)$

| -2 | -5 | 6 | -2 | -3 | 1 | 5 | -6 |
0 1 2 3 4 5 6 7



Crossing Sum

LA

-2	-5	6	-2
0	1	2	3

RA

-3	1	5	-6
4	5	6	7

Sum = -2+6 = 4+(-5) = -1+(-2) = -3

LSum = 4(2)

Sum = -3+1 = -2+5 = 3-6 = -3

RSum = 3(6)

CrossingSum = LSum + RSum

= 4(2) + 3(6)

= 7(2,6)

LA

-15	17	19
0	1	2

RA

-4	-7	-3
3	4	5

Sum = 19+17 = 36-15 = 21

LSum = 36(1)

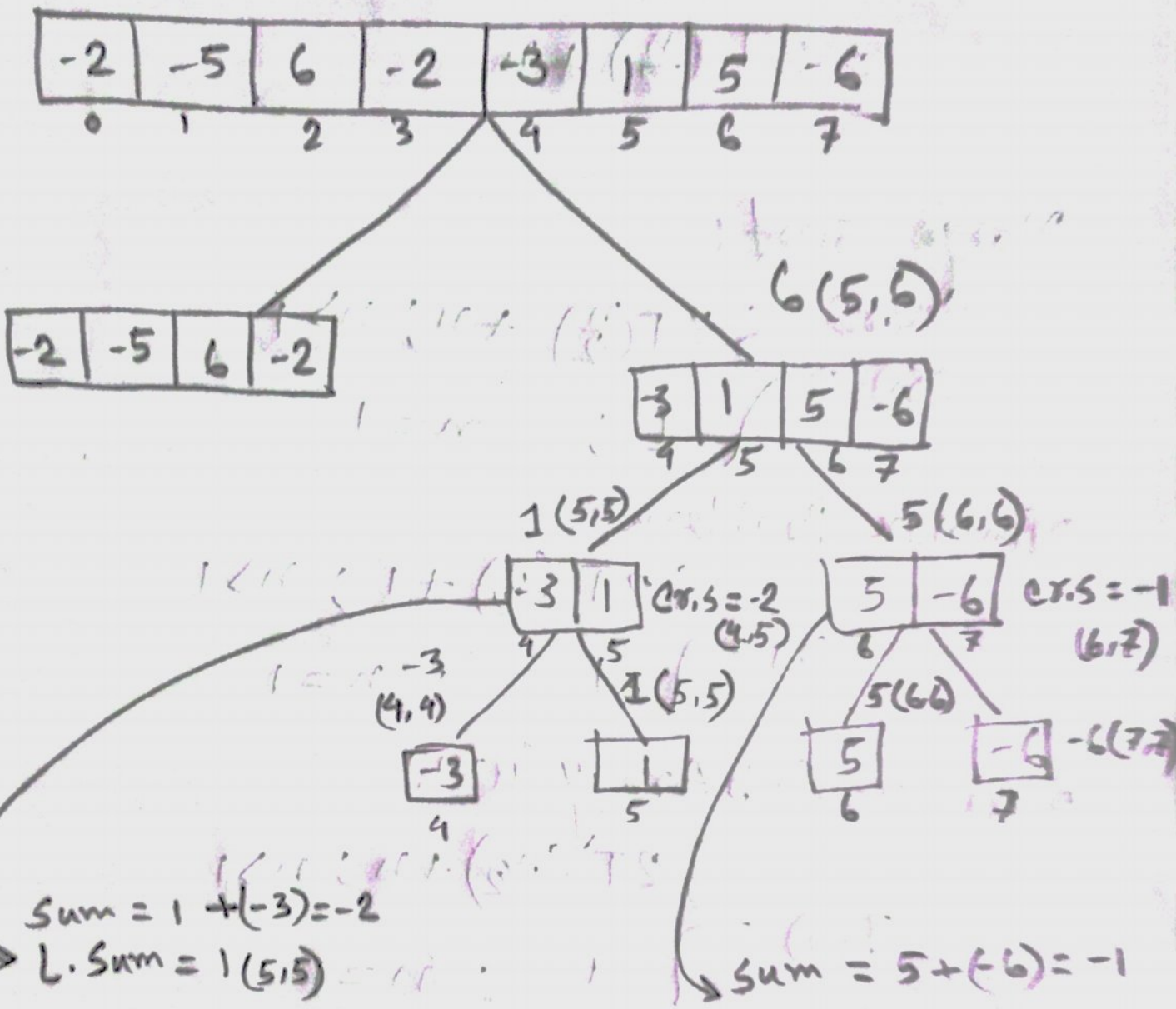
Sum = -4-7 = -11-3 = -14

RSum = -4(3)

∴ Cr.Sum = 36(1) + (-4(3))

= 32(1,3)

DSA II - H.W (L-6-10 Oct)



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

$$= 2T\left(\frac{n}{2}\right) + n$$

Marge sort:

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + n ; & n > 1 \\ 1 ; & n = 1 \end{cases}$$

MinMax Problem:

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + 1 ; & n > 1 \\ 1 ; & n = 1 \end{cases}$$

MaximumSubArraySum:

$$T(n) = \begin{cases} 2T(n/2) + n ; & n > 1 \\ 1 ; & n = 1 \end{cases}$$

Binarysearch:

$$T(n) \begin{cases} T\left(\frac{n}{2}\right) + 1 ; & n > 1 \\ 1 ; & n = 1 \end{cases}$$

$T(n)$

$T(n/2)$

$T(n/4)$

$O(1)$

Recursion solving

→ Substitution

→ Master theorem

→ Recursion tree

Recursion tree:

exm 1:

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + n ; & n > 1 \\ 1 ; & n = 1 \end{cases}$$

Divide $= \frac{n}{a} = \frac{n}{2}$

No. of child (@ each node) $= b = 2$

combine complexity $= \Phi n$

conquer " $= 1$

$$T(n) = \begin{cases} bT\left(\frac{n}{a}\right) \\ +cc \\ con.c \end{cases}$$

cc → combine complexity

con.c → conquer complexity

No. of leaf node $= b^{\log_a n} = 2^{\log_2 n}$

$T(n)$ = internal layer complexity $= n\log_2 n + n$

+ leaf layer " $= O(n\log_2 n)$

leaf layer complexity = (No. of leaf node × conquer complexity)

internal layers complexity:

$(\log_2 n) \times n = n \log_2 n$

→ leaf layer complexity $= 2^{\log_2 n} \times 1$

$= n^{\log_2 2}$

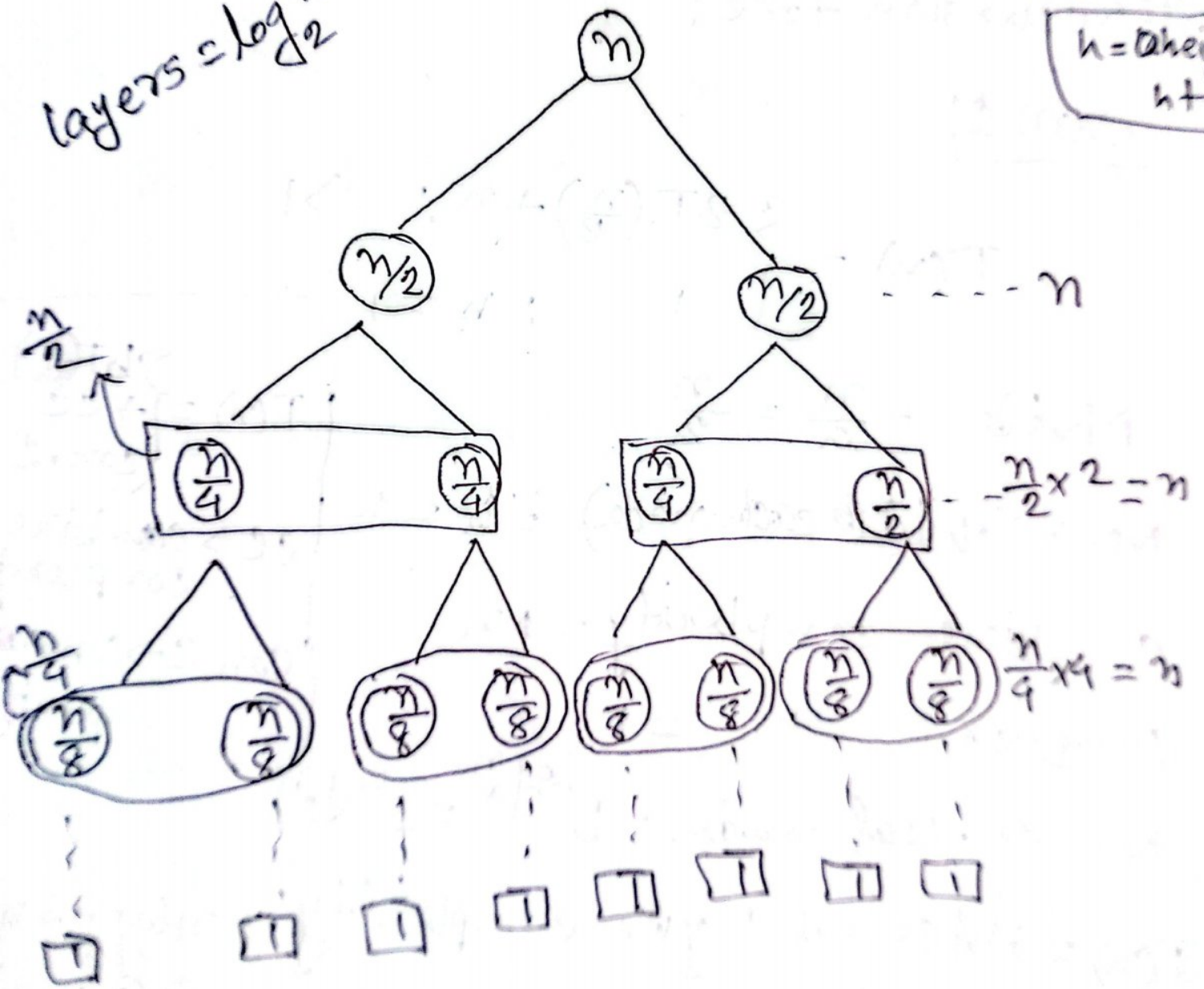
$= n$

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
⋮
$2^h = ?$

h = theight

Tree:

layers $= \log_2 n$



$$a^{\log_k b} = b^{\log_k a}$$

exm-2

$$T(n) = \begin{cases} 2T(\frac{n}{2}) + 1 & ; n > 1 \\ 1 & ; n = 1 \end{cases}$$

Divide $= \frac{n}{a} = \frac{n}{2}$

No. of child (each node) $= b = 2$

combine complexity $= 1$

conquer $= 1$

No. of leaf node $= b^{\log_a n} = 2^{\log_2 n} = n$

internal layers complexity:

$c + 2c + 4c + 8c + 16c + \cdots$

$= c(1 + 2 + 4 + 8 + 16 + \cdots)$

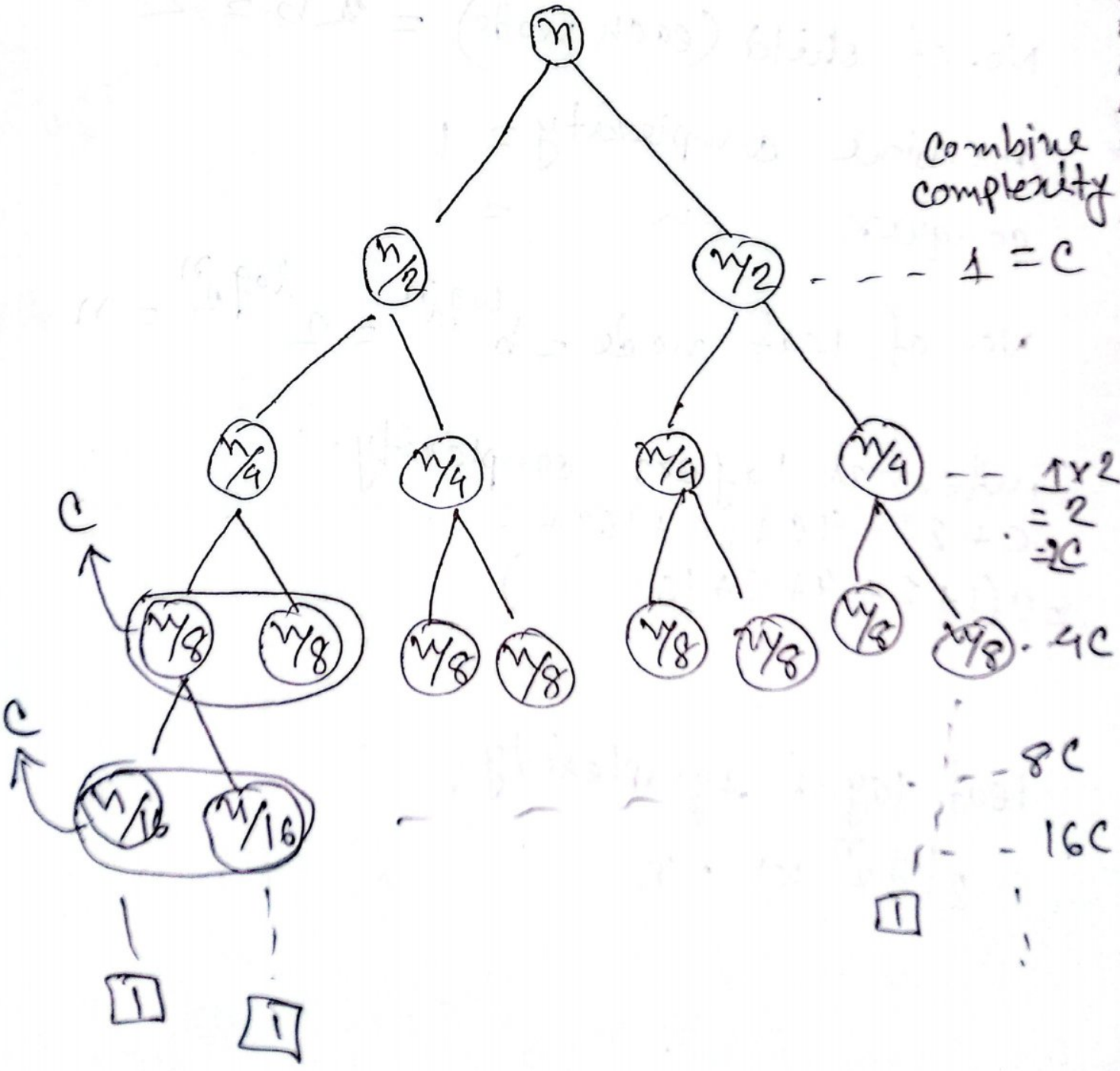
leaf layer complexity:

$2^{\log_2 n} \times 1 = n$

$T(n)$ = internal layer complexity
+ leaf layer complexity

$= c(1+2+4+8+--)+n$

$= O(n)$



Exm: 3

$$T(n) = \begin{cases} T\left(\frac{n}{2}\right) + 1 & ; n > 1 \\ 1 & ; n = 1 \end{cases}$$

