⇒ Amortized Cost Analysis:-

- It is also known as average cost Analysis. It is the method of analysing the cost associated with the data structure that averages the worst operations out over time. This is required because one worst case cannot signify the overall performance of an algorithm. So, Amortize analysis is used to average out the costly operations in the worst case.

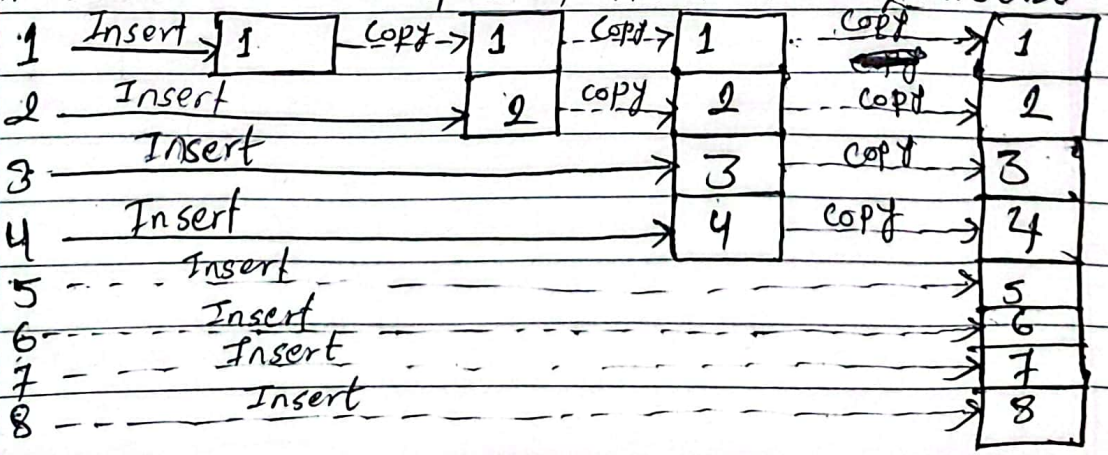
- There are three main types of amortized analysis:

① Aggregate Analysis.
② Accounting Analysis.
③ Potential Analysis.

1) Aggregate Analysis:-

In aggregate analysis, there are two steps, first, we must show that the sequence of 'n' operations takes T(n) time in the worst case Then, we show that each operation takes $\frac{T(n)}{n}$ time on average. Therefore, in aggregate analysis each operation has some cost.

Consider an example of Hash Table Insert.



Insert	i	1	2	3	4	5	6	7	8	9	10
size	i	1	2	4	4	8	8	8	8	16	16
cost	c_i	1	2	3	1	5	1	1	1	9	1

Here, c_i is the cost involved, which is the sum of copy + insert

now,

$$c_i = \begin{cases} i & \text{if } (i-1) \text{ is the exact power of 2} \\ 1 & \text{otherwise.} \end{cases}$$

Again,

$$c_i = n + \sum_{j=0}^{\lfloor \log_2 i \rfloor} 2^j$$

or, $c_i = [1+1+1+1+\cdots) + (1+2+4+8+\cdots)$

As this is the geometric series, it can be generalized as,

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

Average cost $O(n) = \frac{O(n)}{n} = 1$

Amotized cost Analysis of hash table, if the table doubling is also included.

→ While using dynamic table as array, following are the step needed to be followed, when table become full.

i) Allocate memory for twice the old table size.
ii) copy the contents of old table to new table.
iii) free the old table.

If the table doubling cost is also included then,

Item	size	operation cost	copying cost	Doubling cost	Total
1	1	1	0	0	1
2	2	1	1	1	3
3	4	1	2	1	4
4	8	1	4	0	1
5	8	1	4	1	6
6	8	1	0	0	1
7	8	1	0	0	1
8	8	1	0	0	1
9	16	1	8	1	10
10	16	1	0	0	1

Here, C_i = operation cost + copying cost + Doubling cost

i.e. $C_i = n + \sum_{i=0} 2^i + \sum_{j=0} 1^j$

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

$= 2n + 2n - 3$

$= 4n - 3$

$= O(n)$

$\therefore \hat{C}_i = O(n) = O(n)/n = 1$ #

2) Accounting Analysis:-

The accounting method is apply named because it borrows ideas and terms from accounting. This method is burroweed from finance where there is the concept of ~~storing~~ storing excess amount in the bank for future use. Here, each operation is assigned a

charge called the amotized cost. Some operation can be charged more or less than they actually cost, we assign the difference called credit. Creadit can be never be negative in any sequence of operations. This method requires setting the cost of $\hat{c}_i$ (Amotized Cost), which should be equal for all inserts such that. Bank ≥ 0 (cost) which is turn will satisfy the inserts of all the input $\hat{c}_i = 3$.

Q.N-1

$\leftarrow$ Bank$_i$ = ~~Bank$_i$~~ + ($\hat{c}_i - c_i$)

$\hat{c}_i$	3	3	3	3	3	3	3	3	3	3
Bank$_i$	2	3	3	5	3	5	7	9	3	1
c_i	1	2	3	1	5	1	1	1	9	1

$\hat{C}_i = 3n$

~~Q~~

i	1	2	3	4	5	6	7	8	9	10
size$_i$	1	2	4	4	8	8	8	8	16	16
c_i	1	2	3	1	5	1	1	1	9	1

$\hat{c}_i$	3	3	3	3	3	3	3	3	3	3
Bank	2	2	3	5	3	5	7	9	3	5

if guess = 2

$\hat{c}_i$	2	2	2	2	2	2	2	2	2	2
Bank	1	1	0	1	-1					

which is invalid.

So, $\hat{c}_i = 3$ $\leftarrow$ limit is correct guess.

If every 1 operation cost is 3 then, n operation cost is 3n.

$\therefore O(n) = \frac{3n}{n} = 3 = 1$ (constant).

3) Potential Analysis :-

Here, we have to define potential function (ϕ).

$\phi(D_0)$

$\phi(D_i)$

$\phi(D_{i-1})$

potential difference $= \phi(D_i) - \phi(D_{i-1})$

Lets assume $\phi(D_i) = 2i - 2^{\lceil \log i \rceil}$

$\phi(D_{i-1}) = 2(i-1) - 2^{\lceil \log (i-1) \rceil}$

$\hat{C}_i = C_i + \phi(D_i) - \phi(D_{i-1})$

Here,

$$C_i = \begin{cases} i & \text{if } i-1 \text{ is an exact power of 2} \\ 1 & \text{otherwise.} \end{cases}$$

$$\therefore \hat{C}_i \begin{cases} i, & \text{if } i-1 \text{ is an exact power of 2} \\ 1 & \text{otherwise} \end{cases}$$

$\cdot 2i - 2^{\lceil \log i \rceil} - [2(i-1) - 2^{\lceil \log (i-1) \rceil}]$

Case I :-

$\hat{C}_i = i - 2i - 2^{\lceil \log i \rceil} - 2i - 2 - 2^{\lceil \log (i-1) \rceil}$

$= i - 2(i-1) + 2 + (i-1)$ generalization.

$= i - 2i + 2 + 2 + i - 1$

$= 4 - 1$

$= 3$

Case II :-

$\hat{C}_i = 1 - 2^{\lceil \log i \rceil} + 2 + 2^{\lceil \log (i-1) \rceil}$

$= 1 - (i-1) + 2 + (i-1)$ (generalization)

$= 1 - i + 1 + 2 + i - 1$

$= 3$

⇒ Randomized Algorithm -:

- A probabilistic algorithm is an algorithm where the result and/or the way the result is obtained depend on chance. These algorithms are also sometimes called randomized algorithms.
- It makes use of randomizer (random number generator) to make some decision in a algorithm.
- The output and/or execution time of a randomized algorithm could also differ from run to run for the same input.
- It can be categorized into:
  1) Las Vegas Algorithm
  2) Monte Carlo Algorithm

⇒ Las Vegas Algorithm -:

- The algorithm always same (correct) output for the same input.
- Execution time of this algorithm is characterized as a random variable.
- The execution time depends on the output of randomizer.
- Type of Randomized quick sort.
- It guarantees the correctness of the solution.
- Upper bound cannot fixed.

