ICS Fall 2021

Algorithm: Part 4

Algorithm design workshop

Before you start,

- 5 compulsory questions, 1 bonus questions
- one compulsory question each team, find a solution + **code** it
- Do bonus questions when you complete the compulsory question

- Some hints are given on the slide next to each question, but you'd better read them **after** you have tried the question by your own.
- Start code or sample solutions are available in Brightspace/Resources

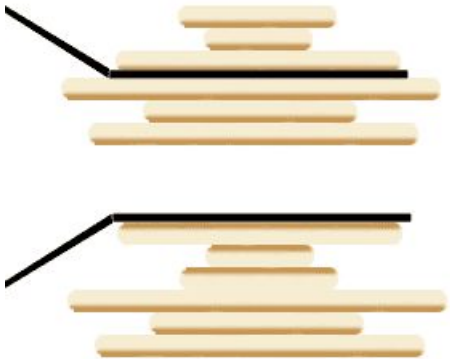
1. Pancake sorting

Given a list of numbers, you can imagine it is a stack of n pancakes of different size. You are **only** allowed to slip a spatula under one pancake and flip. You task is to sort these pancakes, make the smallest at the top. **Goal:** sort the pancakes (smallest at the top). This link shows how pancake sorting works: https://www.youtube.com/watch?v=kk-_DDgoXfk.

The following code is for generation of "unsorted pancakes".

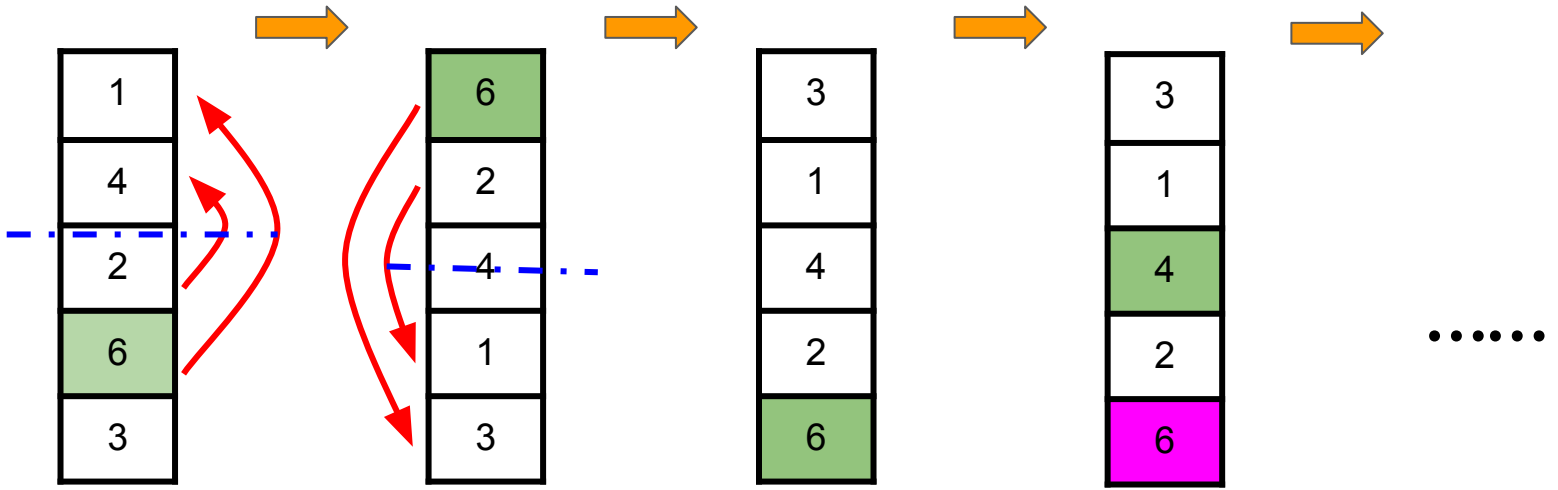
```
import random
random.seed(0)


def get_pancake(a, b, n):
    return [random.randint(a, b) for i in range(n)]
```



