1. Greedy Algorithm Paradigm: Greedy Algorithm paradigm builds up a solution piece by piece. It always choose the next piece that offers the most obvious and immediate benefit. This approach never reconsider the choices taken previously. This algorithm is mainly used to solve optimization problem.

- If we want to calculate or solve optimization problems like knapsack problem in that case we use greedy Algorithm.

2. (i) Activity Selection Problem

T.C

(i) when the given set of activities are already sorted

$T.C = O(n)$.

(ii) when the given set of activities are not sorted.

$T.C = O(n \log n)$.

$S.C = O(1)$

(ii) Job sequencing

$T.C = O(n^2)$

$S.C = O(n)$

(iii) Fractional knapsack

$T.C = O(n \log n) + O(n) = O(n \log n)$

$S.C = O(n)$

(4) Huffman Encoding

T.C = O(nlogn)

S.C = O(n)

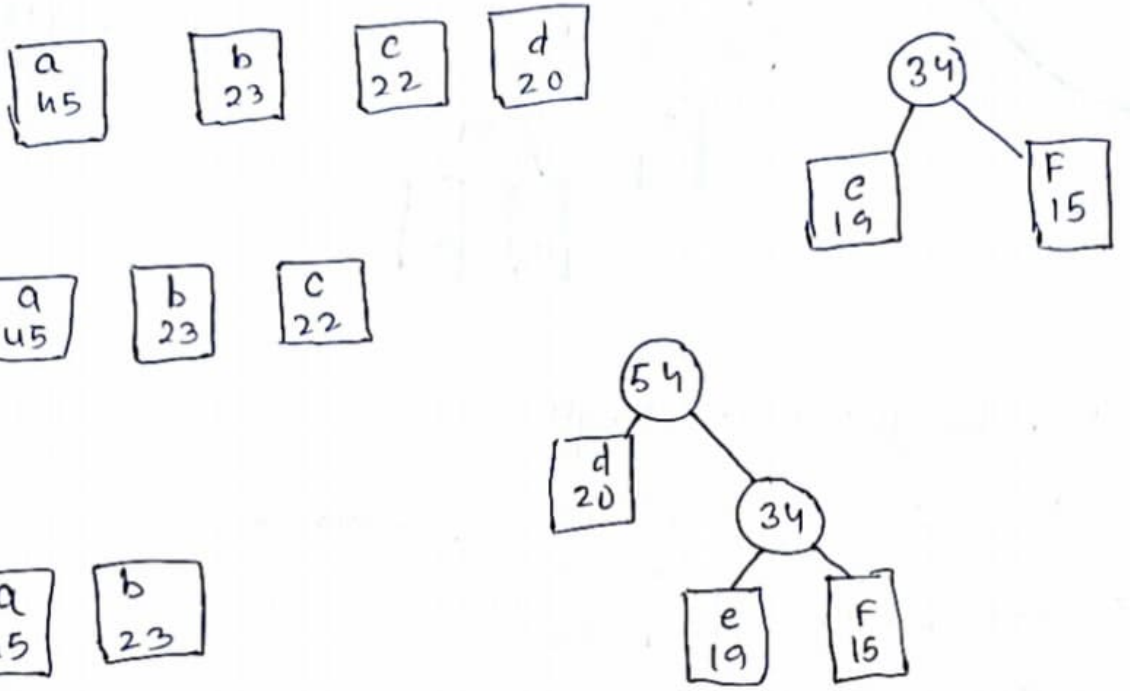
Ans.4 Priority Queue is used while implementing Huffman Encoding. A min heap data structure can be used to implement the functionality of a priority queue.

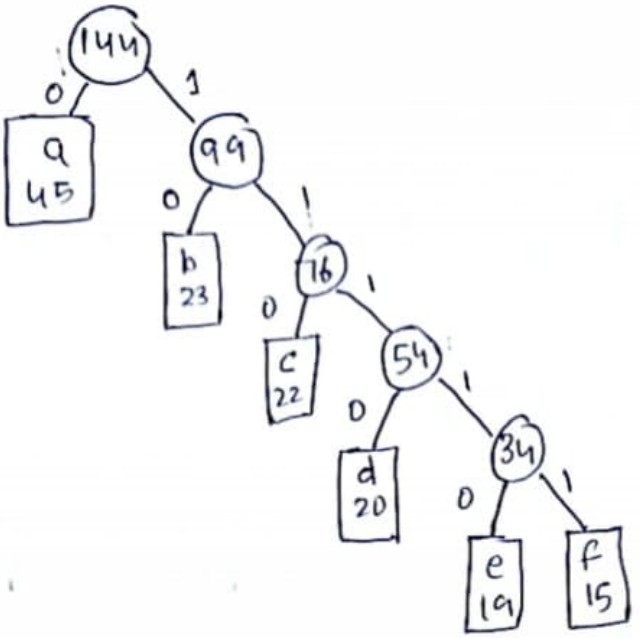
Ans9: In many problems. Greedy algorithm fails to find out an optimal solution, moreover it may produce a worst solution. Problems like Travelling Salesman and Knapsack cannot be solved using this approach.

Fractional knapsack problem can be used solved in reasonably good time by greedy algorithm but 0/1 Knapsack problem can be solved by greedy algorithm in a reasonable time.

Ans: 3

a:45 b:23 c:22 d:20 e:19 F:15





a = 0

b = 10

c = 110

d = 1110

e = 11110

f = 11111

minimum weight path length

$1X45 + 2X23 + 3X22 + 4X20 + 5X19 + 6X15$

$= 45 + 46 + 66 + 80 + 95 + 90$

$= 422.$

Avg. Total length $= \frac{422}{144} = 2.9305$ Ans

Ans 5:

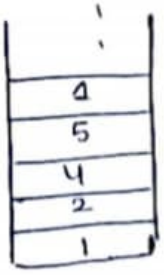
value	10	5	15	7	6	18	3
weight	2	3	5	7	1	4	1

Item = 7

weight = 15.

object	value	weight	fractional.
1	10	2	10/2 = 5
2	5	3	5/3 = 1.66
3	15	5	15/5 = 3
4	7	7	7/7 = 1
5	6	1	6/1 = 6
6	18	4	18/4 = 4.5
7	3	1	3/1 = 3

Ans 5. contd:



} → 2

$\frac{2}{3} \times 15$

weight
= 6 + 10 + 18 + 15 + 3 + 6
= 58 Ans

Ans 10. We use priority queue (max heap) to solve job sequencing problem and its time is $O(n^2)$
we can optimize this by using disjoint set data structure and this time complexity is $O(n \log n)$.

Algorithm

1. Sort all jobs in descending order of profit.
2. Initialize the result sequence as first job in sorted jobs.

If the current jobs can fit in the current result sequence without missing the deadline add current jobs to the result else ignore the current job.