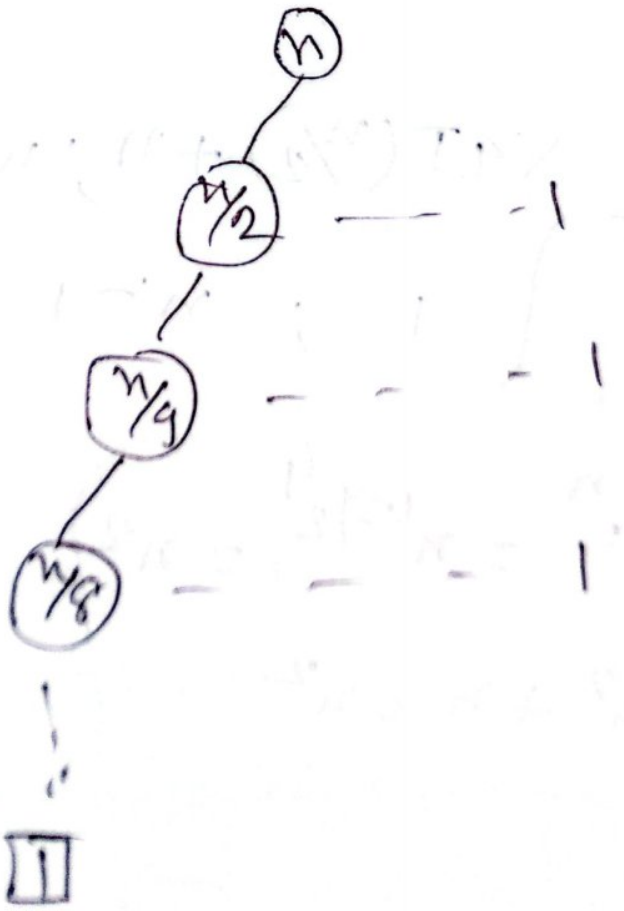
Divide $= \frac{n}{a} = \frac{n}{2}$

No. of child (each node) $= b = 1$

combine complexity $= 1$

conquer " $= 1$

no. of leaf node $= b^{\log_a n} = n^{\log_2 1} = n^0 = 1$



internal layers complexity:

$$\log_2^n \times 1 = \log_2^n$$

leaf layers :

$$1^{\log_2^n} \times 1 = 1$$

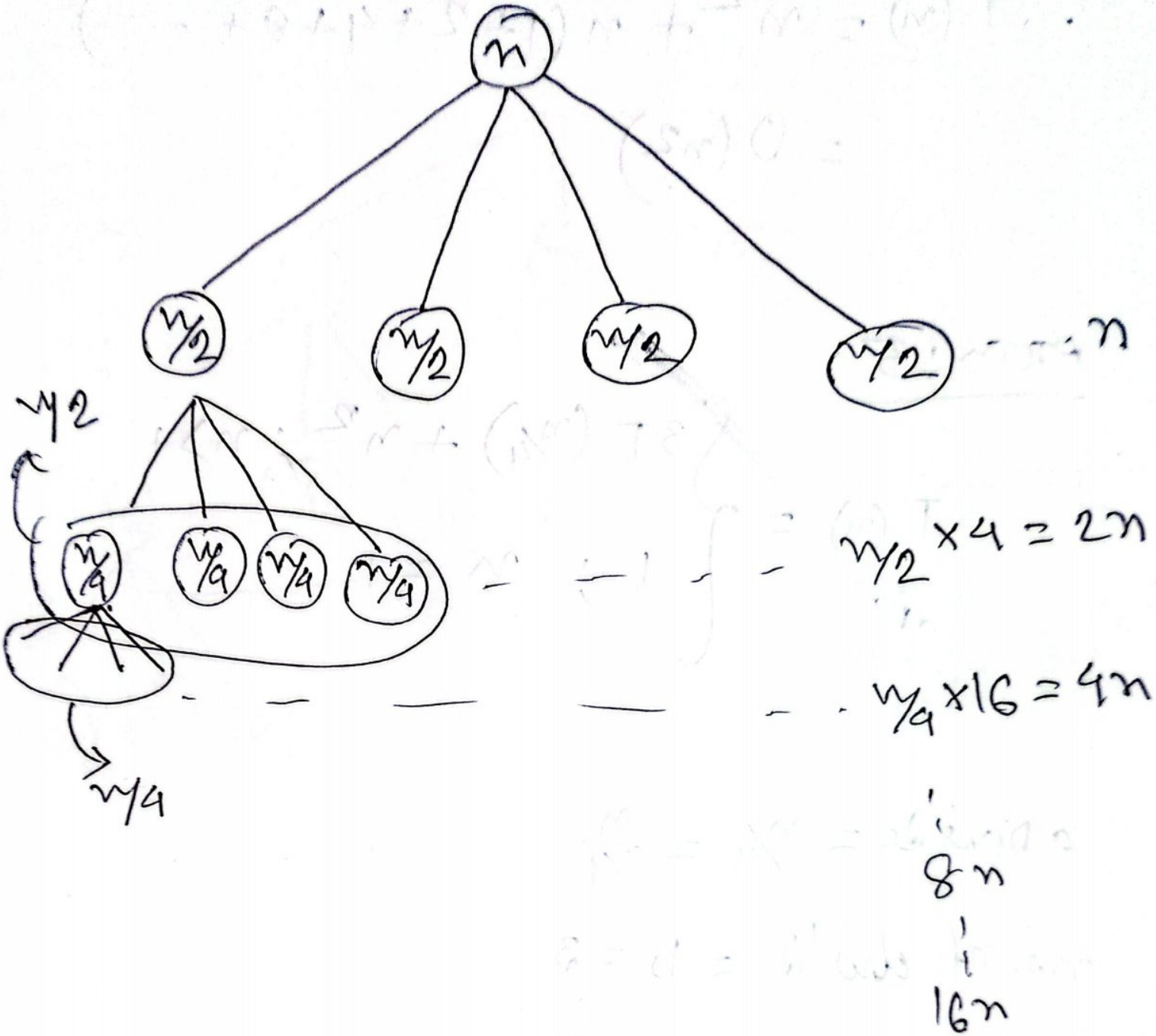
$$\therefore T(n) = \log_2^n * 1 = O(\log_2^n)$$

exm - 4

$$T(n) = \begin{cases} 4T(n/2) + n ; & n > \\ 1 ; & n = 1 \end{cases}$$

$$4^{\log_2^n} = n^{\log_2^4} = n^2$$

$$n^2 + n = n^2$$



internal layers complexity:

$n + 2n + 4n + 8n + \cdots$

$n(1 + 2 + 4 + 8 + \cdots)$

leaf layer complexity:

$n^2 \times 1 = n^2$

$$\therefore T(n) = n^2 + n(1+2+4+8+\cdots)$$

$$= O(n^2)$$

exm : 5

$$T(n) = \begin{cases} 3T(n/4) + n^2 ; & n > 1 \\ 1 ; & n = 1 \end{cases}$$

a Divide $= n/a = \frac{n}{4}$

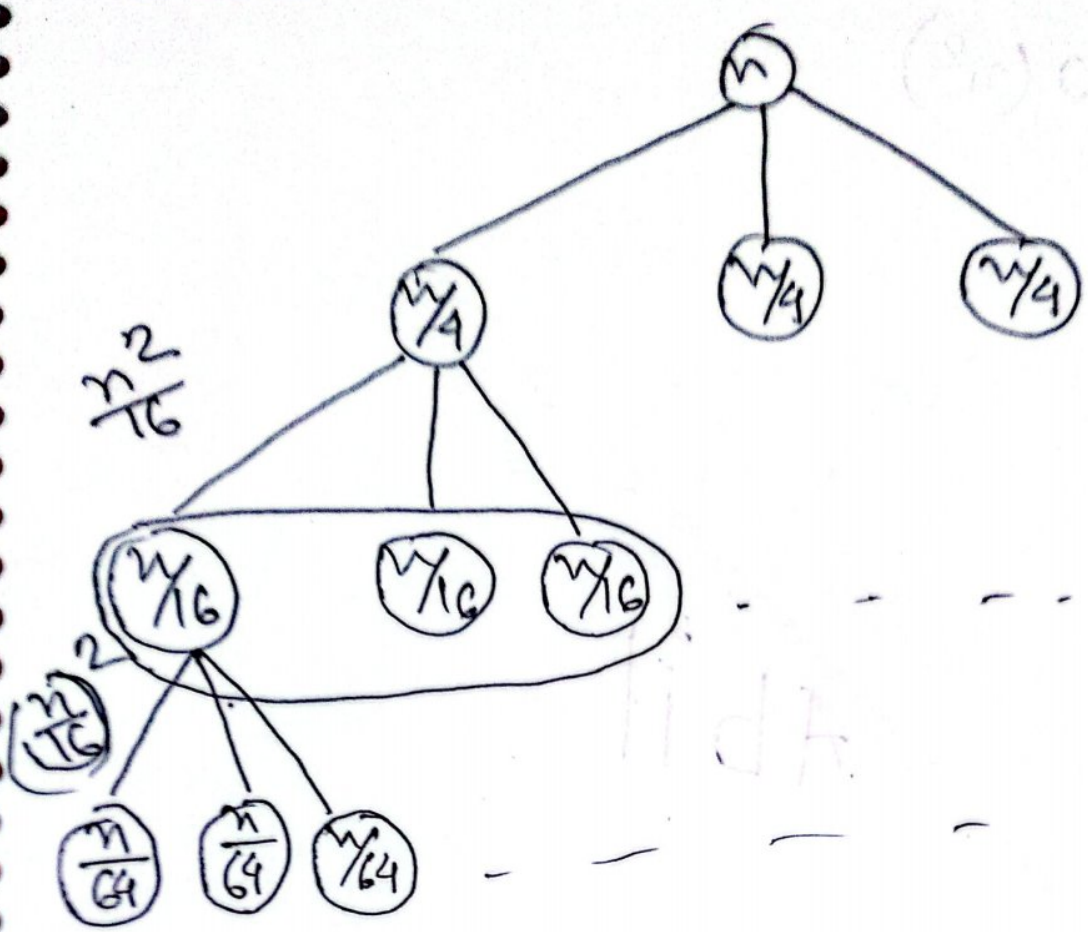
no. of child $= b = 3$

combine com. $= n^2$

conquer $= 1$

No. of leaf node $= 3^{\log_4 n} = n^{\log_4 3} = n^{0.79}$

tree:



$\left(\frac{3}{16}\right)^0 n^2$

$3 \times \frac{n^2}{16} = \frac{3}{16} n^2$

$9 \times \frac{n^2}{256} = \left(\frac{3}{16}\right)^2 n^2$

$\left(\frac{3}{16}\right)^3 n^2$

internal layer comp.

$n^2 + \frac{3}{16} n^2 + \left(\frac{3}{16}\right)^2 n^2 + \cdots$

$= n^2 \times c = cn^2$

leaf " "

$n^{0.79} \times 1 = n^{0.79}$

$$T(n) = cn^2 + n^{0.79}$$

$$= O(n^2)$$

DSA II - L-8 - 16 oct

Greedy Approach / Strategy

Coin change

ex1

Change Amount = 52727

bill = 47273

pay = 100000

coin/notes = [~~1000~~, 1000, 500, 200, 100, 50, 20, 10, 5, 2, 1]

minimum number of coins = 57

Exm-2

Amount = 30

coins = [25, 10, 1]

Current amount = 30

25 X 1 = 25

" " = 5

1 X 5 = 5

" " = 0

Total coins = 6

[minimization ~~maximization~~ ---
--→ Greedy
optimal এর ~~এর~~ times
Greedy ^ 2.7 ^ এর থেকে worst solution
~~&~~ বেরনো]

# amount = 127
coins = [50, 1, 20, 5, 10, 17]

→ sorted coins descending order
= [50, 20, 17, 10, 5, 1]

C.A. = 127
50 x 2 = 100
R.A = 27

C.A = Current Amount
R.A. = Remaining

$20 \times 1 = 20$

R.A $= 7$

$5 \times 1 = 5$

R.A. $= 2$

$2 \times 1 = 2$

0

Total coins $\geq$

optimal soln. $= 4$

$[4, 4 \times 2.7]$

$= [4, 10.8]$

$\therefore 4 \leq 6 \leq 10.8$

Activity Selection (maximization)

[always optimal solution (4?) Greedy (4?) ~~or~~ case-4.]

