Assignment:

Q.

Data item	A	B	C	D	E	F	G	H
Frequency	22	5	11	19	2	11	25	5

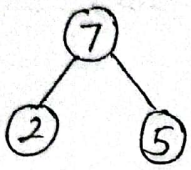
Construct Huffman tree for the given data items.

→

step 1:

Available frequency: 22, 5, 11, 19, 2, 11, 25, 5

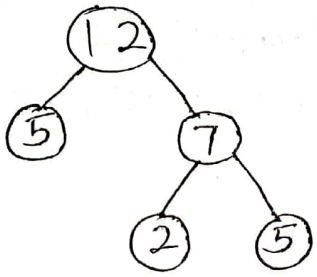
Two minimum frequency: 2, 5



Step 2:

Available frequency: 22, 7, 11, 19, 11, 25, 5

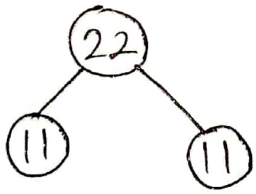
Two minimum frequency: 5, 7



Step 3:

Available frequency: 12, 22, 11, 19, 11, 25

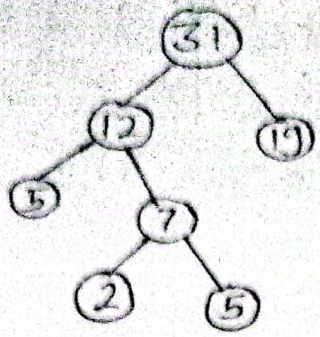
Two minimum frequency: 11, 11



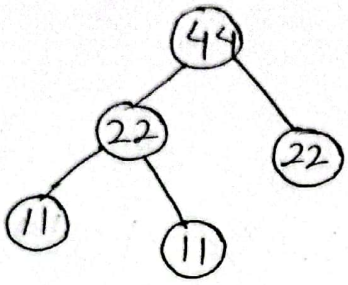
Step 4:

Available frequency: 12, 22, 22, 19, 25

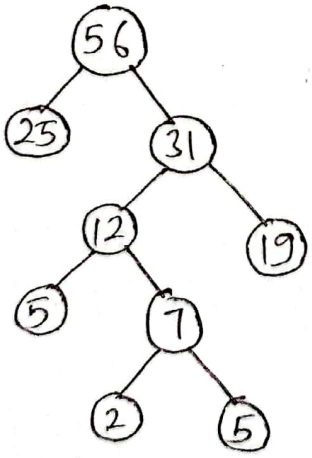
Two minimum frequency: 12, 19



Step 5:
Available frequency: 31, 22, 22, 25
Two minimum frequency: 22, 22



Step 6:
Available frequency: 44, 31, 25
Two minimum frequency: 31, 25



Step 7:
Available frequency: 56, 44
Two minimum frequency: 56, 44

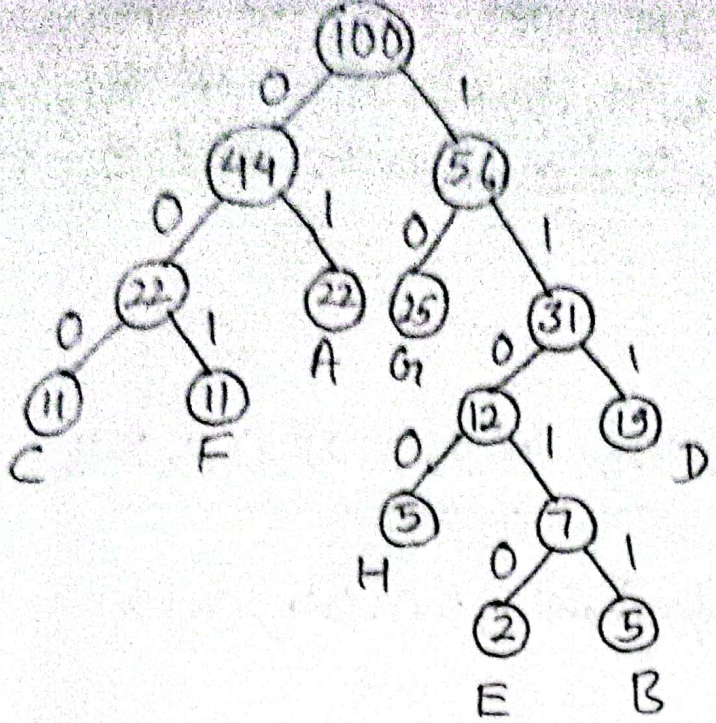


Fig: Huffman Tree

Now, the Huffman Code (or variable length code) for each character is as follows:

Data item	frequency	Variable length code	no. of bits
A	22	01 → 2 bit	22×2 = 44
B	5	11011 → 5 bit	5×5 = 25
C	11	000 → 3 bit	11×3 = 33
D	19	111 → 3 bit	19×3 = 57
E	2	11010 → 5 bit	2×5 = 10
F	11	001 → 3 bit	11×3 = 33
G	25	10 → 2 bit	25×2 = 50
H	5	1100 → 4 bit	5×4 = 20
			272 bits

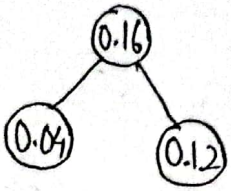
Q. Generate huffman codes for the symbol A, B, C, D, E, F with probability of occurrence are 0.2, 0.28, 0.2, 0.16, 0.12 and 0.04 respectively.