⇒ Monte Carlo Algorithm -:

- It is a algorithm whose output might differ from run to run for the same input.
- It does not guarantees the correctness of the solution.
- Type of primility Testing.
- Upper bound can be fixed.
- The execution time is fixed.

CamScanner

⇒ Las Vegas Algorithm

```
void LasVegas()
{
  while (1) {
    int i = random()%2;
    if (i) return;
  }
}
```

```
LasVegasSearch(A,n) {
  while (true)
  {
    randomly choose an
    element out of n.
    if (a found)
      return true;
  }
}
```

⇒ monte Carlo Algorithm -:

```
MonteCarloSearch(A, n, x) {
  p=0
  while (p≤x) {
    randomly choose an element out of n.
    If (a found) {
      flag = true;
    }
    return flag;
  }
```

⇒ primality Testing -:

- to check whether the given number is prime or not

⇒ Bruth force Algorithm

If n is the no to be checked

```
i = 2
for i = 2 to n-1
n % i = 0
  n == i
    prime
  not prime.
```

→ If $a^{n-1} \% n == 1$

prime

else

not prime.

n is the number to be checked.

$a < n$ (the numbers)

let $n = 2$

$a = 1$

$\Rightarrow 1^{2-1} \% 2 \Rightarrow 1 \% 2 \neq 1$

$n = 3$

$a = 1, 2$ and $2^{3-1} \% 3 = 4 \% 3$

$\Rightarrow 1^{3-1} \% 3 = 1 \% 3 = 1$ $= 1$

$n = 4$

$a = 1, 2, 3$

If $a = 1 \Rightarrow 1^{4-1} \% 4 = 1$

If $a = 2 \Rightarrow 2^{4-1} \% 4 = 0$ ⇒ complexity - O(1)

If $a = 3 \Rightarrow 3^{4-1} \% 4 = 3$ // · monte carlo

⇒ Primality Testing :-

- Any integer greater than one is said to be a prime if its only divisors are 1 and the integer itself. By convention, we take 1 to be a non-prime, then 2, 3, 5, 7, 11, and 13 are the first six primes. Given an integer n, the problem of deciding whether n is a prime is known as primality testing.

```
Las Vegas () {
    while (true) do {
        i := Random () mod 2;
        If (i >= 1) then return;
    }
}
```

Unit-1.2 : Advance Algorithm Design Techniques :-

⇒ Greedy Algorithms :-

- It is a simplest and straight forward method (approach).
- It take decision on the basis of current available information without worrying about the effect of the current decision in future.
- It works in phases. At each phase :
  - You take the best you can get right now, without regard for future consequences.
  - You hope that by choosing a local optimum at each step.
  - You will end up at a global optimum.
- At each stages decision is made regarding whether particular input is an optimal solution.

```
Greedy ( A[], n ) {
    for (i=1 ; to n ) {
        x = select (A)
        if ( feasible x ) {
        solution = solution + x ;
    }
    return solution ;
}
```

eg Huffman coding, Shannon Fano Algorithm Job scheduling etc.

→ lossless data encoding schemes is disadvantage.
→ Relatively slower process.

⇒ Huffman coding :-

- By David A. Huffman (1949)
- Huffman coding is a very popular algorithm for encoding data.
- It is greedy (best for greedy approach) algorithm.
- It reduces the average access time of codes as much as possible.
- It is a unique code generator algorithm.
- It generate always prefix code (optimal prefix).
- It is a lossless data compression algorithm.
- It use variable length code words to represent character's in a message.
- Shorter code words are assign to more frequent characters, and longer code words to less frequent characters.
- This reduces the total no of bits required to represent the message.
- Algorithm construct the binary tree of nodes where each leaf node represents a character in the message.
- The frequency of each character is used to determine the position of nodes in the tree.
- character with higher frequency are closer to the root, while those with lower frequency are farther from the root.
- Root to leaf node to generate binary code.
- The resulting code words are prefix-free, ensuring unique decoding.
- Huffman coding are used in data compression, image and video compression, and file archiving.
- eg best is Huffman coding for greedy.

Example-1

String = { "aaaa bbb ·e f ;;;;;;;" }

Frequency = {a=4, b=3, e=1, f=1, & ;=6}

Sol^n

A character is 1 byte long = 8-bit

∴ 8 × 19 = 152 bit (∵19 → total string)

code = a = 01 = 2

b = 110 = 3

Total bits = (frequency of ';') * length of code for ;) + (frequency of 'a') * length of code for a) + ------

= 6×2 + 4×2 + 4×2 + 3×3 + 4×1 + 4×1 = 45 bits.

Average code length = Total no of bits / Total no of symbol

= 45 / 19 = 2.37.

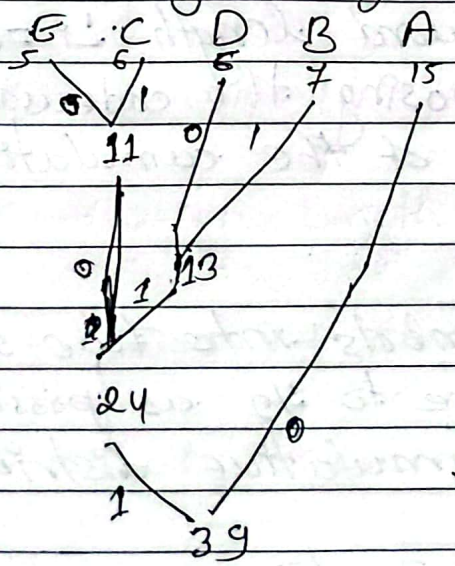
∴ Average code length is 2.37 bits per symbol as compared to 8-bit per symbol before encoding.

Example-2 ↓

symbol	A	B	C	D	E
count	15	7	6	6	5

Sol^n

Sorting all symbol first and use huffman coding.



⇒ A = 0 = 1

B = 111 = 3

C = 101 = 3

D = 110 = 3

E = 100 = 3

∴ Total bit length = 1×15 + 3×7 + 3×6 + 3×6 + 3×5

= 87

Average bit length = 87 / 39 = 2.231 bits

Entrophy $(H) = \sum_{i=1}^{n} P_i \log(1/n)$.

or $- \sum_{i=1}^{n} P_i \log(P_i)$

eg Hello $\Rightarrow$ $H = 1/5$, $e = 1/5$, $l = 2/5$, $o = 1/5$

$\therefore H = -(1/5)\log_2(1/5) - (1/5)\log_2(1/5) - (2/5\log_2 2/5) - (1/5)\log_2(1/5)$.

$\Rightarrow H = \ldots$

$\Rightarrow$ Shannon-Fano coding:-

- Related to field of data compression.
- Cloud shannon and Robert Fano - scientist
- Related techniques for constructing a prefix code based on a set of symbols and their probabilities. (estimated or measured).

$\rightarrow$ Shannon's method:-

- Chooses a prefix code where a source symbol 'i' is given the codeword length $L_i = \lceil -\log_2 P_i \rceil$
- one common way of choosing the codewords uses the binary expansions of the cumulative probabilities.

$\rightarrow$ Fano's method:-

- Divides the source symbols into two sets (0 & 1) with probabilities as close to 1/2 as possible.
- It is based on the cumulative distribution function.
- It provides low optimization.

Example :-

m.	a	b	c	d	e
P	0.3	0.15	0.15	0.2	0.2

Sol:-

	0.3	0.15	0.15	0.2	0.3
0.3 (a)	0	0			
0.2 (d)	0	1			
0.2 (e)	1	0			
0.15 (b)	1	1	0		
0.15 (c)	1	1	1		

Example :- Applying Shannon Fano coding for following message.