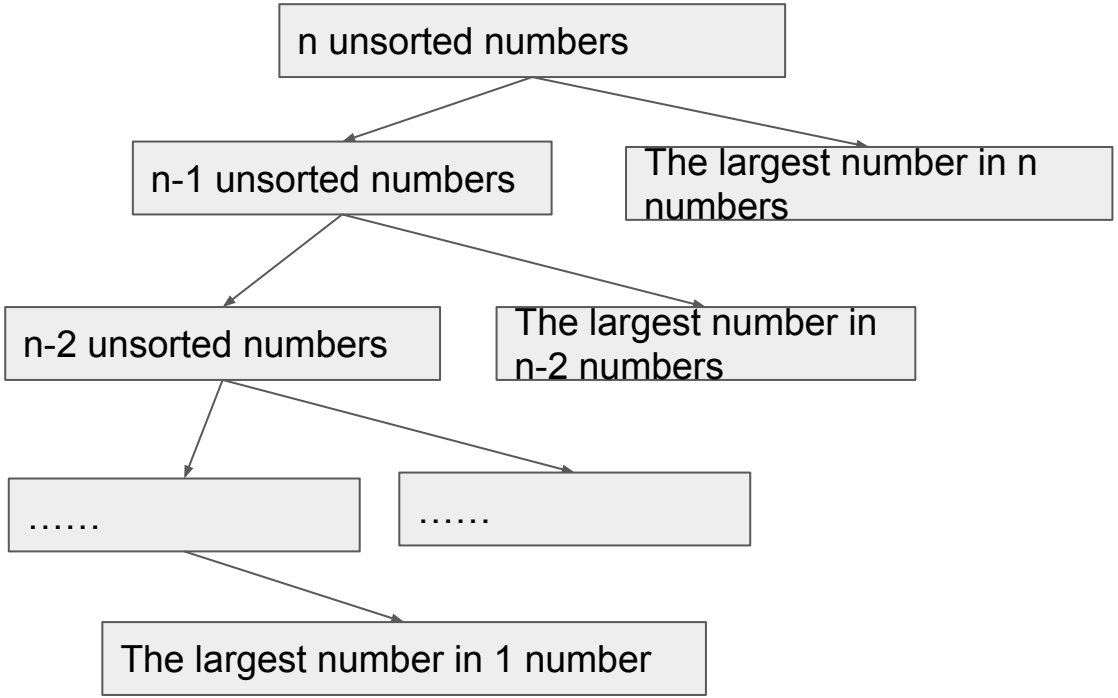
Let a = [1, 4, 2, 6, 3], in pancake sorting, you are only allowed to change the position of an element by slip-flip. For example, if we want to move 6, we can only move it to the beginning of a, and, after the moving, a = [6, 2, 4, 1, 3]. We can move 6 to the end of a by another slip-flip, then, a = [3, 1, 4, 2, 6].

Pancake sort analysis



1. Find the largest element in the unsorted part
2. Flip it to the top (i.e., rotate the sublist along its center line)
3. Flip it to the bottom
4. Repeat step 1 to 3, stop until the whole list is sorted

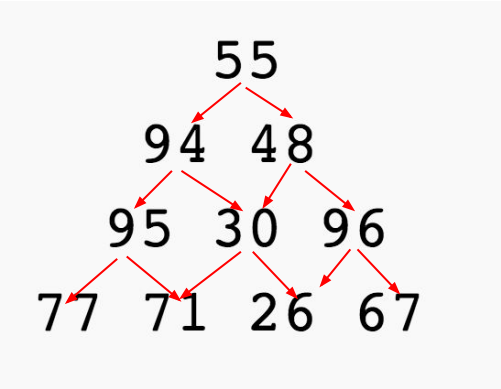
Pancake sort: use recursion



- The solution can be represented by a tree
- The sorting process is a preorder traversal

2. Maximum sum descent

There are positive integers in a triangle; a walk from a node can only go to one of its two child nodes. For example, from the number 48 in the 2nd level, you can only go to 30 or 96 in the 3rd level. A descent is a path from the top to the bottom, for example, 55→94→30→26 is a descent with the sum is 205. **Goal:** find the largest sum of all descents from the root to the base.