→

Step 1:

Available probability: 0.2, 0.28, 0.2, 0.16, 0.12, 0.04

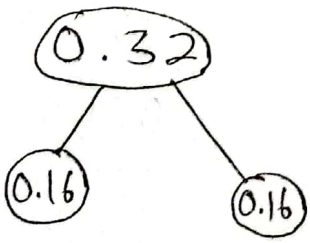
two minimum probability: 0.04, 0.12



Step 2:

Available probability: 0.16, 0.2, 0.28, 0.2, 0.16

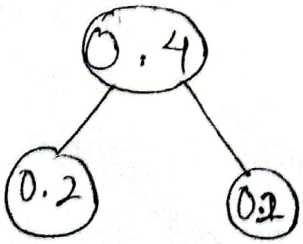
two minimum probability: 0.16, 0.16



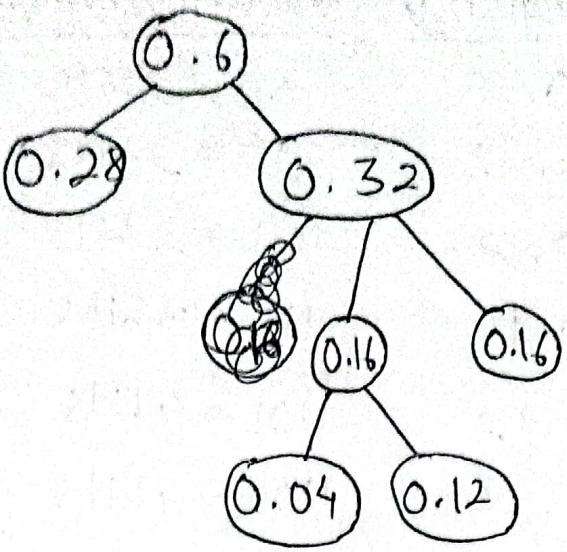
Step 3:

Available probability: 0.32, 0.2, 0.28, 0.2,

two minimum probability: 0.2, 0.2



step 4:
Available probability: 0.4, 0.32, 0.28
two minimum probability: 0.32, 0.28



step 5:
Available probability: 0.6, 0.4

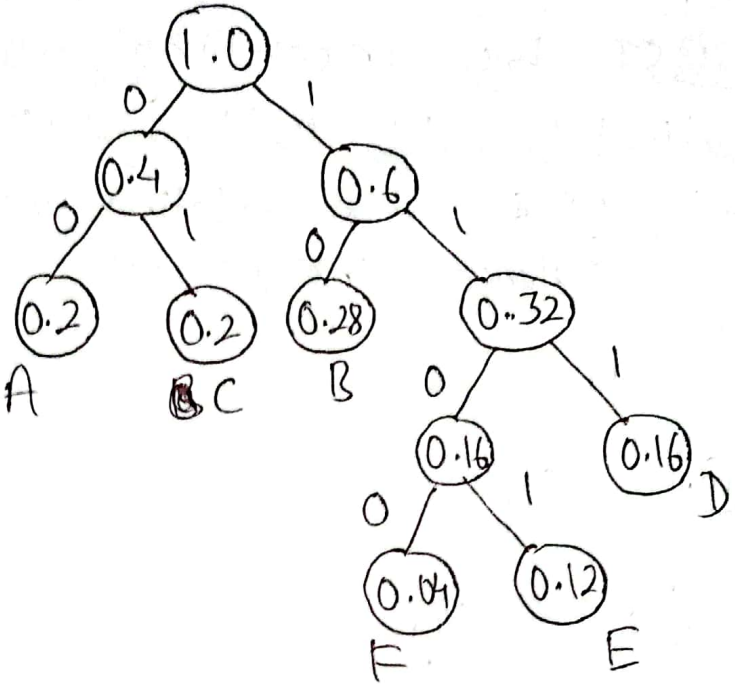


Fig: Huffman tree

Now, the Huffman code (or variable length code) for each character is as follows:

Data item	frequency	variable length code	no. of bits
A	0.2	00 → 2 bits	0.2×2=0.4
B	0.28	10 → 2 bits	0.28×2=0.56
C	0.2	01 → 2 bits	0.2×2=0.4
D	0.16	111 → 3 bits	0.16×3=0.48
E	0.12	1101 → 4 bits	0.12×4=0.48
F	0.04	1100 → 4 bits	0.04×4=0.16
			2.48 bits

Assignment:

Form a BST by inserting number from 14, 10, 17, 12, 11, 20, 18, 25, 20, 8, 22, 23 and form a BBST by deleting 20.