Activity	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
Start	1	0	1	4	2	5	3	4
Finish	3	4	2	6	9	8	5	8

→ Sort the tasks based on-

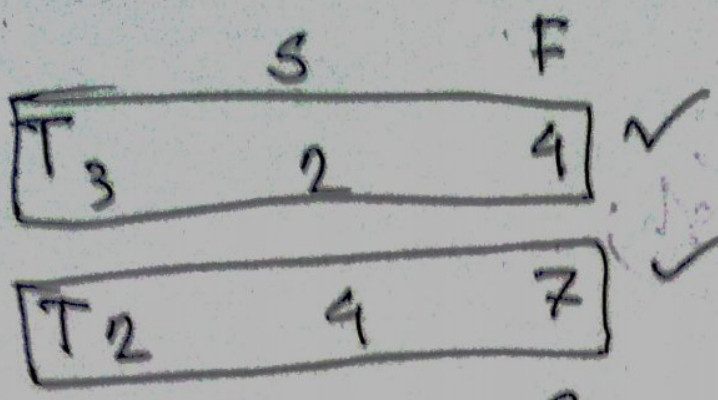
- Start time
- Finish " * (optimal solution)
- Duration

	S	F
T_1	1	8
T_2	4	7
T_3	2	4
T_4	3	9

sort → start

	S	F	
T_1	1	8	✓
T_3	2	4	1 ✓
T_4	3	9	
T_2	4	7	

Sort → Finish



Sort → Finish time

Activity	a_3	a_1	a_2	a_7	a_4	a_8	a_6	a_5
start	1	1	0	3	4	4	5	2
Finish	2	3	4	5	6	8	8	9

Picked = 1 + 1 = 2 + 1 = 3

tasks = $[a_3, a_7, a_6]$

availability = 2 ~~3~~ 8

```
struct Task{
    int start;
    int finish;
};

int activitySelection (Task tasks[], int n){
  sort (tasks); // based on finish time
  int picked = 1
  int availability = tasks [0]. finish
  for(int i=1; i<n; i++){
     if (tasks[i].stoort >= availabity){
        picked++;
        availability = tasks[i].finish;
     }
  }
  return picked;
}
```

[Sort - এর optimal time complexity
$O(n \log n)$

DSA II - 2 - 10 - 21 oct

Greedy -

knapsack

↳ 0/1 knapsack

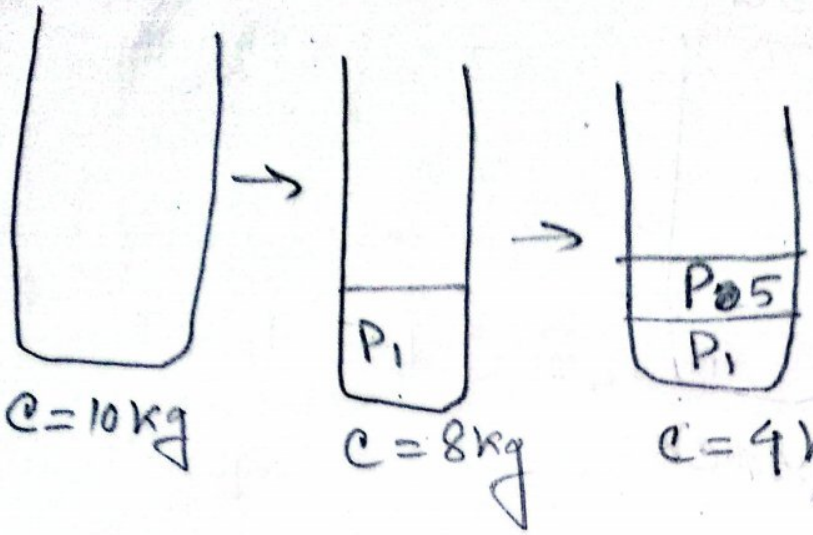
↳ fractional "

0/1 knapsack

Product	weight	value	value/weight
1 ← P_1	2	10	5
5 ← P_2	5	15	3
4 ← P_3	6	30	3.33
3 ← P_4	4	20	~~3.33~~
2 ← P_5		18	4.55

