Assignment No. 1

Title: Write a recursive & non-recursive program to calculate Fibonacci numbers and analyze their time complexity.

Objectives:
- To understand Fibonacci numbers using recursive & non-recursive method.
- To understand how recursive & non-recursive algorithm works.
- To analyze time complexity of both the algorithm.

Outcome:
- Student will able to implement Fibonacci numbers using recursive & non-recursive method.
- Student will be able to analyze time complexity of both the algorithm.

Theory:
In Maths, the Fibonacci numbers are the numbers ordered in a distinct Fibonacci sequence. These numbers were introduced to represent the positive numbers in a sequence, which follows a defined pattern. The list of the numbers in a Fibonacci series is represented by the recurrence relation: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... ∞

A Fibonacci number is a series of numbers in which each Fibonacci number is obtained by adding the two preceding numbers. It means that the next number in

in the series is the addition of two preceding numbers. Let the first two no. in the series be taken as 0 & 1. By adding 0 & 1, we get the third no. as 1. Then by adding the second & the third no. (i.e) 1 & 1, we get the 4th no. as 2, & similarly, the process goes on. Thus we get the 3rd no as 1, 4th no. as 2, & similarly the process goes on. Thus, we get the Fibonacci series as 0, 1, 1, 2, 3, 5, 8, ... Hence, the obtained series is called the Fibonacci number series.

Fibonacci Numbers Formula: The sequence of Fibonacci no. can be defined as: $F_n = F_{n-1} + F_{n-2}$

Where F_n is the n^{th} term or number

F_{n-1} is the $(n-1)^{th}$ term.

F_{n-2} is the $(n-2)^{th}$ term

From the eqⁿ, we can summarize the definition as, the next no. in the sequence, is the sum of the previous two numbers present in the sequence, starting from 0 to 1. So, the first six terms of Fibonacci sequence is 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 ....

Implementation of Fibonacci no. using non-recursive algorithm

```
classfib_nonrec {
static void Fibonacci (int N)
}
int num1 = 0, num2 = 1;
int counter = 10;
while (counter > 0) {
    System.out.print(num + " ");
    int num3 = num2 + num1;
num1 = num2;
        num2 = num3;
        counter = counter - 1;
}}
```

```
public static void main (String args[])
  {
    int N = 10;
    Fibonacci (N);
  }
}
```

Analysis of Algorithm:

- The Time Complexity of Fibonacci series is T(N) ie. linear, we have to find the sum of two terms & it is repeated n times depending on the value of n.
- The space complexity of the Fibonacci series using dynamic programming is O(1).

What is Recursion ?

The process in which a function calls itself directly or indirectly is called recursion & the corresponding function is called a recursive function. Using a recursive algorithm, certain problems can be solved quite easily.

Ex. of such problems are Towers of Hanoi (TOH), Inordeel Preordeel Postorder Tree Traveesals, A recursive function solves a particular problem by calling a copy of itself & solving smaller subproblems of the original problems. Many more recursive calls can be generated as & when required. It is essential to know that we should provide a certain case in order to terminate this recursion process so we can say that every time the function calls itself with a simpler version of the original problem.

Need of Recursion:
Recursion is an amazing technique with the help of which we can try to reduce the length of our code & make it easier to read & write. It has certain advantages over the iteration technique which will be discussed later. A task that can be defined with its similar subtask, recursion is one of the best solutions for it.

Implementation of Fibonacci numbers using non-recursive algorithm (Method -1)

```
public class FibonacciExample1 {
  public static void main (String args[]) {
    int n1=0, n2=1, n3, i, max=5;
    System.out.print(n1+" "+n2);
    for (i=2; i<max; ++i) {
      n3 = n1+n2;
      System.out.print(" "+n3);
      n1 = n2;
      n2 = n3;
    }
  }
}
```

Analysis of Algorithm:

- The time complexity of Fibonacci series is T(N) ie linear we have to find the sum of two terms & it is repeated n times depending on the value of n.
- The space complexity of the Fibonacci series using dynamic programming is O(1).