(Balanced Binary Tree)

→

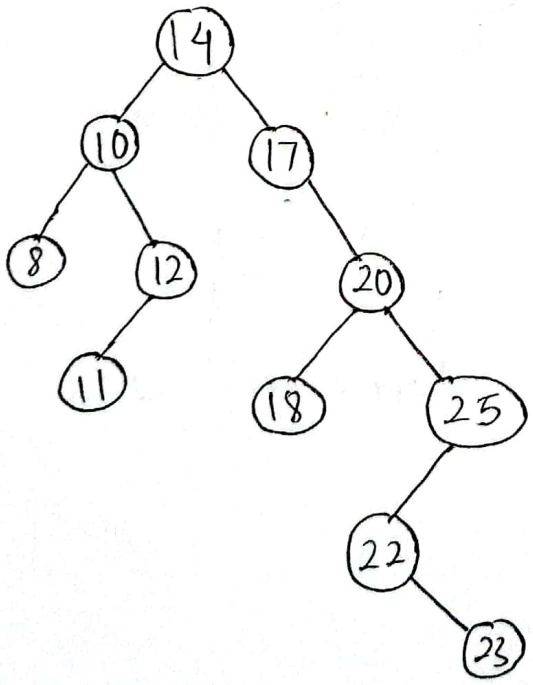
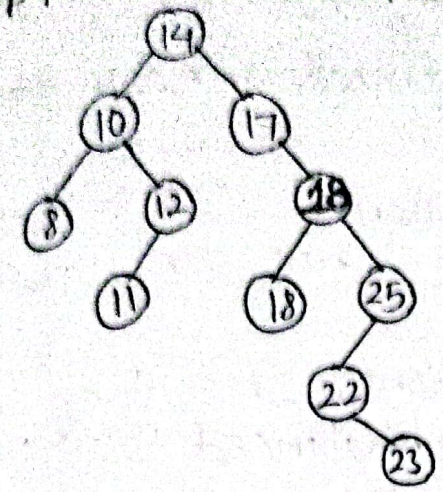


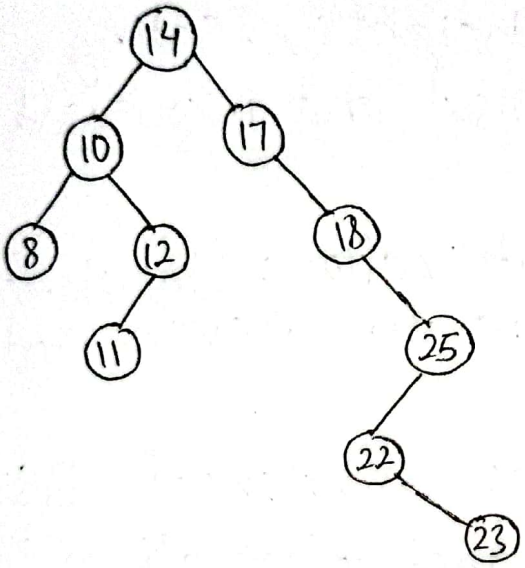
Fig: BST

* Delete 20:

step1: Copy 18 in place of 20



step2: Delete previous 18. node.



~~step1: Copy 22 in place of 20~~

Assignment:

Q. Problem with unbalanced binary search tree

→

The problems with unbalanced binary search tree are as follows:

1. Inefficient search: Unbalanced BST result in longer search time due to unbalanced tree structures.
2. Slow insertions and deletions: Inserting or deleting nodes in an unbalanced BST requires time consuming rebalancing processes.
3. Uneven tree height: Unbalanced BSTs have uneven subtree heights, leading to inefficient traversal and increased memory usage.
4. Suboptimal space utilization: Unbalanced BSTs waste memory due to uneven node distribution.
5. Reduced performance: Unbalanced BSTs suffer from degraded ~~operation~~ performance with operations taking longer than expected.
6. Sensitivity to input order: Unbalanced BST's structure and performance vary depending on the order of element insertion.

Q. Advantage of balanced binary search tree.

→ Advantages of balanced binary search tree are as follows:

1. Efficient searches: Balanced search trees allow for quick retrieval of data, making searches faster and more efficient.
2. Fast insertions and deletions: Balanced trees facilitate speedy additions and removals of elements, ensuring efficient data manipulation
3. Optimal tree height: Balanced trees maintain an optimal height, resulting in efficient traversal and reduced memory usage.
4. Improved performance: Balanced trees offer consistent and improved overall performance in comparision to unbalanced trees.
5. Effective memory usage: Balanced trees utilize memory efficiently, minimizing wasted space and optimizing space utilization.
6. Adaptability to changing data: Balanced trees can handle dynamic datasets, accomodating frequent additions, deletions and modifications while maintaining performance efficiency.

Assignment:

Q. Need of AVL tree

→

i) AVL trees provide efficient searching operations with a time complexity of O(logn).

ii) AVL trees balance themselves during insertions and deletions.

iii) AVL trees are useful for range queries and maintaining sorted data.

iv) Implementation of AVL trees is relatively straightforward i.e. easy to implement.