Capacity = 10 kg

#



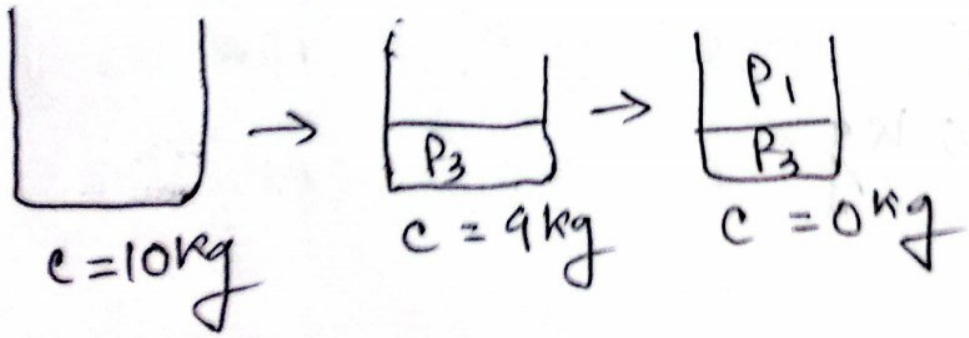
w: 2+4 = 6kg

v: 10 + 18 = 28$

#

Product	weight	value	value/weight
3← P_1	4	20	5
2← P_2	5	30	6
1← P_3	6	40	6.67

capacity = 10 kg



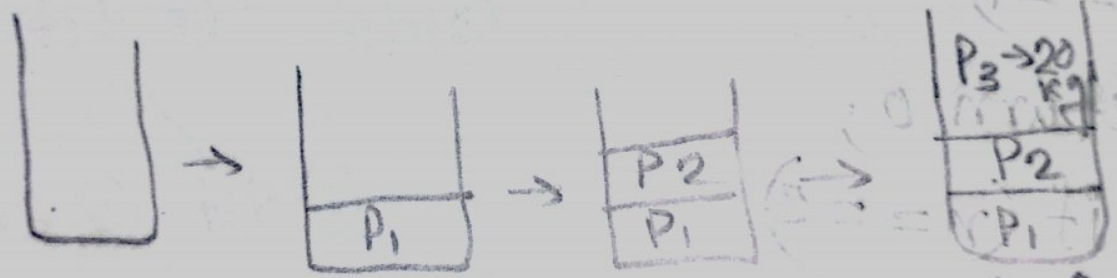
∴ w: 6+4=10kg

v: 40+20=60$

fractional knapsack

Product	weight	value	v/w
		60	6
1 ← P_1	10		
	20	100	5
2 ← P_2			
	30	120	4
3 ← P_3			

Capacity = 50 kg.



c=50kg c=40kg c=20kg c=0

∴ w: 10+20+20 = 50kg

v: 60+100+80 = 240$

*Greedy never fails in fractional knapsack.

Dynamic Programming

fibonacchi series:

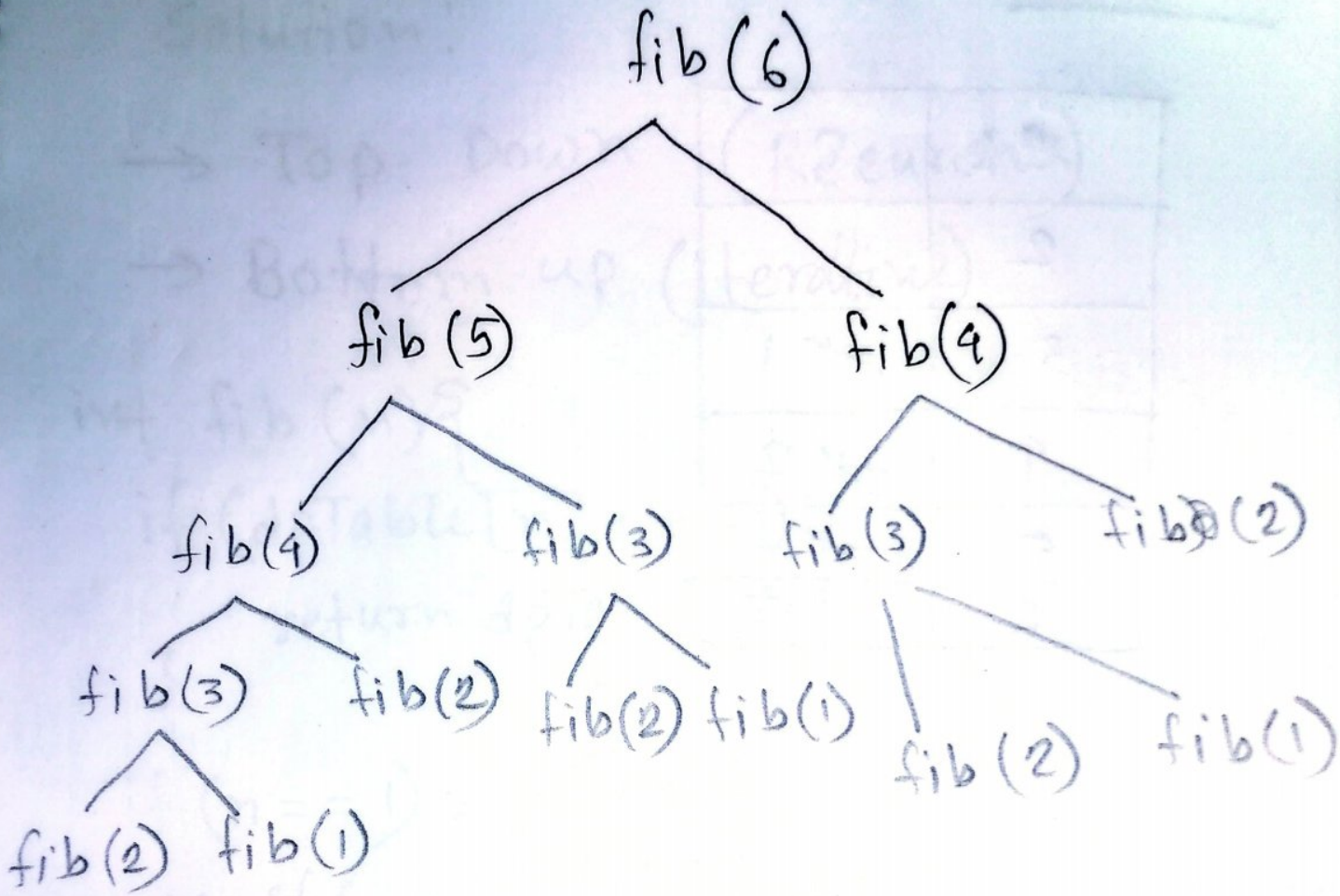
0 1 1 2 3 5 8 13 21

1st 2nd

3 4 _ _ _ _

```
int fib (int n) {
    if (n==1)
        return 0;
    else if (n==2)
        return 1;
    else{
        int fnd1 = fib(n-1);
        int fnd2 = fib(n-2);
        int result = fnd1+fnd2;
        return result;
}
}
```

if n = 6.

