**Answer (1) :**

// Indexing starts from 0.

// A - given array containing books, i - current index of books ,

// Wmax  - maximum width of shelf, R - List of shelves, i.e., a list of lists.

// **Book\_Organize** returns the minimum total height of shelf

1.**Book\_Organize**(A[ ],i,Wmax,R[ [ ] ]):

2. if i==n : // if all books are traversed we have reached base case

3. final\_height =0

4. for i in R: // traversing over all shelves

5. h,flag = height\_width\_validator(i,Wmax) //calculating max height and is total width is

feasible

6. if flag == FALSE: // if width is unfeasible return a very large value

7. return INFINITE

8. final\_height= final\_height+ h //calculating final height of shelves from their individual

heights

9. return final\_height

10. result1=R.add\_element\_last( A[i] ) // ADDING the ELEMENT A[i] to LAST LIST/SHELF.

11. result2=R.add\_element\_last( **[** A[i] **]** ) //Creating a new shelf/list with ELEMENT A[i] and placing at the end of R.

12. return **argmin** { Book\_Organize ( A , i+1 , WMax,result1), Book\_Organize ( A , i+1 , WMax,result2) }

// Book is either added to the previous shelf or to a new shelf.

// **shelf**: list of books in that shelf, **Wmax** - maximum width of the shelf.

// returns the maximum height of that shelf and a flag stating if it's a valid shelf

13.**height\_width\_validator(shelf,Wmax):**

14. max\_height=-1

15. w=0

16. for j in shelf :

17. if j.height > max\_height:

18. max\_height= j.height // calculates maximum height among all books in shelf

19. w += j.width // calculates total width of books in that shelf

20. if w > Wmax: // if total width exceeds maximum possible width Wmax

21. return max\_height,FALSE; // return the height calculated and flag FALSE

22. return max\_height,1 // if width is valid return height and flag TRUE

**NOTE -**

* Each element in R, is a list that represents a shelf that contains some number of books i.e. R is a list of lists.
* The maximum size of R and maximum size of shelf can be equal to a number of books.

**Rough justification of proof:**

**Height\_width\_validator:-** This function takes two arguments as input: a list of books & the maximum possible width. It performs two operations: calculates the maximum height of that shelf i.e. the maximum height among the books, and checks if the total width of books is feasible or not.

If feasible then it returns the **maximum height** of the book with **flag=TRUE**, else return **some height** with **flag=False.**

The **Book\_Organize** function takes 4 parameters as inputs:

**A** is initialized with **n** **books**, **i** is initialized with **1**, **Wmax** is initialized with **maximum shelf width**, **and R** is **initialized with the first shelf/list containing A[0] i.e first book as a starting point.**

Since the indexing starts from 0, the ith  index contains **(i+1)th** book.

Now, for any ith recursive call, if the value of i=x then it refers to the **(x+1)th** book.

Each recursive call of **Book\_Organize**  will access a BOOK. The BOOK has two possibilities:

1. BOOK goes on the current shelf.In this case **book A[ i ] i**s added to the last existing Shelf/List present in **R. [ Ref. line 11 ]** Then **Book\_Organize** is called recursively for the next book in array **A**.
2. BOOK goes on a new shelf. In this case a book **A[ i ]** is added to a newly created shelf/List **&** that list is added to the end of **R [Ref. line LINE 12 ].** Then **Book\_Organize** is called recursively for the next book in array **A**.

Since we need to minimize the **total height** of the shelves, **argmin** is applied to the returned values from the above 2 cases.