Assignment:

Form a BST by inserting number from 14,10,17,12,11,20,18,25,20,8,22,23 and form a Binary Balanced Tree by deleting 20.

→

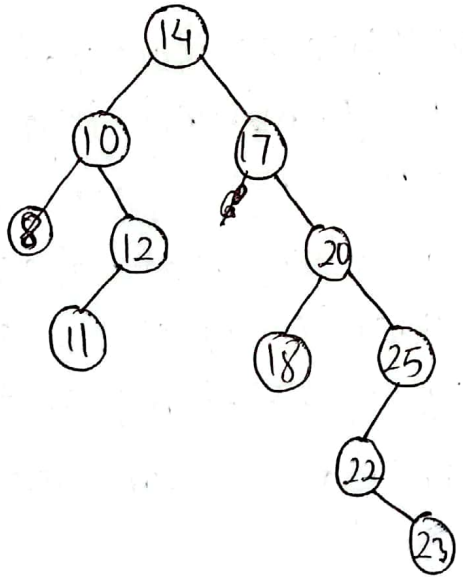
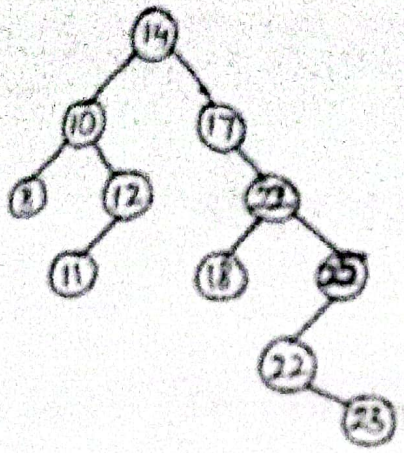
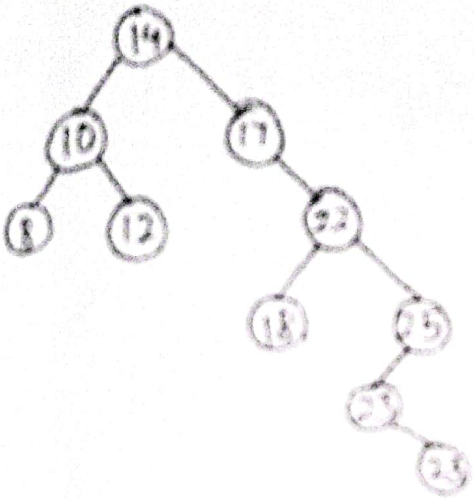


Fig: BST

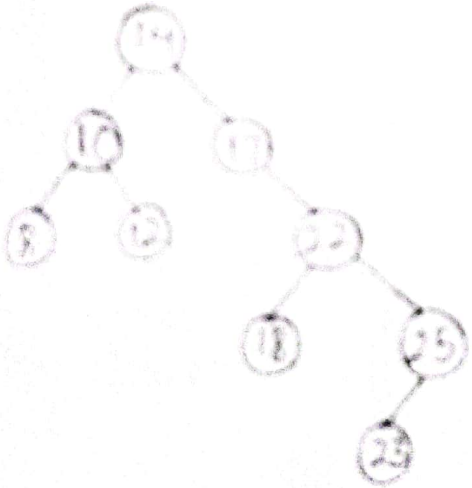
Step 1: Copy 22 in place of 20 i.e. smallest in right.



Step 2: Copy 23 in place of 22.



Step 3: Delete 23



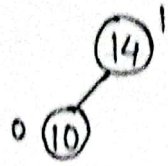
Assignment:

Form an AVL balanced tree from numbers 14, 10, 17, 12, 11, 20, 18, 25, 20, 8, 22, 23.

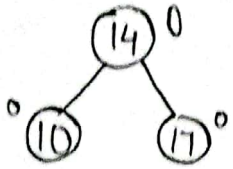
→

Step 1: Make 14 as root node.

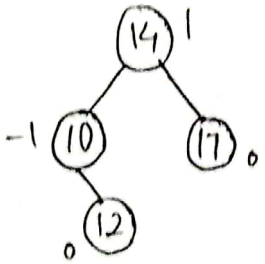
Step 2: Insert 10 on left of 14.



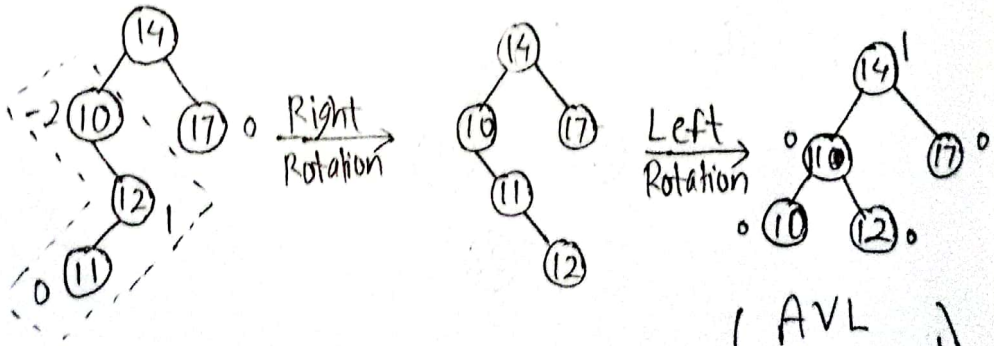
Step 3: Insert 17 on right of 14.