$[m] = [\, m_1 \quad m_2 \quad m_3 \quad m_4 \quad m_5 \quad m_6 \,]$

$[P] = [\, 0.30 \quad 0.25 \quad 0.15 \quad 0.12 \quad 0.08 \quad 0.10 \,]$

Sol:-

m	p		stage 1	stage 2	stage 3	codeword	length
m_1	0.30	0.55	0	0		00	2
m_2	0.25		0	1		01	2
m_3	0.15	0.45	1	0	0	100	3
m_4	0.12		1	0	1	101	3
m_6	0.10		1	1	0	110	3
m_5	0.08		1	1	1	111	3

Average length $L = \sum_{K=1}^{n} p_n h_K = \sum_{K=1}^{6} p_6 h_K$

$= (0.30 \times 2) + (0.25 \times 2) + (0.15 \times 3) + (0.12 \times 3) + (0.10 \times 3) + (0.08 \times 3) = 2.48$ bits

$\therefore \Rightarrow L = 2.45$ bits

⇒ Job scheduling with deadlines:-

- It has unique processor machine.
- Each job 'Ji' requires one unit time to complete.
- Each job 'Ji' has di deadlines and P_i - profits.
- Ji is - associated with P_i iff it meets its deadline
- Find the subset of jobs and its sequence, so that its profit is maximum.
- A machine can provide service to any one job at a time.
- The greedy approach to this problem involves sorting the job by their profits. in descending order and then scheduling them in the order in which they appear in the sorted list, as long as their deadlines have not been exceeded.

Algorithm:-

- Arrange all job J_i in decending order with respect to P_i.
- Allocate timeslot with respect to maximum deadlines.
- k = min (max (deadline), di)
- Allocate the time slot for J_i with respect to value of k.
- If the time slot is already occupied then search for the positions before it.
- If all the time slot is occupied publish the set and sequence and max profit.

• It has uniprocessor, non-preemption, and every job takes one unit of time.

n = maximum deadline & m = total job.

∴ total time = n x m.

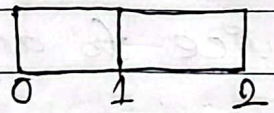
Example-1

	1	2	3	4
Job (J_i)	J_1	J_2	J_3	J_4
Profit (P_i)	100	10	15	17
deadline (d_i)	2	1	2	1

Soln

Time slot = max deadline → max deadline = 2

Since time slot = 2.

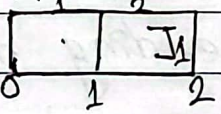


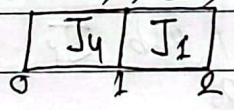
descending order is:

J_i	J_1	J_4	J_3	J_2
P_i	100	27	15	10
d_i	2	1	2	1

Iteration-1 $i=1$

$K = \min(\max(\text{deadline}), d_i) = \min(2,2) = 2$.

① 

② 

$\Rightarrow$ set = $\{J_1, J_4\}$

Sequen = $\{J_4, J_1\}$ $\Rightarrow$ profit max = 27+100 = 127 #

Iteration-2 : $i=2$

$k = \min(\max(\text{deadline}), d_i) = \min(2,1) = 1$

Example :

J_i	J_1	J_2	J_3	J_4
P_i	30	15	75	80
d_i	1	2	3	2

Soln descending order - sorting:

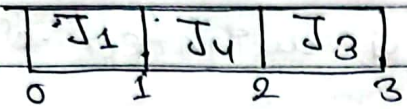
J_i	J_4	J_3	J_1	J_2
P_i	80	75	30	15
d_i	2	3	1	2

Total time slot = 3

for i=1

k = min (max (deadline), di) = max(3,2) = 2

⇒ J4 in slot 2



⇒ 30 + 80 + 75 = 185 # profit max.

for i=2

k = min (max (deadline), di) = min (3,3) = 3 #

for i=3

k = min (max (deadline), di) = min (3,1) = 1 #

⇒ set = { J1, J3, J4 }

sequence = { J1, J4, J3 }

profit max = 30 + 80 + 75 = 185 #

⇒ Analysis :-

Case-I :- Each time slot calculated is vaccant, so just insert is required which is equal to the no of insertion. ∴ O(n) = n. #

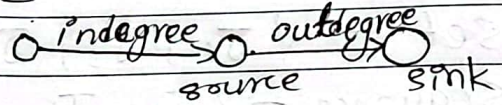
Case-II :- Time slot calculated is not vacant, so required another iteration to find the vacant slot. ∴ O(n) = n(n-1) ⇒ n^2 #

Algorithm

```
Greedy Job (d, J, n) {
    J = {1}
    for i = 2 to n do {
        if all deadlines in J ∪ {i} are feasible then
            J = J ∪ {i}
    }
    return J
}
```

⇒ Tree vertex splitting - (TVS) -:

- Consider directed binary tree each edge is leveled with weights.
- TVS can be defined as:
  Let $T = (V, E, W)$ be a weighted directed tree. .
  where,
  $V \rightarrow$ set of vertices
  $E \rightarrow$ set of edges
  $W(i,j) \rightarrow$ weight of edge $\langle i,j \rangle \in E$. and
- Source vertex has in-degree '0' and sink vertex has out degree.



- we define delay of path 'p' denoted as $d(P)$ = sum of weight of path.
  ⇒ Delay of Tree, $d(T) = \max(d(P))$ ∀
- Let $T = (V, E, W)$ be a directed tree. A weighted tree can be used to model a distribution network in which electrical signal are transmitted.
- The transmission of power from node to another may result in some loss.
- Each edge in the tree is labeled with the loss that occurs in traversing that edge.
- The network may not be able to tolerate losses beyond a critical limit.
- In places where the loss exceeds the tolerance level, boosters have to be placed.
- Given a network and a loss tolerance level, the tree vertex splitting problem is to

determine an optimal placement of boosters.

- A greedy approach to solve this problem is to compute for each node $u \in V$, the maximum delay $d(u)$ from u to any other node in its subtree.
- If u has a parent v such that
$d(u) = \max\{d(v) + w(parent(u), v)\}$
$d(u) \rightarrow$ delay of node.
v - set of all edges & u belongs to child(u)
δ tolerance value.

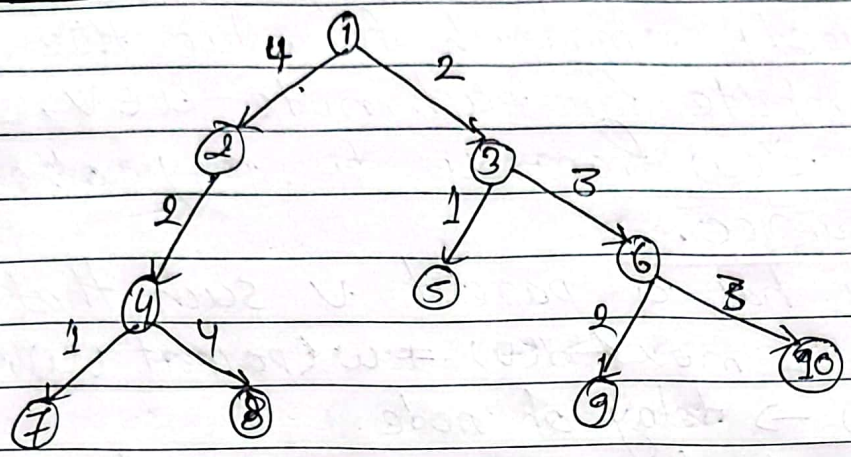
- **Algorithm -:**

```
Algorithm TVS(T, δ){
    if (T ≠ 0) then
    {
        d[T] = 0;
        for each child v to T do.
        {
            TVS(v, δ);
        d[T] = max{d[T], d[v] + W[T, v]};
    }
    if ((T is not the root) and (d[T] +
            W(parent(t), T) → δ)) then.
    {
        write [T];
        d[T] = 0;
    }}}
```

Analysis -:

- visit each vertex n;
- In each vertex there is constant time required for calculation and computing it with δ. ∴ $O(n) = n$ #

Example:-



If $d(u) >= \delta$; then place the booster.

Sol:

Here all leaf nodes delay = 0 and leaf nodes are 7, 8, 9, 10, 5

$\delta = 5$

$d[7] = \max\{0 + w(4,7)\} = 1$

$d[8] = \max\{0 + w(4,8)\} = 4$

$d[9] = \max\{0 + w(6,9)\} = 2$

$d[10] = \max\{0 + w(6,10)\} = 3$

$d[5] = \max\{0 + w(3,5)\} = 1$

$\Rightarrow d[7], d[8], d[9], d[10], d[5] < \delta \Rightarrow$ 0 ✓

$d[4] = \max\{1 + w(2,4) + 4 + w(2,4)\}$

$= \max\{1+2, 4+2\} = 6 > \delta$ booster ✓

$d[6] = \max\{2 + w(3,6) + 3 + w(3,6)\}$

$= \max\{2+3, 3+3\} = 6 > \delta$ booster ✓

$d[2] = \max\{6 + w(1,2)\}$

$= \max\{6+4\} = 10 > \delta$ booster ✓

$d[3] = \max\{1 + w(1,3), 6 + w(1,3)\}$

$= \max\{3, 8\} = 8 > \delta$ booster ✓

∴ No need to find tolerance value for node 1 because from source only power is transmitting.

