Day 32 / 100 :

Topic - Binary Tree

1 Problem statement: Minimum speed to Arrive on Time (Medium)

You are given a floating-point number hour, representing the amount of time you have to reach the office. To commute to the office, you must take n trains in sequential order. You are also given an integer array dist of length n, where dist[i] describes the distance (in kilometres) of the Ith train ride.
Each train can only depart at an integer hour, so you may need to wait in between each train ride.
For example, if the 1st train ride takes 1.5 hours, you must wait for an additional 0.5 hours before you can depart on the 2nd train ride at the 2-hour mark.
Return the minimum positive integer speed (in kilometres per hour) that all the trains must travel at for you to reach the office on time, or -1 if it is impossible to be on time.
Tests are generated such that the answer will not exceed 107 and hour will have at most two digits after the decimal point.

Example 1:
Input: dist = [1,3,2], hour = 6
Output: 1
Explanation: At speed 1:
- The first train ride takes 1/1 = 1 hour.
- Since we are already at an integer hour, we depart immediately at the 1 hour mark.
The second train takes 3/1 = 3 hours.
- Since we are already at an integer hour, we depart immediately at the 4 hour mark.
The third train takes 2/1 = 2 hours.
- You will arrive at exactly the 6 hour mark.

Example 2:
Input: dist = [1,3,2], hour = 2.7
Output: 3

Explanation: At speed 3:
- The first train ride takes 1/3 = 0.33333 hours.
- Since we are not at an integer hour, we wait until the 1 hour mark to depart. The second train ride takes 3/3 = 1 hour.
- Since we are already at an integer hour, we depart immediately at the 2 hour mark. The third train takes 2/3 = 0.66667 hours.
- You will arrive at the 2.66667 hour mark

Solutions :

Approach 1 - Binary Search

Intuition:

- We have ans limit from 1 to 10^7 and we have to find minimum speed to reach the office on time .
- so whenever there is range given for ans then here comes binary search in picture.

Approach:

- Make l=1 , r= 10^7
- Now we have to ceiling of all time until n-2 because we have to wait for next train to arrive at integer time.
- But last stop is our destination so we dont need to wait hence we can add non-integer time.
- For every mid we try to find minimum speed to reach destination.
- Base case: we need hours at least greater than n-1 because we have to wait for another train until next integer.

Complexity:

- Time complexity:O(Nlog(10^7))
- Space complexity:O(1)

```
class Solution {
public:
    int minSpeedOnTime(vector<int>& dist, double hour) {
        int n = dist.size();
        if(hour <= n-1) return -1;
        int l=1 , r=1e7;
        while(l<r){
            int mid = l + (r-l)/2;
            double time =0;
            for(int i=0 ; i<n-1 ; i++) time +=
ceil((double)dist[i]/mid);
            time += (double)dist[n-1]/mid;

            if(time <= hour) r =mid;
            else l = mid+1;
        }
        return l;
    }
};
```

2 Problem statement: Letter Combination of Phone number (Medium)

Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in any order.
A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

Example 1:
Input: digits = "23"
Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"]

Example 2:
Input: digits = ""
Output: []

Example 3:
Input: digits = "2"
Output: ["a","b","c"]

Solutions :

Approach 1 - Hash Map

Intuition:

- Each digit in the keypad is mapped with a set of characters that forms a string. Once a digt is pressed, we are faced with the choice of deciding which character to choose from the string it represents. This process continues for each subsequent digit that it pressed.

- Ultimately, we are simply deciding which character to choose from the string represented by the digit and we keep on repeating this process, generating subcases for each choice, till we exhaust the string of digits given to us.

Approach:

- Initialize a vector of strings charMap that stores the string of characters mapped to each digit of its index. Now, call the function generateCombos and pass to it charMap along with the starting index of digits, the string digits, a temporary string temp to store the strings generated and the vector of strings sol to store the possible combinations of strings generated.

- Initialize an integer num and assign to it the numerical value represented by the idxth index of digits. Now, initialize a string str, and assign to it the string represented by the digit.

- Iterate str and push into temp each of its characters. Afterwards, recursively call generateCombos with the next index of digits. Once the function call has been returned, pop out the character that was pushed into temp and move on the next character of str.

- Once idx becomes equal to the length of digits, push into sol the non-empty string temp.

Complexity:
Time complexity: O(N, where 'N' is the length of the digits array
Space complexity: O(1), since the only extra space we are using is for charMap and it always occupies 28 bytes.

```
class Solution {
public:
    void generateCombos(int idx, string& digits, string& temp,
vector<string>& sol, vector<string>& charMap){
        if(idx==digits.length()){
            if(temp.length()) sol.push_back(temp);
            return;
        }

        int num=digits[idx]-'0';
        string str=charMap[num];

        for(int i=0;i<str.length();i++){
            temp.push_back(str[i]);
            generateCombos(idx+1, digits, temp, sol, charMap);
            temp.pop_back();
        }
    }

    vector<string> letterCombinations(string digits) {
        string temp;
        vector<string> sol;
        vector<string> charMap={"", "", "abc", "def", "ghi", "jkl",
"mno", "pqrs", "tuv", "wxyz"};

        generateCombos(0, digits, temp, sol, charMap);
        return sol;
    }
};
```