Step 4: Insert 12 on right of 10.



Step 5: Insert 11 on left of 12.

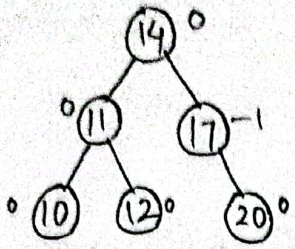


(Not balanced)

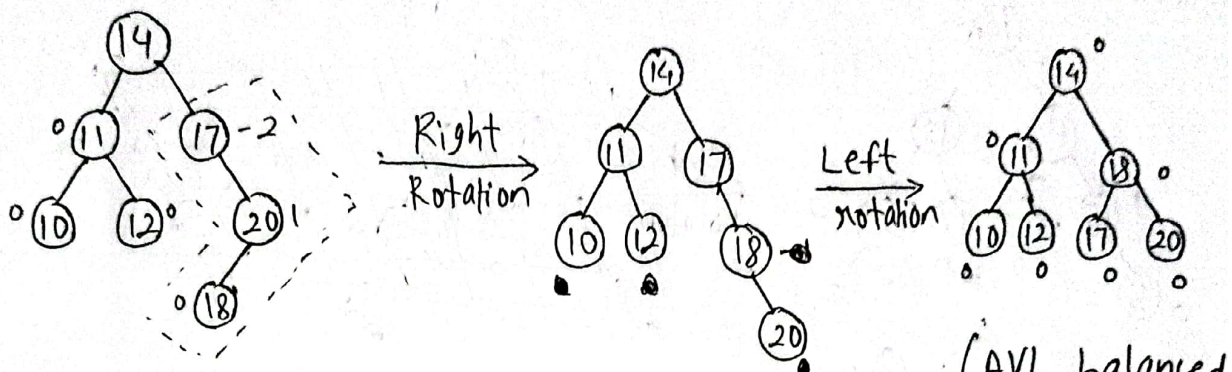
(AVL balanced)

Here, balance factor of node 10 is -2. So, it is not balanced and it is in dogleg pattern. So, we apply double rotation.

Step 6: Insert 20 on right of 17.



Step 7: Insert 18 on left of 20.

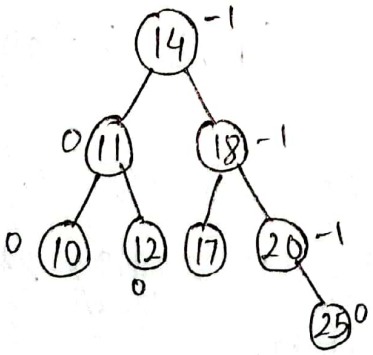


(Not balanced)

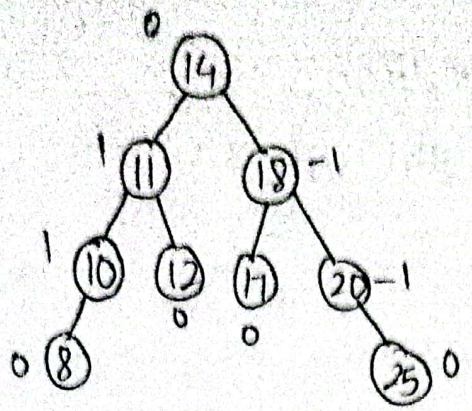
(AVL balanced

Here, balance factor of node 17 is -2. So, it is not balanced. And, it is in dogleg pattern. So, we use double rotation.

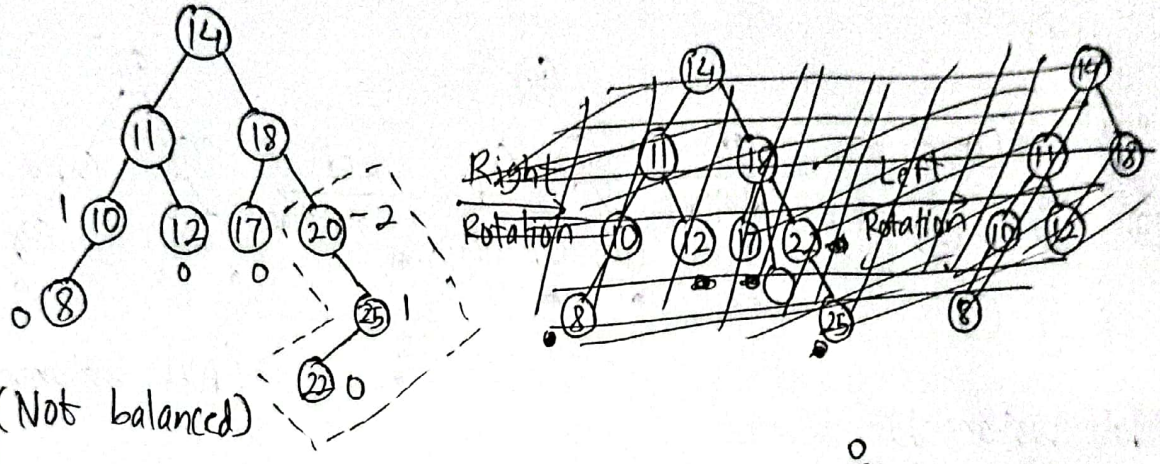
Step 8: Insert 25 on right of 20.