⇒ Dynamic Programming :-

- It is a technique in compute programming used to solve optimized problems by breaking down a large problem into smaller problems and solving each sub-problem only once.
- principle of optimal substructure which means that the optimal solution to a large problem can be found by combining the optimal solutions to smaller sub-problems.
- Dynamic programming is efficient in finding optimal solutions for cases with lots of overlapping sub-problems.
- It solves problems by recombining solutions to sub-problems, when the sub-problems themselves may share sub-sub-problems.

eg. fibonacci number -

1, 1, 2, 3, 5, 8, 13, 21, 34, - - -.

Sol^n

```
def fib(n):
    if n==0 or n==1:
        return n
    else:
        return fib(n-1) + fib(n-2)
```

- Dynamic programming is an algorithm design method that can be used when the solution to a problem can be viewed as the result of a sequence of decision.

⇒ Difference between Greedy vs Dynamic Algorithm :-

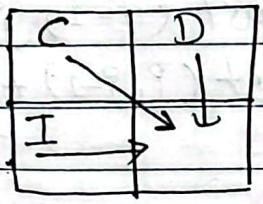
Greedy Algorithm	Dynamic Algorithm
→ makes locally optimal choices at each step	→ Breaks down a problem in smaller subproblems.

CamScanner

→ Does not guarantee an optimal solution.	→ Guarantee an optimal solution.
→ ~~Does~~ Faster than dynamic programming.	→ slower than greedy programming.
→ may not consider all possible choices	→ consider all possible choices.
→ Works well when the problem ~~has~~ the greedy choice property.	→ can solve a wide range of a problems.
→ memory efficient as it only needs to store the current solution.	→ memory intensive, as it needs to store solution to a sub-problems.
Example: Huffman coding, kruskal's algorithm etc	example: knapsack problem, fibonacci sequence. etc.

⇒ String Editing :-

- There are two string source and destination. The goal of the problem is to convert source to destination by applying minimum edit operations on string 'source'.
- we have given two string $x = x_1, x_2, x_3, \dots, x_n$ and $y = y_1, y_2, y_3, \dots, y_m$; where $x_i = 1 \le i \le n$ and $y_j = 1 \le j \le m$ are number of finite set of symbols known as the alphabets, we want's to transform x to y using a sequence of edit operation on x.
- The permissible edit operations are:
  Delete — 1 cost → denote D(x_i)
  change/copy — 2 cost → denote C(x_i y_j)
  Insert — 1 cost → denote I(y_j)
- The problem of string editing is to identify a minimum cost sequence of edit operations that will transform x into y.



Example :-

String-1 → a a b a b (source)
String-2 → b a b b (destination)

Total cost = 5(source) + 4(destination) = 9 #

Technique-1 : Remove all characters within string-1, Insert the required string:

d d d d d → b → a → b → b
D D D D D I I I I ⇒ cost = 9 #

Technique-2:

a ∅ a b a b → b → First Insert

D D I ⇒ cost = 3

Technique-3:

a b a b ∅ b ← Replace with b

R D ⇒ cost = 2+1 = 3 #

Technique-4:

∅ b a b ∅ b b

R R D ⇒ cost = 5 #

Technique-5:

b ∅ a b ∅ b

I D D ⇒ cost = 3

$$cost(i,j) = \begin{cases} 0 & i=j=0 \\ cost(i-1,0) + D(x_i) & i>0;\ j=0 \\ cost(0,j-1) + I(y_j) & i=0;\ j>0 \end{cases}$$

or

$$cost(i,j) = \min \begin{cases} cost(i-1,j) + D(x_i) & i>0 \\ cost(i-1,j-1) + c(x_i,y_j) & i>0,\ j>0 \\ cost(i,j-1) + I(y_j) & j>0 \end{cases}$$

Example:

Source = a a b a b

destination = b a b b

Sol^n

i \ j	Null 0	a_1	a_2	b_3	b_4
Null 0	0 →	1	2	3	4
b_1	1 →	2	1 →	2	3
b_2	2	3	2 →	3	4
b_3	3	2	3	2 →	3
b_4	4	3	2	3	4
b_5	5	4	3	2 →	3

Step-1 :- Case · i=0 ; j=0 , cost (0,0) = 0

Case · i > 1 , j = 0

cost (0,1) = cost (0, 1-1) + I (y1) = 0+1 = 1

cost (0,2) = cost (0, 2-1) + I (y2) = 1+1 = 2

cost (0,3) = cost (0, 3-1) + I (y3) = 2+1 = 3

cost (0,4) = cost (0,4-1) + I (y4) = 3+1 = 4

Step2

cost (1,0) = cost (1-1, 0) + D (x1) = 0+1 = 1

cost (2,0) = cost (2-1, 0) + D (x2) = 1+1 = 2

cost (3,0) = cost (3-1, 0) + D (x3) = 2+1 = 3

cost (4,0) = cost (4-1, 0) + D (x4) = 3+1 = 4

cost (5,0) = cost (5-1, 0) + D (x5) = 4+1 = 5

Row-1

$$cost(1,1) = min \begin{cases} cost(1-1,1) + D(x_1) \\ cost(1-1,1-1) + c(x_1,y_1) \rightarrow \text{change cost} \\ cost(1,1-1) + I(y_1) \end{cases}$$

= min { 1+1, 0+2, 1+1 } = 2 #

$$cost(1,2) = min \begin{cases} cost(1-1,2) + D(x_1) \\ cost(1-1,2-1) + c(x_1,y_2) \\ cost(1,2-1) + I(y_2) \end{cases}$$

= min { 2+1, 1+0, 2+1 } = 1 #

$$cost(1,3) = min \begin{cases} cost(1-1,3) + D(x_1) \\ cost(1-1,3-1) + c(x_1,y_3) \\ cost(1,3-1) + I(y_3) \end{cases}$$

= min { 3+1, 2+2, 1+1 } = 2 #

$$cost(1,4) = min \begin{cases} cost(1-1,4) + D(x_1) \\ cost(1-1,4-1) + c(x_1,y_4) \\ cost(1,4-1) + I(y_4) \end{cases}$$

= min { 4+1, 3+2, 2+1 } = 3 #

Row-2 $cost(2,1) = \min \begin{cases} cost(2-1,1) + D(x_2) \\ cost(2-1,1-1) + C(x_2, y_1) \\ cost(2,1-1) + I(y_1) \end{cases}$

$\min\{2+1, 1+2, 2+1\} = 3$ #

$cost(2,2) = \min \begin{cases} cost(2-1,2) + D(x_2) \\ cost(2-1,2-1) + C(x_2, y_2) \\ cost(2,2-1) + I(y_2) \end{cases}$

$= \min\{1+1, 2+0, 3+1\} = 2$ #

$cost(2,3) = \min \begin{cases} cost(2-1,3) + D(x_2) \\ cost(2-1,3-1) + C(x_2, y_3) \\ cost(2,3-1) + I(y_3) \end{cases}$

$= \min\{2+1, 1+2, 2+1\} = 3$ #

$cost(2,4) = \min \begin{cases} cost(2-1,4) + D(x_2) \\ cost(2-1,4-1) + C(x_2, y_4) \\ cost(2,4-1) + I(y_4) \end{cases}$

$= \min\{4+1, 2+2, 3+1\} = 4$ #

Row-3 $cost(3,1) = \begin{cases} cost(3-1,1) + D(x_3) \\ cost(3-1,1-1) + C(x_3, y_1) \\ cost(3,1-1) + I(y_1) \end{cases}$