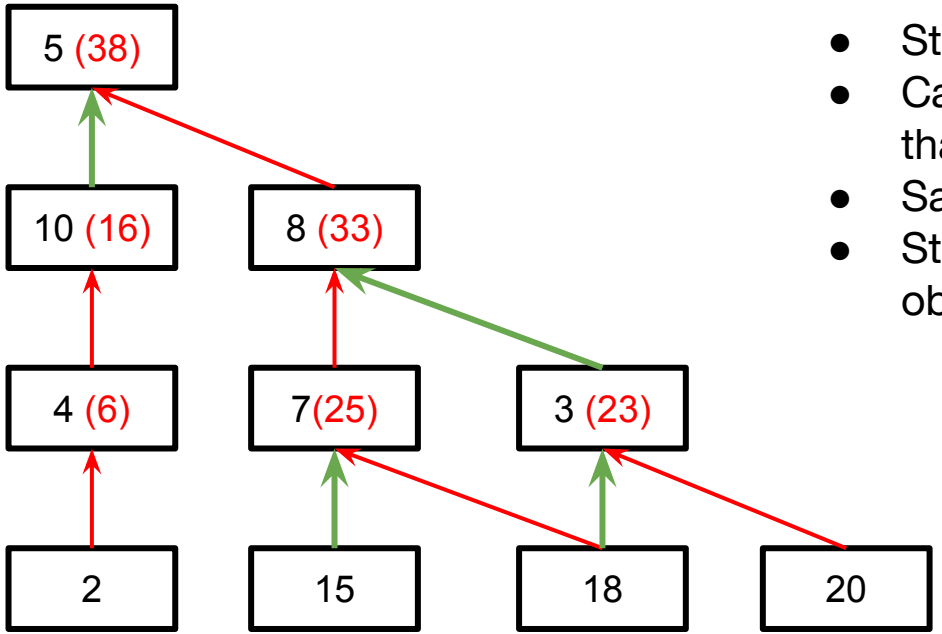
- One descent is (55 -> 94 -> 30 -> 26) = 205
- Maximum descent is 321 (55 -> 94 -> 95 ->77)

```
In [27]: run maxsum.py
triangle --
[17]
[15, 8]
[5, 10, 8]
[16, 6, 10, 12]
[19, 10, 5, 15, 12]

maximum sum --
[17]
[32, 25]
[37, 42, 33]
[53, 48, 52, 45]
[72, 63, 57, 67, 57]
```

Greedy, Dynamic programming? Buttom-up, recursion?

Maximum sum path: analysis

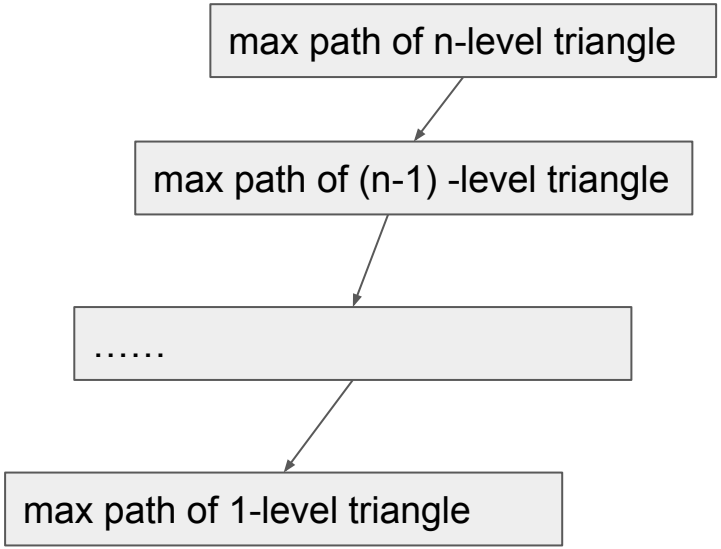


- Start from the bottom
- Calculate the sum of a node and nodes that it connects to.
- Save the larger sum
- Stop when reach the top, and the sum obtained is the maximum value

It can be solved by recursion as well, can you do it?

