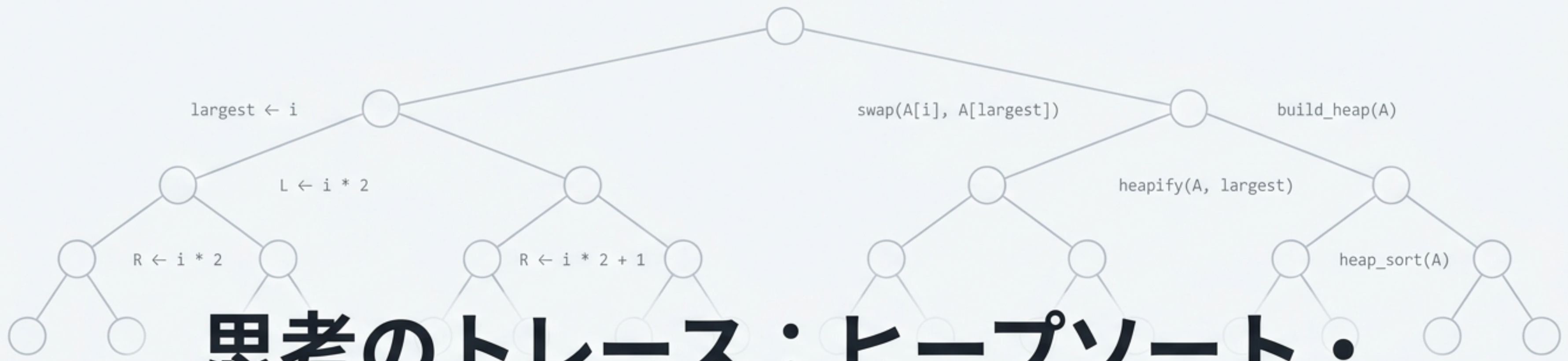




思考のトレース：ヒープソート・ アルゴリズムの完全解剖

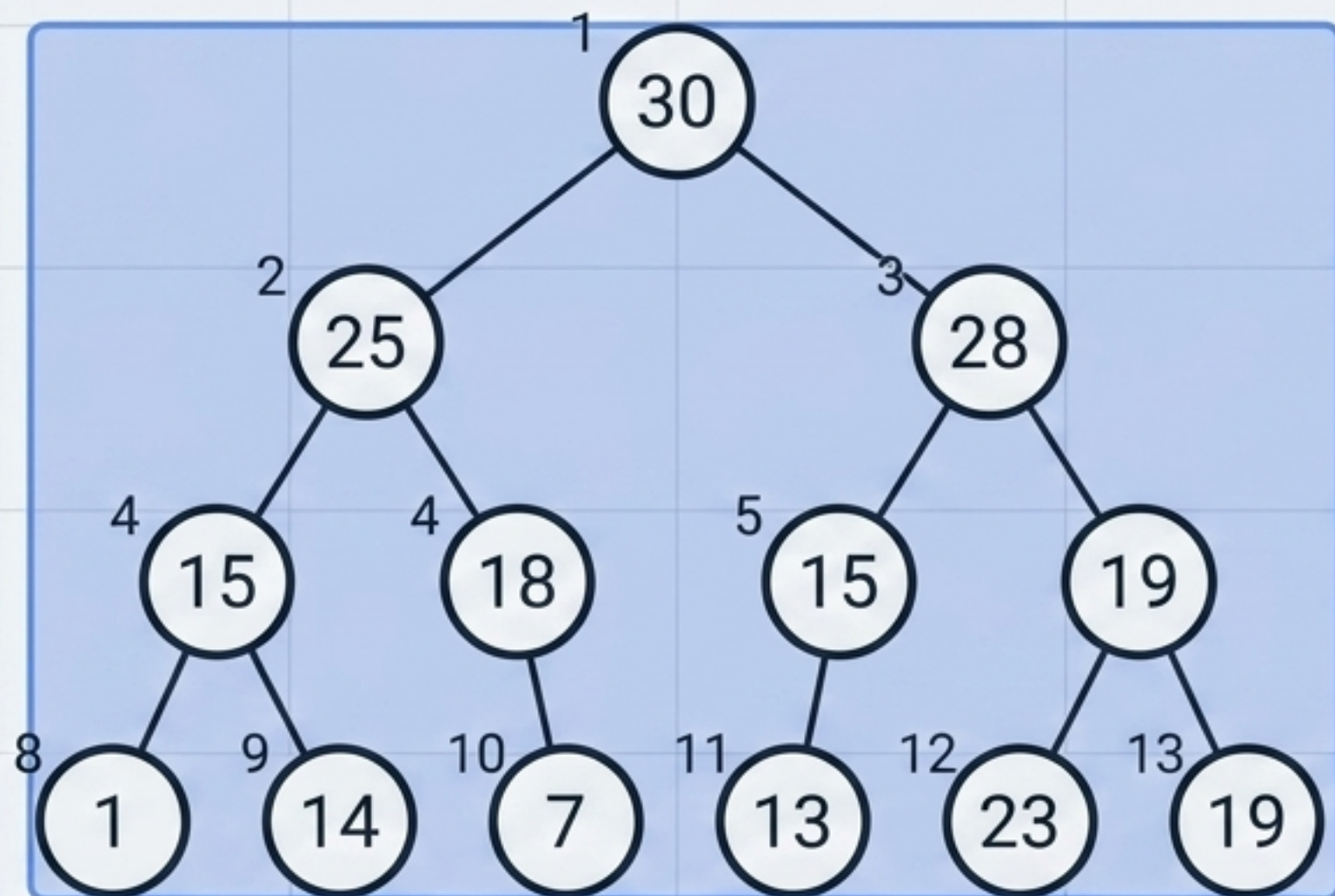
ソフトウェア開発技術者 午後I問5 — 解答への「視点」を手に入れる