$= \min\{3+1, 2+0, 3+1\} = 2$ #

$cost(3,2) = \min \begin{cases} cost(3-1,2) + D(x_3) \\ cost(3-1,2-1) + C(x_3, y_2) \\ cost(3,2-1) + I(y_2) \end{cases}$

$= \min\{2+1, 3+2, 2+1\} = 3$ #

$$cost(3,3) = \min \begin{cases} cost(3-1, 3) + D(x_3) \\ cost(3-1, 3-1) + c(x_3, y_3) \\ cost(3, 3-1) + c(y_3) \end{cases}$$

$$= \min \{ 3+1, 2+0, 3+1 \} = 2 \#$$

$$cost(3,4) = \min \begin{cases} cost(3-1, 4) + D(x_3) \\ cost(3-1, 4-1) + c(x_3, y_4) \\ cost(3, 4-1) + I(y_4) \end{cases}$$

$$= \min \{ 4+1, 3+0, 2+1 \} = 3 \#$$

Row-4

$$cost(4,1) = \min \begin{cases} cost(4-1, 1) + D(x_4, y_1) \\ cost(4-1, 1-1) + c(x_4, y_1) \\ cost(4, 1-1) + I(y_1) \end{cases}$$

$$= \min \{ 2+1, 3+2, 4+1 \} = 3 \#$$

$$cost(4,2) = \min \begin{cases} cost(4-1, 2) + D(x_4) \\ cost(4-1, 2-1) + c(x_4, y_2) \\ cost(4, 2-1) + I(y_2) \end{cases}$$

$$= \min \{ 3+1, 2+0, 3+1 \} = 2 \#$$

$$cost(4,3) = \min \begin{cases} cost(4-1, 3) + D(x_4) \\ cost(4-1, 3-1) + c(x_4, y_3) \\ cost(4, 3-1) + I(y_3) \end{cases}$$

$$= \min \{ 2+1, 3+2, 2+1 \} = 3 \#$$

$$cost(4,4) = \min \begin{cases} cost(4-1, 4) + D(x_4) \\ cost(4-1, 4-1) + c(x_4, y_4) \\ cost(4, 4-1) + I(y_4) \end{cases}$$

$$= \min \{ 3+1, 2+2, 3+1 \} = 4 \#$$

Row 5

$$\text{Cost}(5,1) = \min \begin{cases} \text{cost}(5-1,1) + D(x_5) \\ \text{cost}(5-1,1-1) + C(x_5, y_1) \\ \text{cost}(5,1-1) + I(y_1) \end{cases}$$

$= \min \{ 3+1, 4+0, 5+1 \} = 4$ #

$$\text{cost}(5,2) = \min \begin{cases} \text{cost}(5-1,2) + D(x_5) \\ \text{cost}(5-1,2-1) + C(x_5, y_2) \\ \text{cost}(5,2-1) + I(y_2) \end{cases}$$

$= \min \{ 2+1, 3+0, 4+1 \} = 3$ #

$$\text{cost}(5,3) = \min \begin{cases} \text{cost}(5-1,3) + D(x_5) \\ \text{cost}(5-1,3-1) + C(x_5, y_3) \\ \text{cost}(5,3-1) + I(y_3) \end{cases}$$

$= \min \{ 3+1, 2+0, 3+1 \} = 2$ #

$$\text{cost}(5,4) = \min \begin{cases} \text{cost}(5-1,4) + D(x_5) \\ \text{cost}(5-1,4-1) + C(x_5, y_4) \\ \text{cost}(5,4-1) + I(y_4) \end{cases}$$

$= \min \{ 4+1, 3+0, 2+1 \} = 3$ #

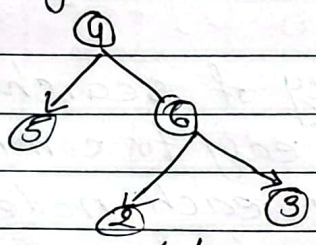
= minimum cost $= 1+1+0+0+0+1 = 3$

optimal cost (3) #

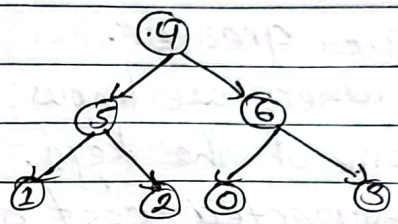
⇒ Optimal Binary Search Tree - (OBST) -:

1) Incomplete Binary Tree
2) Complete Binary Tree
3) Left skewed
4) Right skewed.

⇒ Binary Tree (BT) -:

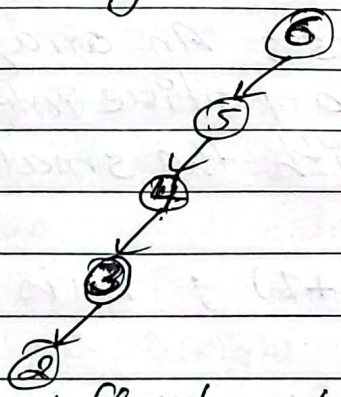


Eg: Incomplete Binary Tree

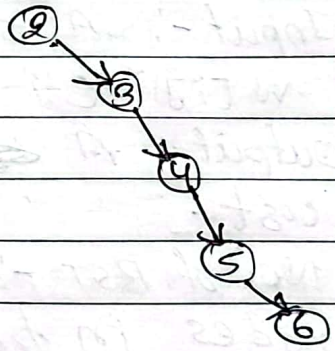


Eg: Complete Binary Tree.

⇒ Binary Search Tree -:



Eg Left skewed BST



Eg Right skewed BST.

Binary Tree	Binary Search Tree
- A tree data structure in which each node has at most two children without any ordering restrictions.	- A tree data structure in which each node follows a specific ordering rule, left subtree contain only nodes with keys less than parent nodes & right subtree contains only node with key generate then the parent node.
- Ordering of node not required.	- ordering of nodes are sorted.
- Efficiency : O(log n)	- Efficiency : O(log n).

Definition:-

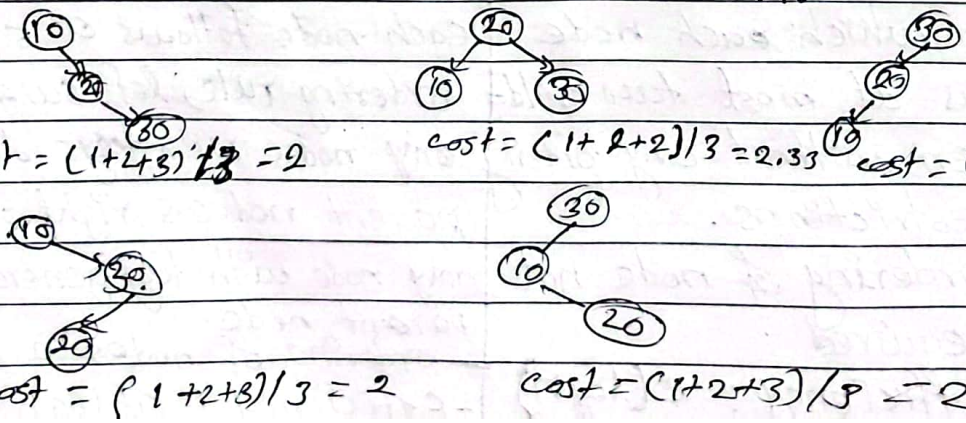
- Binary search Tree (BST) is a tree where the key values are stored in the internal nodes. The external nodes are null node. The keys are stored lexicographically, i.e. for each internal node all the key in the length sub-tree are less than the keys in the node, and all the keys in the right sub-tree are greater.
- When we know the frequency of searching each one of the keys, it is quite easy to compute the expected cost of accessing each node in the tree. An optimal binary search tree is a BST, which has minimal expected cost of locating each node.
- Input:- A set of n integers; An array W where; W[i] (1 ≤ i ≤ n) stores a positive integer weight.
- Output:- A BST on S with the smallest average cost.
- No. of BST :- = $^{2n}C_n / (n+1)$; n is no of nodes in tree.