- Recall the example of picking the fewest bills

Maximum path: recursion



- To know the max path of a triangle of n levels, we need to know the max path of that of the (n-1) levels
- The solution is intuitive
- Since the way to find the max path of each level is the same, we can use recursion.
- It is a postorder traversal.

3. Number placement

Give a list of n numbers, and a list of n-1 preset inequality sign. The task is to insert the numbers so that the inequality holds. **Goal:** insert the numbers so that the **inequality** holds.

Example:

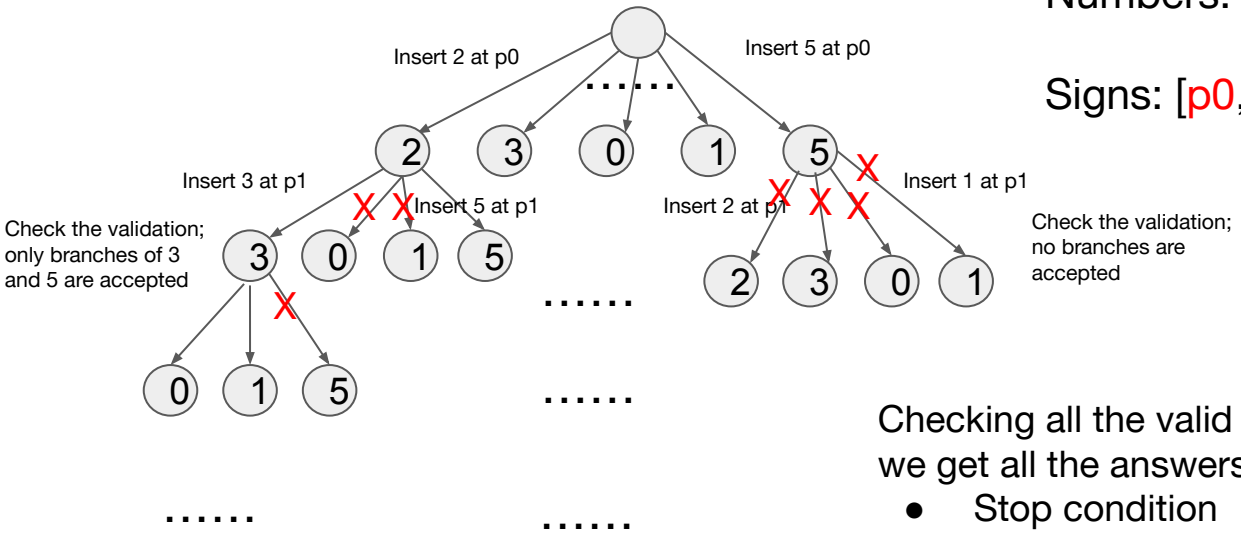
Numbers: [2, 3, 0, 1, 5]; Signs: [‘<’, ‘>’, ‘<’, ‘<’]

Solution: 0 < 5 > 1 < 2 <3.

```
In [35]: run sign_ins.py
[1, '<', 20, '>', 9, '<', 19, '>', 16, '>', 10, '<', 13, '>', 12]
```

Hint: How about sorting the list first?

Analysis: backtracking



Numbers: [2, 3, 0, 1, 5];

Signs: [p0, ‘<’, p1, ‘>’, p2, ‘<’, p3, ‘<’, p4]

Checking all the valid branches along the tree, we get all the answers.

- Stop condition
- How to valid

How about solving it by **reduction**?

4. Tiling problem

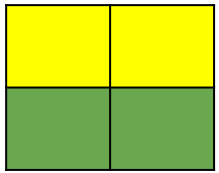
- Given a “2 × n” board and tiles size “2 × 1”, count the number of ways to tile the given board using the 2 × 1 tiles.
- A tile can either be placed horizontally, i.e., as a 1 × 2 tile, or vertically, i.e., as 2 × 1 tile.