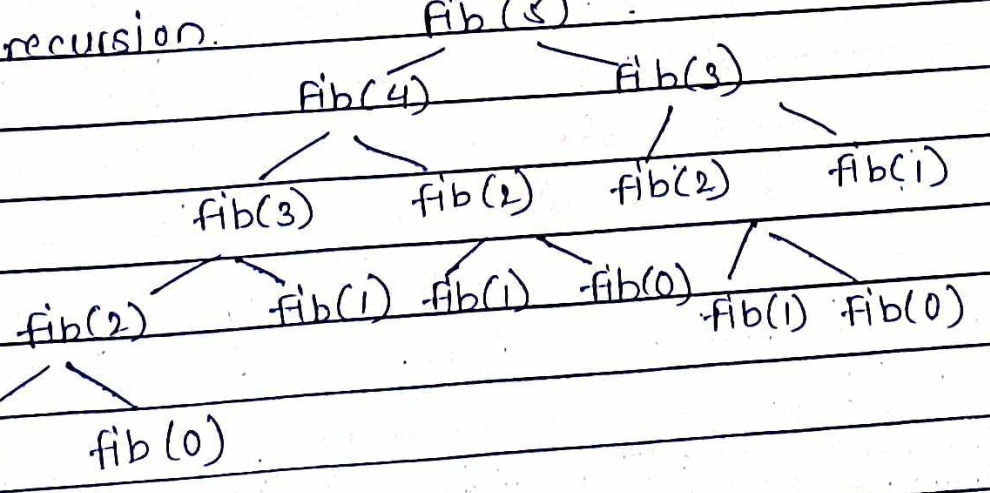
Implementation of Fibonacci no using recursive algo (Method-2)

```
class FibonacciExample2 {
  static int n1=0, n2=1, n3=0;
  static void printFibonacci(int count) {
    if (count>0) {
```

```
n3 = n1 + n2;
n1 = n2;
n2 = n3;
System.out.print(" "+n3);
print Fibonacci (count - 1);
}
}
public static void main (String args[]){
int count = 10;
System.out.print(n1+" "+n2); // printing 0 & 1
print Fibonacci (count - 2); // n-2 because 2 nos are
                              already printed
}
}
```

Analysis of Algorithm:

Time complexity : Exponential, as every function calls two other functions. If the original recursion tree were to be implemented then this would have been the tree but now for n times the recursion funct° is called Original tree for recursion.



The space complexity of the fibonacci series using dynamic programming is O(1).

Conclusion:

Both recursive & iterative program have the same problem solving powers ie. every recursive program can be written iteratively & vice versa is also true. The recursive program has greater space requirements than the iterative program as all functions will remain in the stack until the base case is reached. It also has greater time requirements because of function calls & return overhead.

It also shows that if recursion is not implemented properly then it leads to overhead & increase in time complexity.

Assignment No. 3

Title: Write a program to solve Fractional knapsack problem Using Greedy method.

Objectives:
- To solve a fractional knapsack problem using greedy method.
- To understand how greedy algorithm works.
- To analyze time complexity of the algorithm.

Outcome :
- Student will be able to implement Fractional knapsack problem Using Greedy Method.
- Student will be able to analyze time complexity of the Fractional knapsack algorithm.
- Student will be able to implement an algorithm using Greedy algorithm design strategy.

Theory :
What is a Greedy Method ?
- A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment. It doesn't worry whether the current best result will bring the overall optimal result.
- The algorithm never reverses the earlier decision even if the choice is wrong. It works in a top down approach.

- This algorithm may not produce the best result for all the problems. It's because it always goes for the local best choice to produce the global best result.

Advantage of Greedy Approach

- The algorithm is easier to describe.
- This algorithm can perform better than other algorithms.

Drawback of Greedy Approach:

- As mentioned earlier, the greedy algorithm doesn't always produce the optimal solution. This is the major disadvantage of the algorithm.
- For ex, suppose we want to find the longest path in the graph below from root to leaf.