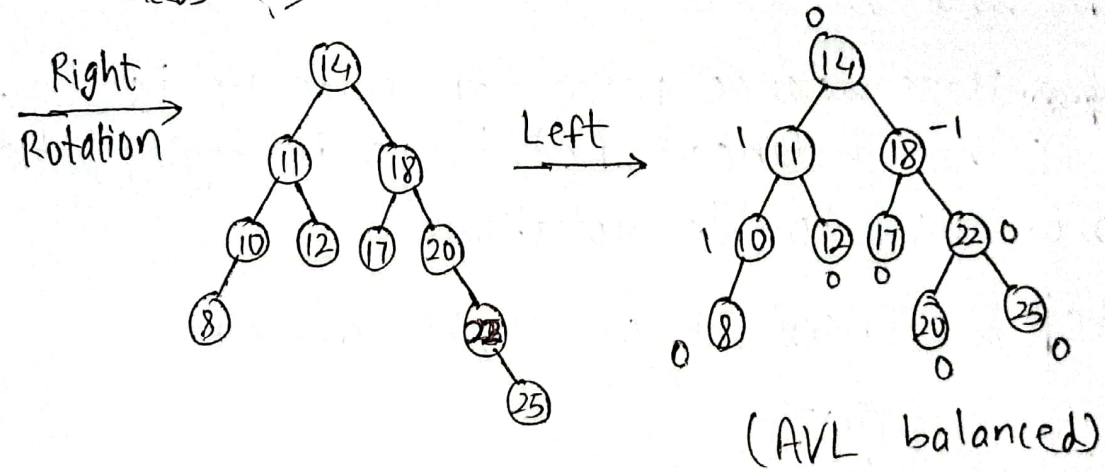


Step 9: Insert 8 on left of 10.



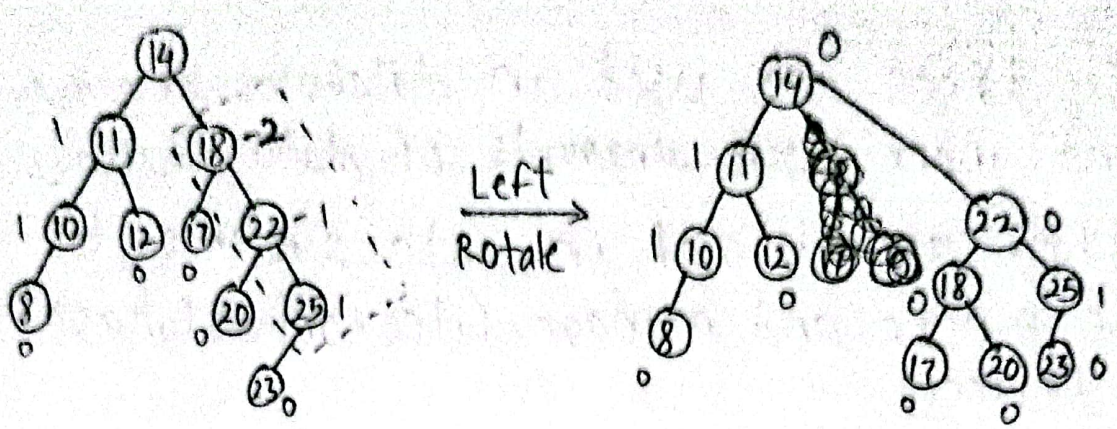
Step 10: Insert 22 on lef of 25.





Here, balance factor of node 20 is -2. So, it is not balanced and it is in dogleg pattern. So, we apply double rotation.

step 11: Insert 23 on left of 25.



(Not balanced) (AVL balanced)

Here, balance factor of node 18 is -2. so, it is not balanced and it is in straight pattern. So, we apply single rotation.