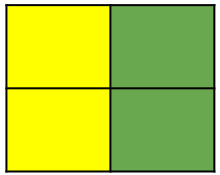
Board

Tile

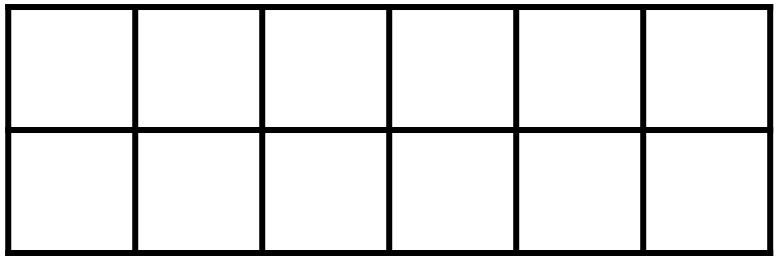
When n=2, we have two ways to cover the board:

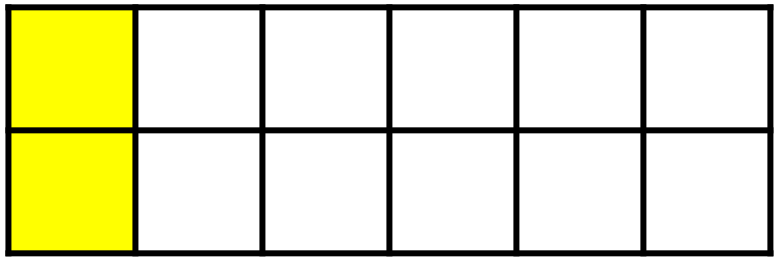
- Two tiles in horizontal
- Two tiles in vertical



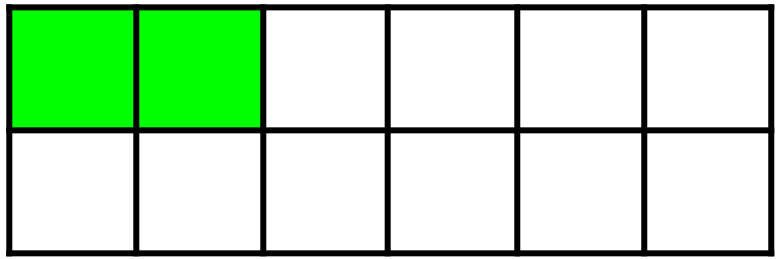


Hint: similar to fibonacci numbers





If the first is vertical, then tiling the rest is to tile a board of (n-1) columns



If the first is horizontal, then tiling the rest is to tile a board of (n-2) columns

5. (Bonus) Cross the bridge (Interview question in Microsoft)

A team of 4 with 1 torchlight are crossing a bridge at night, the bridge allows 2 people to pass maximum. To cross the bridge, one must use the torchlight.

They have different speed, takes 1, 2, 5, and 10 minutes to cross one way, respectively.

- Question 1: what's the minimum total time for the team to cross?
- Question 2: there is a solution to do it with 17 minutes, can you find it?
- Question 3: suppose there are *n* people, what's the solution now?

https://en.wikipedia.org/wiki/Bridge_and_torch_problem

6. (Bonus) Maximize ‘A’s from Leetcode 651

There is a keyboard which only provides **four** keys:

- **Key 1:** (A): print an A on the screen.
- **Key 2:** (Ctrl + A): select the whole screen.
- **Key 3:** (Ctrl + C): copy selection to buffer.
- **Key 4:** (Ctrl + V): print buffer on screen appending it after what has already been printed

Question: Now you press the keyboard n times (with the above 4 keys), find out the maximum number of ‘A’ you can display on screen

6. (Bonus) Maximize ‘A’s from Leetcode

Example 1:

Input: n = 3

Output: 3

Explanation: we can input A A A

Example 2:

Input: n = 7

Output: 9

Explanation: we can input A A A CtrlA CtrlC CtrlV CtrlV

Hint:

- 4 options: press ‘A’, ‘ctrl+A’, ‘ctrl+C’, or ‘ctrl+V’;
- States: the number of press allowed (‘n’), the ‘A’s on the screen (‘num_a’), the ‘A’s in the buffer (buff)
- How do the states change with options? (Note: ctrl+A and ctrl+C always use consecutively, i.e., if one presses ctrl+A, then, for the next key, he/she must press ctrl+C.)