  eg $n=4 \Rightarrow \frac{8!}{4!5!} = 14$ #

Example:- given node: 10, 20, 30

The possible BST are:

for $n=3 \Rightarrow$ no of BST $= \frac{6!}{3!4!} = 5$ #



Example :-

X :	10	20	30	40
freq" :	4	2	6	3

Soln solving with dynamic programming :

i ↓ j →	0	1	2	3	4
0	0	$4^{(1)}$	$8^{(1)}$	$20^{(3)}$	$26^{(3)}$
1		0	$2^{(1)}$	$10^{(3)}$	$16^{(3)}$
2			0	$6^{(1)}$	$12^{(3)}$
3				0	$3^{(1)}$
4					0

Case I : $j-i=0$

$0-0=0 \Rightarrow c[0,0]=0$

$1-1=0 \Rightarrow c[1,1]=0$

$2-2=0 \Rightarrow c[2,2]=0$

$3-3=0 \Rightarrow c[3,3]=0$

$4-4=0 \Rightarrow c[4,4]=0$

Case II : $j-i=1$

$1-0=1 \Rightarrow c[0,1]=10 \Rightarrow 4$

$2-1=1 \Rightarrow c[1,2]=20 \Rightarrow 2$

$3-2=1 \Rightarrow c[2,3]=30 \Rightarrow 6$

$4-3=1 \Rightarrow c[3,4]=40 \Rightarrow 3$

now, cost calculate ;.

$$\cdot\ c[i,j] = \min_{k=i+1}^{j} \{c[i,k-1]+c[k,j]\} + w[i,j]$$

where weight $w[i,j] = \sum_{i=1}^{n} f(i)$.

then,

Case III $j-i=2$

$2-0=2 \Rightarrow c[0,2] = \min_{k=1,2} \begin{Bmatrix} c[0,0]+c[1,2], \\ c[0,1]+c[2,2] \end{Bmatrix} + w[0,2]$

$= \min \begin{Bmatrix} 0+2, \\ 4+0 \end{Bmatrix} + 4+2 = 2+4+2 = 8^{(1)}$ #

$3-1=2 \Rightarrow c[1,3] = \min_{k=2,3} \begin{Bmatrix} c[1,1]+c[2,3], \\ c[1,2]+c[3,3] \end{Bmatrix} + w[1,3]$

$= \min \begin{Bmatrix} 0+6, \\ 2+0 \end{Bmatrix} + 2+6 = 2+2+6 = 10^{(3)}$ #

$4-2=2 \Rightarrow c[2,4] = \min_{k=3,4} \{ c[2,2]+c[3,4],\ c[2,3]+c[4,4] \} + w[2,4]$

$= \min \{ 0+3,\ 6+0 \} + 6+3 = 3+6+3 = 12^{(3)}$ #

Case IV : $j-i=3$

$3-0=3 \Rightarrow c[0,3] = \min_{k=1,2,3} \{ c[0,0]+c[1,3],\ c[0,1]+c[2,3],\ c[0,2]+c[3,3] \} + w[0,3]$

$= \min \{ 0+10,\ 4+6,\ 8+0 \} + 4+2+6 = 8+4+2+6 = 20^{(3)}$ #

$4-1=3 \Rightarrow c[1,4] = \min_{k=2,3,4} \{ c[1,1]+c[2,4],\ c[1,2]+c[3,4],\ c[1,3]+c[4,4] \} + w[1,4]$

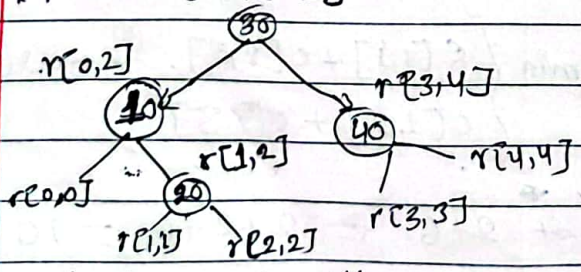
$= \min \{ 0+12,\ 2+3,\ 10+0 \} + 2+6+3 = 5+2+6+3 = 16^{(3)}$ #

Case-V : $j-i=4$

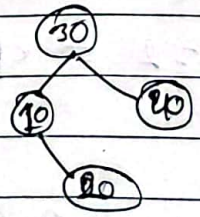
$4-0=4 \Rightarrow c[0,4] = \min_{k=1,2,3,4} \{ c[0,0]+c[1,4],\ c[0,1]+c[2,4],\ c[0,2]+c[3,4],\ c[0,3]+c[4,4] \} + w[0,4]$

$= \min \{ 0+16,\ 4+12,\ 8+3,\ 20+0 \} + 4+2+6+3 = 11+4+2+6+3 = 26^{(3)}$ #

$\therefore\ r[0,4] = 3$



$\Rightarrow$



#

Analysis : Algorithm requires $O(n^3)$ time, since we have three (3) level of nested for loop, each of these loops take an at most n value #

Example :- Optimal Binary Search Tree (OBST) :-

x_i	—	10	20	30	40
f_i	—	4	2	6	3
p_i	—	3	3	1	1
q_i	2	3	1	1	1

Sol^n Now we construct value of cost, weight and root and fill the following table :

$w_{0,0} = 2$	$w_{1,1} = 3$	$w_{2,2} = 1$	$w_{3,3} = 1$	$w_{4,4} = 1$
$c_{0,0} = 0$	$c_{1,1} = 0$	$c_{2,2} = 0$	$c_{3,3} = 0$	$c_{4,4} = 0$
$r_{0,0} = 0$	$r_{1,1} = 0$	$r_{2,2} = 0$	$r_{3,3} = 0$	$r_{4,4} = 0$
$w_{0,1} = 8$	$w_{1,2} = 7$	$w_{2,3} = 3$	$w_{3,4} = 3$	
$c_{0,1} = 8$	$c_{1,2} = 7$	$c_{2,3} = 3$	$c_{3,4} = 3$	
$r_{0,1} = 1$	$r_{1,2} = 2$	$r_{2,3} = 3$	$r_{3,4} = 4$	
$w_{0,2} = 12$	$w_{1,3} = 9$	$w_{2,4} = 5$		
$c_{0,2} = 19$	$c_{1,3} = 12$	$c_{2,4} = 8$		
$r_{0,2} = 1$	$r_{1,3} = 2$	$r_{2,4} = 3$		
$w_{0,3} = 14$	$w_{1,4} = 11$			
$c_{0,3} = 25$	$c_{1,4} = 19$			
$r_{0,3} = 2$	$r_{1,4} = 2$			
$w_{0,4} = 16$				
$c_{0,4} = 32$				
$r_{0,4} = 2$				

$$c[i,j] = \min_{i<k\le j} \{ c[i,k-1] + c[k,j] \} + w[i,j]$$

and weight $w[i,j] = w[i,j-1] + p_j + q_j$

$w[0,0], w[1,1], w[2,2], w[3,3], w[4,4]$ = Initial value of q_i = 2, 3, 1, 1, 1, then root all zero. and initial cost is also all zero.

$j-i=0 \Rightarrow c[0,0], c[1,1], c[2,2], c[3,3], c[4,4] = 0$

$j-i=1 \Rightarrow c[0,1], c[1,2], c[2,3], c[3,4]$

$j-i=2 \Rightarrow c[0,2], c[1,3], c[2,4]$

$j-i=3 \Rightarrow c[0,3], c[1,4]$

$j-i=4 \Rightarrow c[0,4]$.