Assignment:

Q. Explain why we need B Tree?

→ We need B tree because they allow us to efficiently store and retrieve large amount of data in a sorted manner. B-trees provide fast searching capabilities, can handle massiv datasets and maintain data in a sorted order They automatically balance themselves to optimiz performance, making them suitable for databases and file systems. B trees are essential for organizing data, minimizing disk access and enabling efficient queries and data management

Q. Applications of B tree

→

i) B- trees are used in databases to store and index large amounts of data efficiently.

ii) They are utilized in file systems to organize and manage files on storage devices.

iii) B tree are used in operating systems to manage file system structures and optimize disk I/O operations.

iv) B trees are used in search engines to store and index web pages for quick retrieval (obtain back).

v) B trees have applications in network routers and caches to store routing tables and cache data effectively.

Chapter - 5
Sorting Algorithm

Assignment:

* Internal sort / External sort;

→ Internal sort refers to a class of sorting algorithms that are designed to work exclusively within the primary memory (RAM) of a computer system. The main characteristics of internal sorting is that the entire dataset to be sorted can fit comfortably within the available memory thereby enabling direct access and manipulation of data in RAM. These algorithms are typically well suited for managing smaller datasets due to the fast and efficient access to memory elements, resulting in relatively faster sorting times compared to their sorting counterparts.

External sort is a method used to sort large datasets that cannot fit entirely into a computer's memory (RAM). It divides the data into smaller chunks, sorts them in memory and then merges them back together using the computer's secondary storage (hard drive or SSD). This approach is useful for managing and sorting medium to large-sized datasets efficiently, even when memory limitations would otherwise be a hindrance.

* Stable / Unstable sorting:

Stable sorting: When we use stable sorting method on list of items, items with the same value will maintain their relative order in the sorted output, just as they appeared.

Eg:
Original list: Ram (20), Shyam (25), Hem (20)
Stable sorting result: Ram(20), Hem(20), Shyam(25)
(Example of age in stable sorting)

Unstable sorting: With an unstable sorting method, there is no guarantee that items with equal values will keep their original order in the sorted output. Their positions may change compared to how they appeared in the original list.

Eg: Original list: Ram(20), [illegible] Shyam(25), Hem(20)
Unstable sorting result: Ram(20), Hem(20), Shyam(25)
or, Hem(20), Ram(20), Shyam(25)

* Adaptive / Non adaptive sorting:

An adaptive sorting algorithm is one that takes advantage of the existing order or partial sorting of the input data to optimize its performance. If the input data is partially sorted, or nearly sorted, an adaptive sorting algorithms can identify this characteristics and make use of it to reduce the numbers of comparisions and swaps, leading to faster sorting.

A non-adaptive sorting algorithm doesnot consider the initial order or partial sorting of the input data. It treats all datasets uniformly, regardless of whether they are already sorted, partially sorted or in random order. As a result, it performs the same number of comparisions and swaps regardless of the input data's characteristics.

* Inplace sorting:

Inplace sorting is a type of sorting algorithm that rearranges the elements of a list or array directly within the original data structure, without using any additional memory to create a separate copy.

While sorting data in place, the algorithm only performs swaps or other transformations ~~can~~ on the elements of the list, rather than creating a new list to store the sorted values. This can be beneficial in situations where memory usage is a concern or when you want to avoid unnecessary memory overhead.

Discussion based Tutorials:

(b) Algorithm Design Techniques

⇒ There are many techniques and strategies that algorithm designers use to craft algorithms to solve problem optimally. Some are:

1. Divide and Conquer:

i) Break down a problem into smaller subproblems.
ii) Solve each subproblem independently.
iii) Combine the solutions of subproblems to obtain the final solution.

2. Greedy Algorithm:

i) Make locally optimal choices at each step.
ii) The choice made at each step should lead to an optimal global solutions.
iii) Greedy algorithms do not always guarantee the optimal solution for all problems.

3. Backtracking:

i) Systematically explore all possible solutions to a problem.
ii) Move back (backtrack) when the current path is not a feasible solution.
iii) Often used in problems with a large search space, such as puzzles and combinatorial optimization.

The philosophy of data structures is like a guiding set of rules for organizing and handling data in computers. It focuses on making things easier and more efficient.