Greedy Algorithm:

1. To begin with, the sol° set is empty.
2. At each step, an ~~attempt~~ item is added to the sol° set until a sol° is reached.
3. If the sol° set is feasible, the current item is kept.
4. Else, the item is rejected & never considered again.

Knapsack Problem

The knapsack problem is a problem in combinatorial optimization. Given a set of items, each with a weight & a value, determine the no. of each item to include in a collection so that the total weight is less than or equal to a given limit & the total value is as large as possible.
The problem often arises in resource allocation where the decision makers have to choosen from a set of non-divisible projects or tasks under a fixed budget or time constraint, respectively.

Knapsack Problem Variants:

Knapsack problem has the following two variants-

1. Fractional knapsack problem
2. 0/1 knapsack problem

Fractional knapsack problem -

In Fractional knapsack problem,

- As the name suggests, items are divisible here
- We can even put the fraction of any item into the knapsack if taking the complete item is not possible.
- It is solved using the Greedy method.

Fractional knapsack problem Using Greedy Method :

Fractional knapsack problem is solved using greedy method in the following steps -

Step 01 : for each item, compute its value / weight ration

Step 02 : Arrange all items in decreasing order of their value / weight ratio.

Step 03 : Start putting the items into the knapsack beginning from the item with the highest ratio. Put as many items as you can into the knapsack

Ex : For the given instance of knapsack beginning from the item with the highest ratio. Put as many items as you can into the knapsack.

problem find the optimal sol^n using greedy approach if $n=3$ & $M=20$.

Solution:

To solve the problem using greedy approach perform following steps.

Find fractions for Descending order of Profit & weight ration

i	1	2	3
Pi/Wi	30/18	21/15	18/10
Ratio	1.67	1.4	1.8

i	3	1	2
Pi	18	30	21
Wi	10	18	15
Ration in Descending order	1.8	1.67	1.4

- The first object with weight 10 will be placed completely so x_1 will be 1.
- The 2nd object will not be placed completely so find fraction x_2,

$$x_2 = (20-10)/18 = 10/18 = 5/9$$

- The 3rd object will not be placed at all so fraction $x_3 = 0$.
- So the Fraction set is
- And now let us compute $\sum w_i p_i$ & $\sum w_i x_i$.

i	Fraction
x_1	1
x_2	5/9
x_3	0

So, $\sum (w_i p_i) = (30 * (5/9)) + (21 * 0) + (18 * 1) = 34.67$

& $\sum (w_i x_i) = (18 * (5/9)) + (15 * 0) + (10 * 1) = 20$

problem

Hence the maximum profit obtained is 34.67

Time Complexity:

- The main time taking step is the sorting of all items in decreasing order of their value/weight ratio.
- If the items are already arranged in the required order then while loop takes O(n) time
- And algorithm take O(n log n) time to solve problem if we use quick sort for sorting.

Conclusion:

A greedy algorithm is most straightforward approach to solving the knapsack problem, in that it is a one-pass algorithm that constructs a single final sol^n. At each stage of the problem, the greedy algorithm picks the option that is locally optimal, meaning it looks like the most suitable option right now.

Assignment No. 2

Title: Write a program to Implement Huffman's coding algorithm using Greedy Method.

Objective:
- To find out Huffman's coding tree using greedy method.
- To understand how greedy algorithm works.
- To analyze time complexity of the algorithm.

Outcome:
- Student will be able to implement Fibonacci numbers using recursive & non-recursive method.
- Student will be able to analyze time complexity of both the algorithm.
- Student will be able to implement an algorithm using Greedy algorithm design strategy.

Theory:

Greedy Method: Among all the algorithmic approaches, the simplest & straightforward approach is the Greedy method. In this approach, the decision is taken on the basis of current available information without worrying about the effect of the current decision in future.

Greedy algorithms build a solution part by part, choosing the next part in such a way, that it gives an immediate benefit. This approach never reconsiders the choices taken previously. This approach is mainly used to solve optimization problems. Greedy method is easy to implement & quite

efficient in most of the cases. Hence, we can say that Greedy algorithm is an algorithmic paradigm based on heuristic that follows local optimal choice at each step with the hope of finding global optimal sol^.