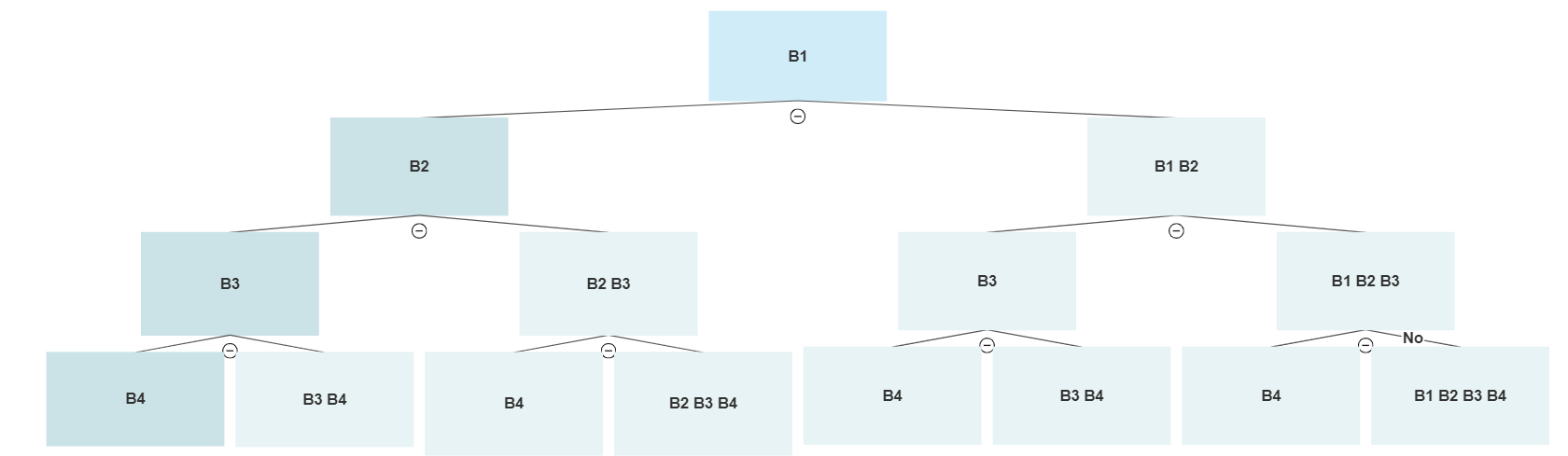
All books are traversed one by one. Each book is added to the shelf/list using the above two cases until the **base case is reached i.e i = n** **[ Line 2 ]**.

When the base case is reached, all books must have been added to **R**. Now the base case will validate if the given configuration is possible or not, and also calculate the cumulative height of all shelves in R using **height\_width\_validator** which will return cumulative height and **flag**  [Line 5 ].

**Flag**=**TRUE** denotes that the given configuration is **feasible** & **Flag**=**FALSE** represents that theconfiguration is **not feasible**.

If the given configuration is feasible, then the function returns the total height of all shelves, else it returns an extremely large value, say **infinite. [ Line7, Line 9]**

Here is an example of four books being added, each path from the root to the leaf represents one possible combination of books and shelves. Nodes containing multiple books denotes that all those books are added to a single shelf/list.



**Time Complexity( height\_width\_validator )**:

Let **C(s)** be the time complexity of **height\_width\_validator** where **s** is the number of books in that given shelf.

Since we are traversing through all the **s** elements of the shelf and performing some constant time operations (say takes time **d’),** So, the total time complexity is:

**C(s) = O(s + d’) =>**

**Time Complexity( Book\_Organize )-**

Let **T(n)** be the running complexity of **Book\_Organize,** where **n** is the number of books in **A**.

In line 12, two recursive calls are made with i updated as (i+1), hence the effective size of the number of books in **A** is reduced by 1. So, it takes **2 \* T(n-1)** time. ….**(P)**

On reaching the base case, **R** contains **n** books which are divided in some shelves, say **m** shelves.

Line[ 4 - 8 ]: The **for loop** and function **height\_width\_validator** are used to traverse over those **m shelves containing n books**.

In other words, they are traversing over **n** elements. So, it takes **O(n)** time. ….**(Q)**

Base case is reached when array A is traversed i.e no elements left in array. So,

Thus from **P & Q :**

Generalizing for some constant k.

On solving above series for k=n we get