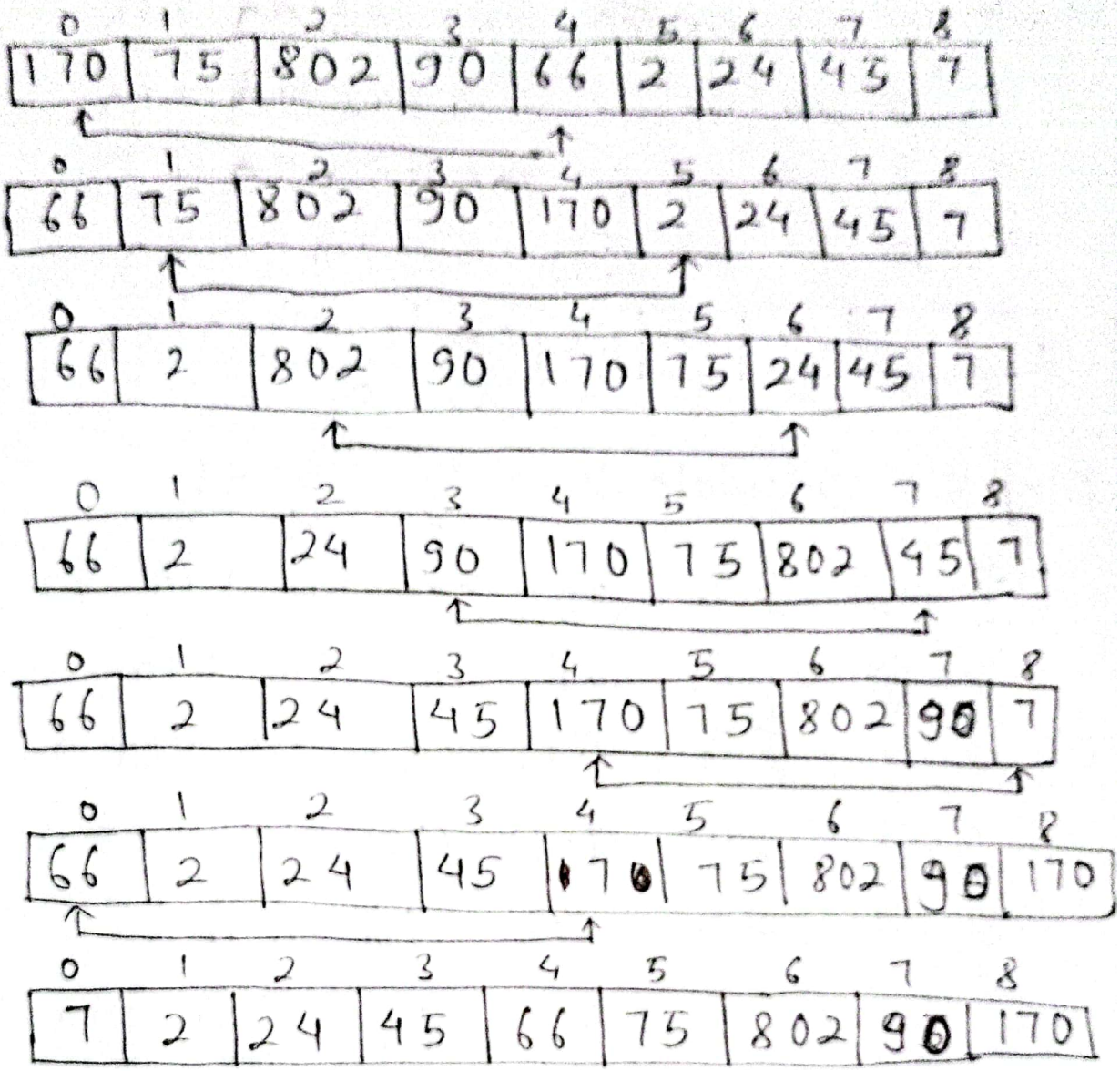
1. Abstraction: This means simplifying complicated things so we don't have to worry the little details. It's like using a TV remote without knowing how it works inside.
2. Efficiency: We want to use computer resources (like memory and processing power) wisely so that everything runs quickly and smoothly.
3. Modularity: ~~Think of it like~~ We want to create small, reusable pieces of code that fit together easily, making it easier to manage and maintain.
4. Correctness: We need data structures to work reliably and give us the correct results.
5. Flexibility: Data can change and our structures should be able to handle those changes without breaking or slowing down.
6. Scalability: As we deal with more and more data, our structures should still perform well and not become slow.
7. Trade-offs: sometimes, we have to make choices about what's more important, like speed versus memory usage. The right balance depends on specific situation.
8. Algorithm Design: We need to pick the best tools (data structures) for each job (algorithm) to get the job done quickly and effectively.

Assignment:

* Use shell sort to sort the following data:
  170, 75, 802, 90, 66, 2, 24, 45, 7

⇒ Here $n = 9$

Step 1: Calculate $h = \frac{n}{2} = \frac{9}{2} = 4$



Step 2: Calculate $h_1 = \frac{h}{2} = \frac{4}{2} = 2$

7 2 24 45 66 75 802 90 170

7 2 24 45 66 75 802 90 170

7 2 24 45 66 75 802 90 170

7 2 24 45 66 75 802 90 170

7 2 24 45 66 75 802 90 170

7 2 24 45 66 75 802 90 170

7 2 24 45 66 75 802 90 170

7 2 24 45 66 75 170 90 802

7	2	24	45	66	75	170	90	802
0	1	2	3	4	5	6	7	8

Step 3: Calculate $h_2 = \frac{h_1}{2} = \frac{2}{2} = 1$

7 2 24 45 66 75 170 90 802

2 7 24 45 66 75 170 90 802

2 7 24 45 66 75 170 90 802

2 7 24 45 66 75 170 90 802

2 7 24 45 66 75 170 90 802

2 7 24 45 66 75 170 90 802

2 7 24 45 66 75 170 90 802

2 7 24 45 66 75 90 170 802

2	7	24	45	66	75	90	170	802
0	1	2	3	4	5	6	7	8