Greedy algorithm is designed using the following steps:
1. Characterize the structure of an optimal solution.
2. Find the set of feasible solutions.
3. Find the locally optimal solutions.

Huffman's Tree:

Huffman tree or Huffman coding tree defines as a full binary tree in which each leaf of the tree corresponds to a letter in the given alphabet. The Huffman tree is treated as the binary tree associated with minimum external path weight that means, the one associated with the minimum sum of weighted path lengths for the given set of leaves. so the goal is to construct a tree with the maximum external path weight.

Huffman coding is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code & the least frequent character gets the largest code.

There are two methods to construct Huffman's Tree.
1. Fixed Length Method
2. Variable Length Method.

Steps to Create Huffman's Tree:
Input as an array of unique characters along with their frequency of occurrences & output is Huffman Tree
1. Create a leaf node for each unique character & build a min heap of all leaf nodes
2. Extract two nodes with the minimum frequency from the min heap.
3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child & the other extracted nod as its right child. Add this node to the min heap.
4. Repeat steps #2 & #3 until the heap contains only one node. The remaining node is the root node & thee tree is complete.

Ex. Obtain the Huffman's Encoding for the following Data. And encode the code message for word "baba" Using Variable length method.

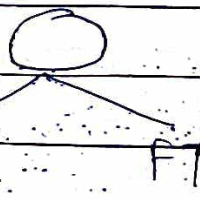
Letters	a	b	c	d	e	F
Frequency	39	10	9	25	7	3

Soln:

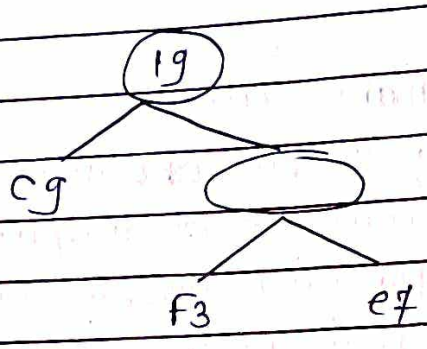
step1 : Arrange the letters in ascending order of frequency

F3 e7 c9 b10 d25 a39

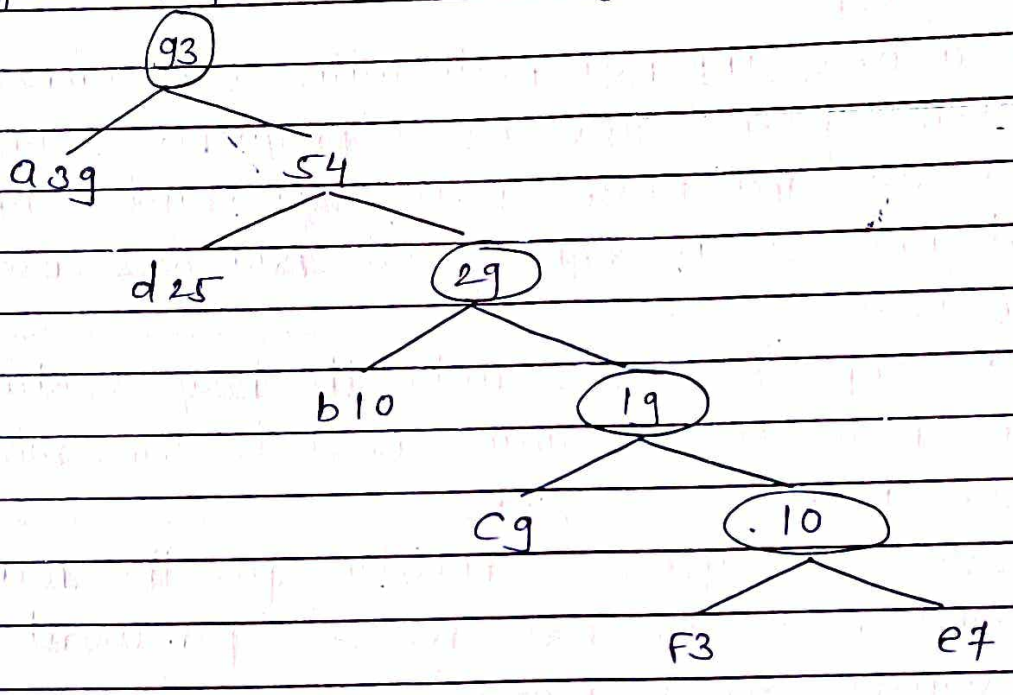
step 2 : Pick 1st two object & merge them to generate parent node