Case-I $j-i=0$

$w[0,0] = q_0 + \cdots = 2 \quad c[0,0]=0 \quad \& \quad r[0,0]=0$

$w[1,1] = q_1 + \cdots = 3 \quad c[1,1]=0 \quad \& \quad r[1,1]=0$

$w[2,2] = q_2 + \cdots = 1 \quad c[2,2]=0 \quad \& \quad r[2,2]=0$

$w[3,3] = q_3 + \cdots = 1 \quad c[3,3]=0 \quad \& \quad r[3,3]=0$

$w[4,4] = q_4 + \cdots = 1 \quad c[4,4]=0 \quad \& \quad r[4,4]=0$

Case-II $j-i=1$

$w[0,1] = w[0,0] + \overset{p_1+q_1}{3+3} \Rightarrow 2+3+3 = 8$

$w[1,2] = w[1,1] + 3+1 \Rightarrow 3+3+1 = 7$

$w[2,3] = w[2,2] + 1+1 \Rightarrow 1+1+1 = 3$

$w[3,4] = w[3,3] + 1+1 \Rightarrow 1+1+1 = 3$ and

$c[0,1] = \min_{k=1} \{c[0,0] + c[1,1]\} + w[0,1] = \min\{0+0\}+8 = 8^{(1)}$

$c[1,2] = \min_{k=2} \{c[1,1] + c[2,2]\} + w[1,2] \Rightarrow \min\{0+0\}+7 = 7^{(2)}$

$c[2,3] = \min_{k=3} \{c[2,2] + c[3,3]\} + w[2,3] \Rightarrow \min\{0+0\}+3 = 3^{(3)}$

$c[3,4] = \min_{k=4} \{c[3,3] + c[4,4] + w[3,4] \Rightarrow \min\{0+0\}+3 = 3^{(4)}$

Case-III $j-i=2$

$w[0,2] = w[0,1] + P_2 + q_2 \Rightarrow 8+3+1 = 12$

$w[1,3] = w[1,2] + P_3 + q_3 \Rightarrow 7+1+1 = 9$

$w[2,4] = w[2,3] + P_4 + q_4 \Rightarrow 3+1+1 = 5$

$$c[0,2] = \min_{k=1,2} \begin{Bmatrix} c[0,0]+c[1,2], \\ c[0,1]+c[2,2] \end{Bmatrix} + w[0,2]$$

$$= \min \begin{Bmatrix} 0+7 \\ 8+0 \end{Bmatrix} + 12 = 7+12 = 19 \text{ and}$$

$r_{0,2} = 1$

$$c[1,3] = \min_{1<k\le 2,3} \{ c[1,1] + c[2,3], \; c[1,2] + c[3,3] \} + w[1,3]$$

$$= \min \{ 0+3, \; 7+0 \} + 9 \Rightarrow 3+9 = 12 \quad \& \quad r_{1,3} = 2$$

$$c[2,4] = \min_{1<k\le 3,4} \{ c[2,2] + c[3,4], \; c[2,3] + c[4,4] \} + w[2,4]$$

$$= \min \{ 0+3, \; 3+0 \} + 5 \Rightarrow 3+5 = 8 \quad \& \quad r_{2,4} = 3 \text{ or } 4$$

Case IV $j - i = 3$

$$w[0,3] = w[0,2] + P_3 + q_3 = 12+1+1 = 14$$

$$w[1,4] = w[1,3] + P_4 + q_4 = 9+1+1 = 11$$

$$c[0,3] = \min_{1<k\le 1,2,3} \{ c[0,0] + c[1,3], \; c[0,1] + c[2,3], \; c[0,2] + c[3,3] \} + w[0,3]$$

$$= \min \{ 0+12, \; 8+3, \; 19+0 \} + 14 = 11 + 14 = 25 \quad \& \quad r_{0,3} = 2$$

$$c[1,4] = \min_{1<k\le 2,3,4} \{ c[1,1] + c[2,4], \; c[1,2] + c[3,4], \; c[1,3] + c[4,4] \} + w[1,4]$$

$$= \min \{ 0+8, \; 7+3, \; 12+0 \} + 11 = 8 + 11 = 19 \quad \& \quad r_{1,4} = 2$$

Case V $j - i = 4$

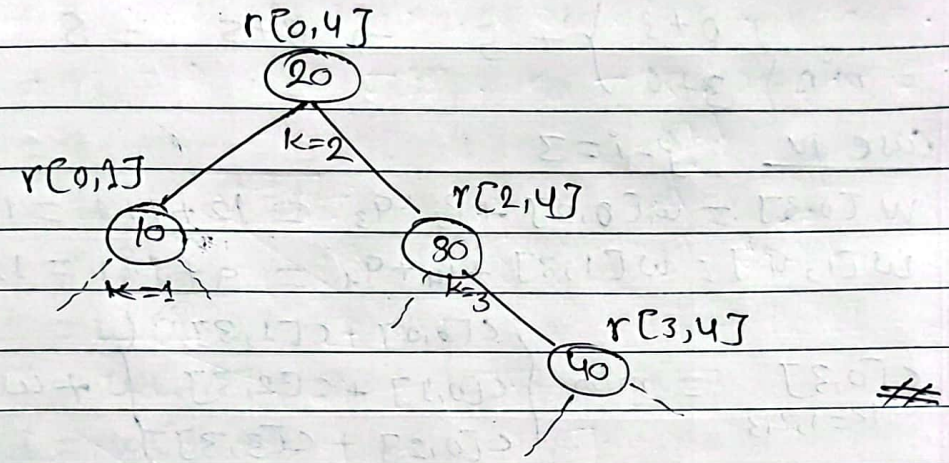
$$w[0,4] = w[0,3] + P_4 + q_4 \Rightarrow 14+1+1 = 16$$

$$\therefore c[0,4] = \min \{ c[0,0] + c[1,4], \; c[0,1] + c[2,4], \; c[0,2] + c[3,4], \; c[0,3] + c[4,4] \} + w[0,4].$$

$$= \min \begin{Bmatrix} 0+19, \\ 8+8, \\ 19+3, \\ 25+0 \end{Bmatrix} +16 \quad = 16+16 = 32 \;\&\; r_{0,4} = 2$$

$\Rightarrow$ minimum cost $= 32/16 = 2$ #

and OBST root is 2 or $r_{0,4} = 2$ then,



⇒ **Backtracking -:**

- Backtracking is a general algorithmic technique that involves exploring all possible solutions to a problem by incrementally building solution and backtracking when we determine that it cannot be complete further.
- It is used in situations, where we need to explore a large, but finite, search space to find a solution.
- It is particularly useful when the search space is too large to explore exhaustively by using heuristics ~~too large to explore e~~ eliminate paths that can't lead to valid solution.

Forward
- General solution vector one at a time.
- Add other feasible alternatives to the vector until the criteria is met.

Backward
- The criteria is not meet traverse backward to immediate node & change the direction.

- The concept of coming backward is also known as pruning.

Algorithm -:

1) Sum of subsets
2) knapsack problem.

⇒ **Sum of Subsets -:**

- The sum of subset problem is a combinational optimization problem that involves finding all possible subset of a given set of elements whose

sum equal a target value. this problem uses solved by using backtracking method.:

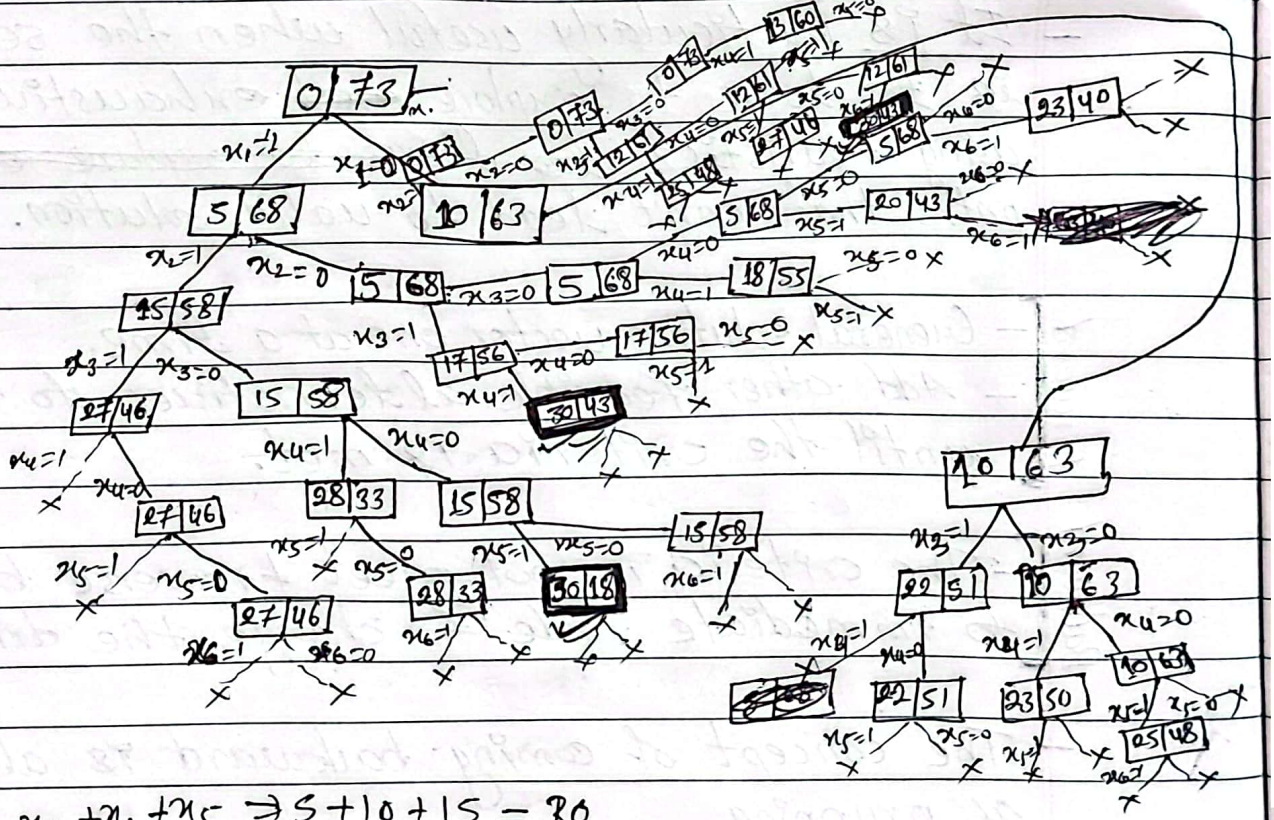
Example :- $w[1:6] = \{5, 10, 12, 13, 15, 18\}$