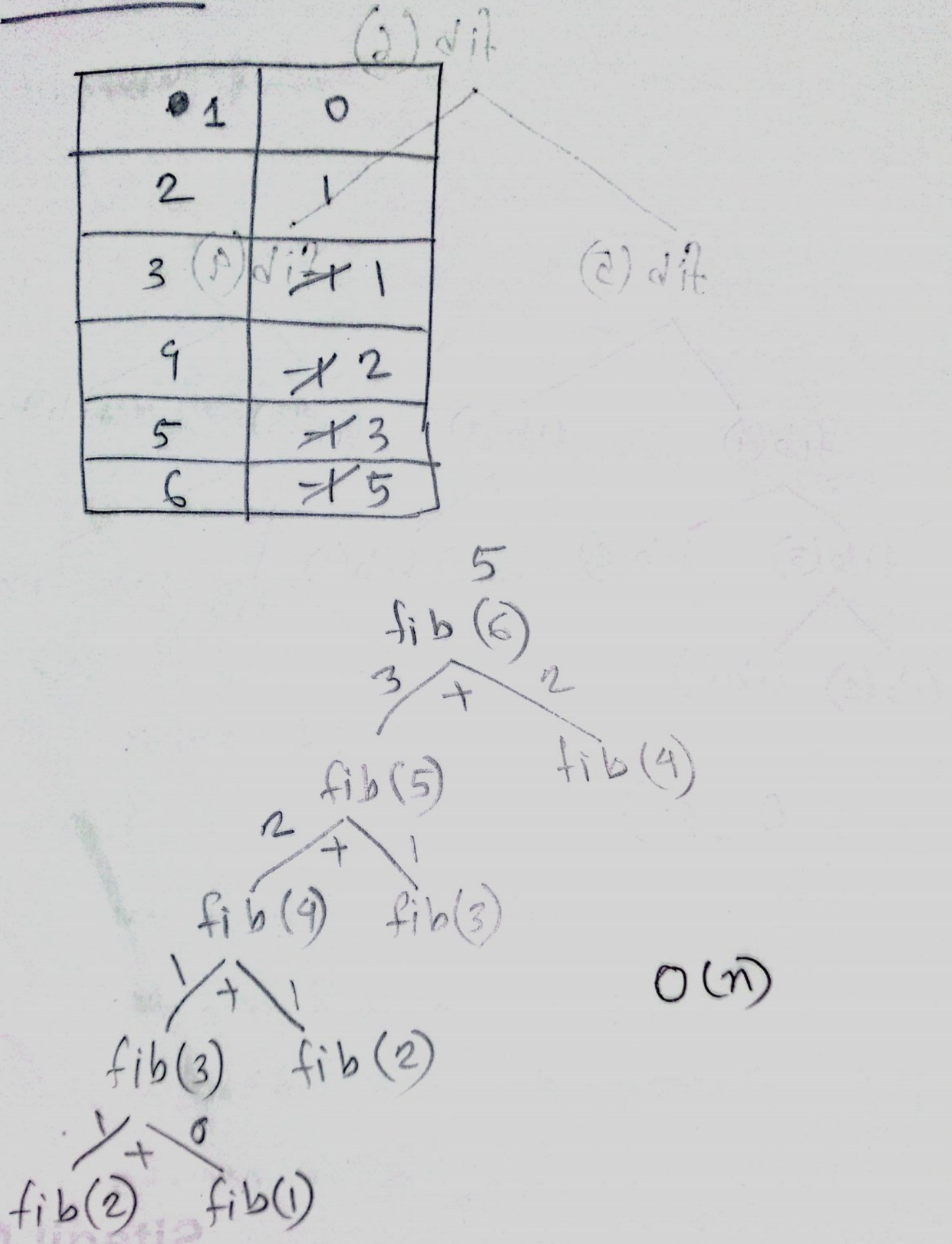


$O(2^n)$

DP Table

Top Down (Recursive)

1	0
2	1
3	~~1~~ 1
4	~~2~~
5	~~3~~
6	~~5~~



O(n)

2 ways to implement a DP

Solution:

→ Top Down (Recursive)

→ Bottom up (iterative)

```
int fib(n){
  if (dpTable[n] != -1){
      return dpTable[n];
  }
  if (n == 1) return 0;
  else if(n == 2) return 1;
  else{
     int fnd1 = fib(n-1)
     int fnd2 = fib(n-2)
     int result = fnd1+fnd2
     dpTable[n] = result;
     return result;
  }
}
```

DP Applys for overlaping sub-problem.

DSA II - L-11 - 30 oct

Dynamic Programming

Coin Chang.

optimal substructural property

ammount = 11

coins = [1, 5, 6, 8]

coin / amount	0	1	2	3	4	5	6	7	8	9	10	11
1	0	1	2	3	4	5	6	7	8	9	10	11
5	0	1	2	3	4	1	2	3	4	5	2	3
6	0	1	2	3	4	1	1	2	3	4	2	2
8	0	1	2	3	4	1	1	2	1	2	2	2

whether we'll take it or not:

amount = 5 - 5 = 0

take: 1 + 0 = 1

not take: 5

amount = 6

· coins = [2, 3, 5]

∞ = 0x7f7f

coins \ amount	0	1	2	3	4	5	6
2	0	∞	1	∞	2	∞	3
3	0	∞	1	1	2	2	2
5	0	∞	1	1	2	1	2