Step 3: Pick next object & merge them with newly generated node, it will be inserted in left side if its value is less than newly generated node's value else at right side.



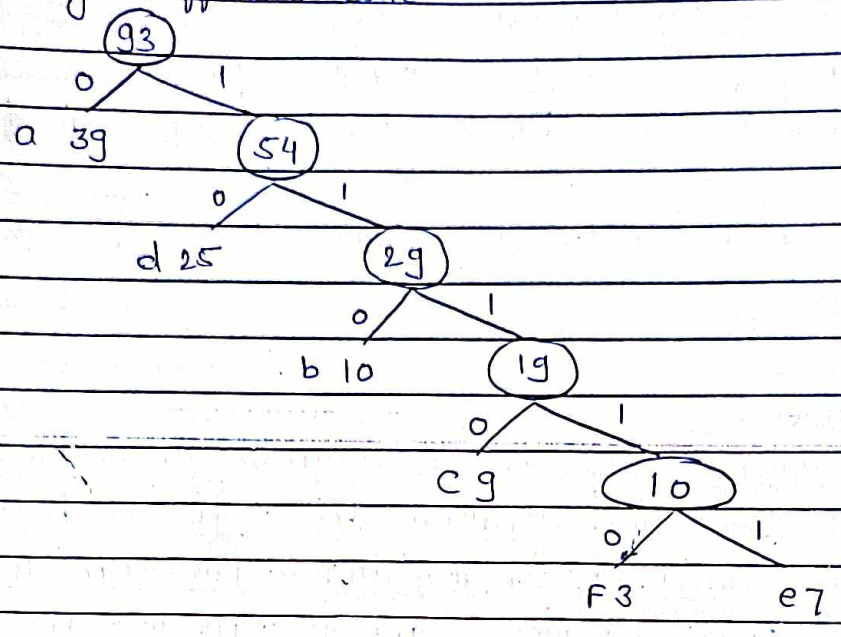
Repeat the process till we get final tree



MET Bhujbal Knowledge City
Institute Of Engineering
Adgaon, Nashik - 422 003.

Date :

Step 5 : Assign Huffman's code



Symbol	Frequency	code	Bits
a	39	0	1
b	10	110	3
c	9	1110	4
d	25	10	2
e	7	11111	5
f	3	11110	5

Encoding the word baba : 11001100

Analysis of Algorithm:

- The time complexity of the Huffman encoding is O(nlo
Where n is the no. of characters in given text.
- The child node holds the input characte. It is asslgned the code formed by subsequent 0s & 1s. The time

various

- We don't have enough information to choose

complexity of decoding a string is O(n), where n is the length of the string.

- The space complexity of the Fibonacci series using dynamic programming is O(n).

Application of Huffman Coding:

1. They are used for transmitting fax & text.
2. They are used by conventional compression formats like PKZIP, GZIP, etc.
3. Multimedia codec like JPEG, PNG & MP3 use Huffman encoding.

Conclusion:

The Huffman algorithm is a greedy algorithm. since at every stage the algorithm looks for the best available options. Greedy is an algorithmic paradigm that builds up a sol^n piece by piece, always choosing the next piece that offers the most obvious & immediate benefit. So the problem where choosing locally optimal also leads to global sol^n are the best fit for Greedy. But still this method does not guarantee that the sol^n obtained is best optimal sol^n.