$n = 6$ and Sum of subset $m = 30$.

Sol

we make total path $2^n = 2^6 = 64$ possible construct tree using backtracking method.

$x_1 = 5$, $x_2 = 10$, $x_3 = 12$, $x_4 = 13$, $x_5 = 15$ & $x_6 = 18$



$\Rightarrow x_1 + x_2 + x_5 \Rightarrow 5 + 10 + 15 = 30$

$x_1 + x_3 + x_4 \Rightarrow 5 + 12 + 13 = 30$

$x_3 + x_6 \Rightarrow 12 + 18 = 30$ //

$$\sum_{i=1}^{k} w_i x_i + w_{k+1} \leq m$$

$$= \sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i > m$$

⇒ **Knapsack Algorithm-:**

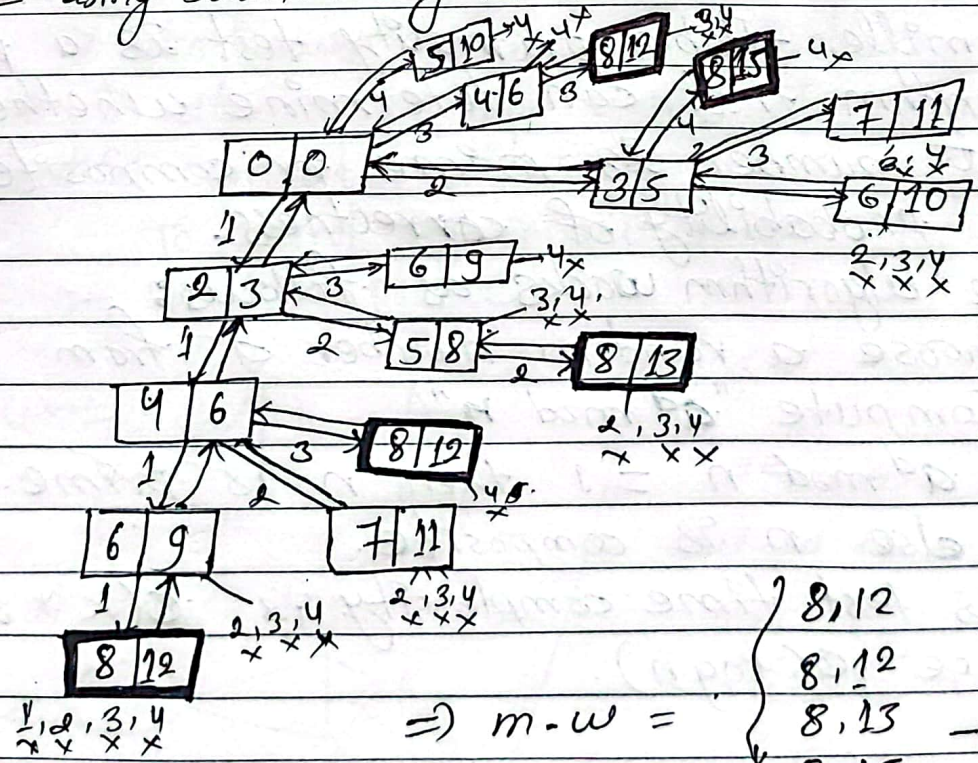
- The knapsack problem is a type of optimization problem where a set of items with different weights and values need to be selected to fill a knapsack of limited capacity.
- The objective is to maximize the total value of the selected items while not exceeding the capacity of the knapsack.
- This knapsack problem using technique is back-tracking.

Example-:

i	1	2	3	4	
w	2	3	4	5	kg
p	3	5	6	10	

max weight $m = 8$ kg.

Sol^n using backtracking on tree;



⇒ m - w = 8, 12 / 8, 12 / 8, 13 / 8, 15 / 8, 12

→ 2, 4 ⇒ 0 1 0 1

⇒ 3 + 5 = 8

∴ p = 5 + 10 = 15 max

Analysis-:

$O(2^n)$ worstcase

⇒ Randomized Algorithm -:

1. Average cost is minimum than other algorithm
2. It might not provide the correct output.
   - To table point 2, we should embed some deterministic approach within Randomization.

eg. • monte Carlo.
• Las Vegas
   - primality Test.
      ↳ miller Robins Algorithm.

⇒ Primality Testing -:

- To perform primality testing using randomized algorithm is called the miller-Robin primality.
- The miller-Robin primality test is a probabilistic algorithm that can determine whether an input number is prime or composite with a high probability of correctness.
- The algorithm works as follows:
  • Choose a random number a from n.
  • Compute "a^d mod n"
  • If $a^d \bmod n = 1$ then n is prime.
    else n is composite.

This test time complexity is $O(k * \log^3 n)$ & space $O(\log n)$.

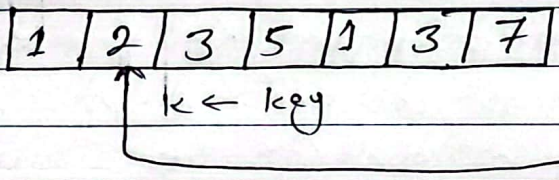
⇒ Identify the Repeated element -:

- Create an empty hash table.
- for each element in the empty array.

- check element in hash table return element is repeated else element inserted in hash table and not repeated.
- If no repeated element found after processing all element in the array, return null.

example:

n = {1, 2, 3, 5, 1, 3, 7}



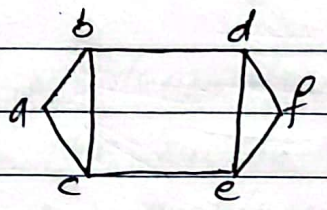
1	
2	
3	
4	
5	
6	
7	

Hash Table.

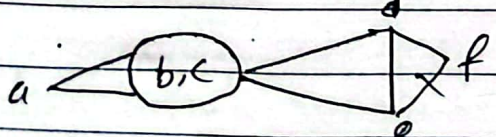
- Karger's Algorithm:-
- Randomized Algorithm
- provide Graph G = (V, E)
- Create sub-graph with minimum cut of edge.
- It is also known as min-cut problem.
- It is also called monte carlo algorithm.

  used in computer Network.
  - To control ~~cong~~ congestion control
  - To set protocol.

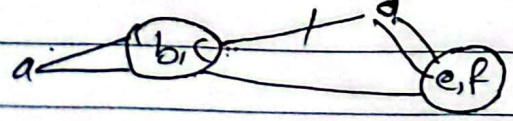
eg G =



cut b to e then



cut e to f then



cut b, c to d then

cut a to b, c, d.

$\Rightarrow \boxed{V_1 \cap V_2 = \phi}$ It