1. For the set of {1, 4, 5, 10, 16, 17, 21} of keys, draw binary search trees of height 2, 3, 4, 5, and 6.

- Height 2

- Height 3

- Height 4

- Height 5

- Height 6

2. Suppose that the search for the key kin a binary search tree ends up in a leaf. Consider three sets: 1) the keys to the left of the search path; 2) the keys on the search path; 3) the keys to the right of the search path. One claims that any three keys a, b and c from these three sets, respectively (say a from 1, b from 2 and c from 3), it is always true that a ≤ b ≤ c. Is this claim true? Explain your answer.

No. Consider we serach for 3 in the following tree. Then a in {1}, b in {2, 3, 4, 6}, c in {5} It is clear we can pick a = 1, b = 6 and c = 5 which violate a ≤ b ≤ c.

```
      2
  1       6
        4
      3   5
```

3. We can sort a given set of n numbers by first building a binarysearchtree containing these numbers (using Tree_insert algorithm repeatedly to insert the numbers on by one) and then printing the number in an inorder tree walk. What are the worst-case and best-case running times for this sorting algorithm?

We know the running time for inserting is proportional to the hight of tree.

Therefore, in the best case, we have nearly balance tree and the hight of that tree is equal to log n. Hence, the running time for best case is $\sum (\log i + d)$, which i from 1 to n, and d is const cost. Accordingly, the total cost is $O(n*\log n)$.

For worst case, we have complete biased linear tree, When we insert the nth element to the tree, the hight of the tree is n-1. Hence, the cost is $\sum ((i-1) + d)$ which i from 1 to n, and d is const cost. Accordingly, the total cost is $O(n^2)$.

4. Suppose two teams A and B are playing a match to see who is the first to win n games (from given n). Assume that A and B are equally competent, so each has a 50% chance of winning any particular game. Suppose they have already played i+j games, of which A won i and B has won j. Given an efficient algorithm to compute the probability that A will go on to win the match. For example, if i=n-1 and j=n-3 then the probability that A will win the match is 7/8, since it must win any of the next three games.

Assume x, y be the remaining games team A, B need to win, then x = n - i, y = n - j.

P(x, y) be the change the team A will win.

For P(1, 1), we know team A has 0.5 to win the match, because A have 50% to win a game. For P(1, 2), We know there is 0.5 that Team A will win next game and win the match, and 0.5 that Team A will lose the game and fallback to P(1, 1) sceniaro.

Accordingly, We can get formula that $P(1, 1) = 0.5 + 0.5*P(1, 2)$. We can use induction to get general relation $P(1, y) = 0.5 + 0.5*P(1, y+1)$. Due to the rule of symmetry, we also know that P(2, 1) is also equal to $0.5 + 0.5*P(1, 1)$ as well.

In fact, we can get the overall recursion function,

1. $P(x, y) = 0.5*P(x-1, y) + 0.5*P(x, y-1)$
2. $P(1, 1) = 0.5$.

Or we can replace the parameter back to get

1. $P(n-i, n-j) = 0.5*P(n-i+1, n-j) + 0.5*P(n-i, n-j+1)$
2. $P(n-1, n-1) = 0.5$.

5. Give an O(nt) algorithm for the following task. Input: a list of n positive integers a1, a2, ..., an; a positive integer t.Question: does some subset of the ai's add up to t?

The recursion function is

1. `F(t, Set) = F(t-a1, Set/a1) || F(t-a2, Set/a2) || F(t-a3, Set/a3) ... || F(t-an, Set/an)` for all ai Set and `ai < t`.
2. `F(t, φ) = false`.
3. `F(0, Set) = true`.

(`Set/a1` mean remove a1 element from Set.)

`F(t-ai, Set)` will be computed for all ai in Set, and therefore is `O(n)`.

We repeat such recursion until we hit `F(0, Set)` or `F(t, φ)`. And the worst case is repeat t times, so the overall time complexity is `O(n*t)`.

6. Professor Midas drives an automobile from Newark to Reno along Interstate 80. His car's gas tank, when full, holds enough gas to travel n miles, and his map gives the distances between gas stations on his route. The professor wishes to make as few gas stops as possible along the way. Give an efficient method by which Professor Midas can determine at which gas stations he should stop, and prove that your strategy yields an optimal solution.

This problem can be solved by greedy algorithm or dynamic programming. I use dynamic programming to solve it.

Assume the route start from x0(Newark), and end at xk(Reno), and i gas stations are located in x1, x2, x3 .... xi, where xi is between x0 and xk.

We can draw the picture like following.

```
x0 ---- x1 -- x2 ---- x3 --- x4 - ... --- xk-1 -- xk
```

Now we can define the recursion function S(xi):

```
if D(xi) < n
  S(xi) = 0
else
  S(xi) = min { S(xi)+1 } for all j < i, and (D(xj) - D(xi)) <= n
```

D is the distance between xi and x0, and S(xi) is the number of stops we have to get to place xi. For `D(xi) < n` we can choose not to stop at any gas stations to arrival D.

For example:

```
k = 7, n = 8
D(x0) = 0, D(x1) = 4, D(x2) = 7, D(x3) = 9, D(x4) = 11, D(x5) = 14, D(x6) = 17,
D(x7) = 19
```

```
S(x7) = min { S(x5)+1, S(x4)+1 }
S(x6) = min { S(x5)+1, S(x4)+1, S(x3)+1 }
S(x5) = min { S(x4)+1, S(x3)+1, S(x2)+1 }
S(11) = min { S(x3)+1, S(x2)+1, S(x1)+1 }
S(x3) = min { S(x2)+1, S(x1)+1 }
S(x2) = 0
S(x1) = 0
```

We can proof we get the optimal solution by induction.

Let OS(k) is the set of optimal solution for traveling from x0 to xi for all 1 < i < k, and O(k) is the optimal solution for traveling to xk.

For example,

`OS(1) = {O(1)}` and `OS(4) = {O(1), O(2), O(3), O(4)}`

For proof by induction, we need the base case and step case,

base case: if k = 1, OS(1) = O(1) we know 0 is the solution for this problem. Otherwise, we have no way to leave x0 because x1 is the cloest gas station to x0.

Induction step:

Assume we get the OS(k-1), we need to proof we can calculate OS(k) through this algorithm.

Since we check all the gas station that can be reached by xk in distance n, say they are `xi, xi+1, ... xk-1`, and we already have the optimal solutions for them, which is `O(i), O(i+1) .... O(k-1)`, we know we can get solution for reaching xk through `xi xi+1 ... xk-1`.

Hence, we choose the minium value among those subproblems, and we are garanteed to get the optimal solution for xk. That is O(k).

Finally, we can get OS(k) by OS(k-1) U { O(k) }. Finish the proof.