コードを読むな、構造を観よ。

ヒープを支配する「2つの絶対ルール」

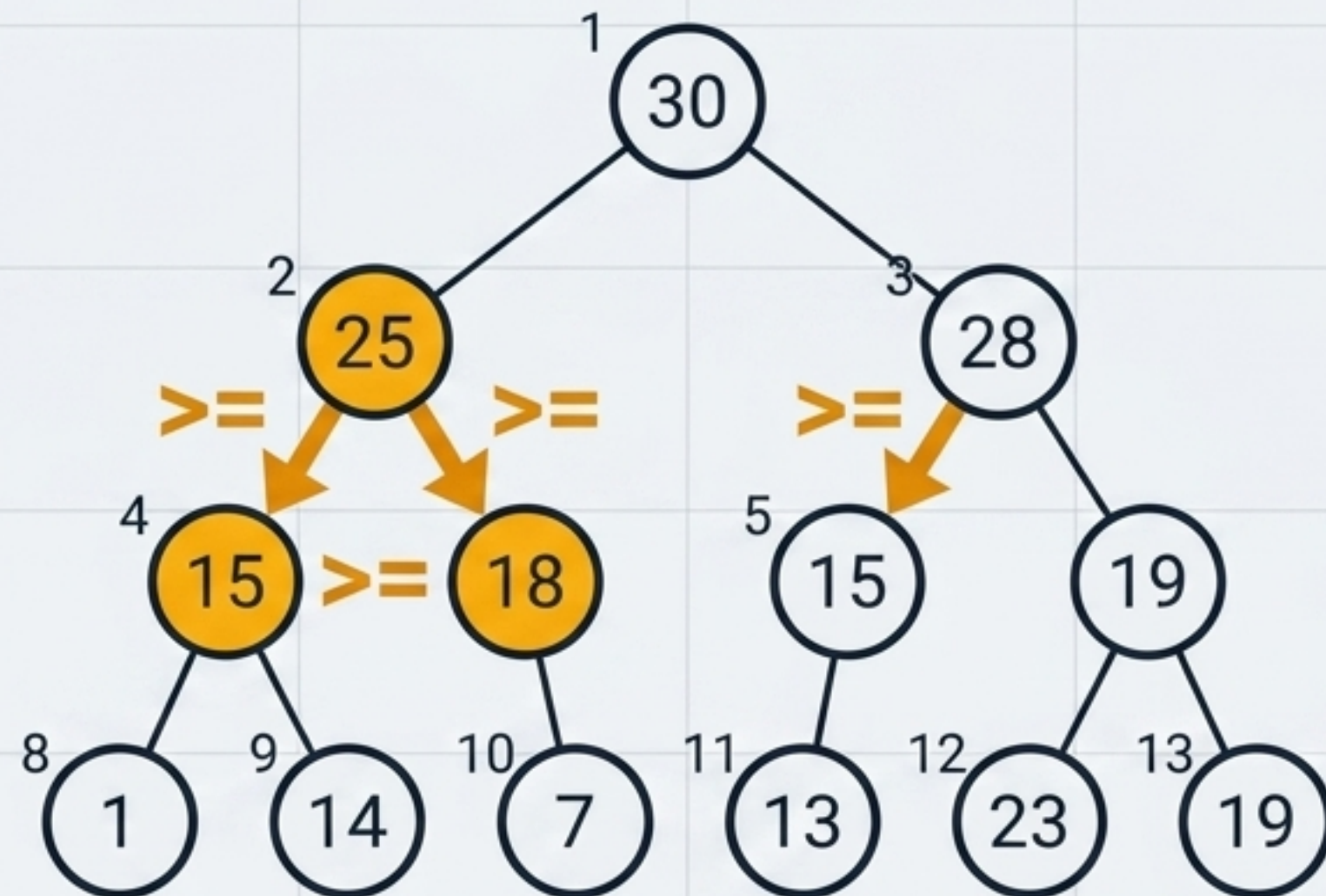
この形状と大小関係が崩れた時、アルゴリズムが発動する。

条件1: 形状のルール



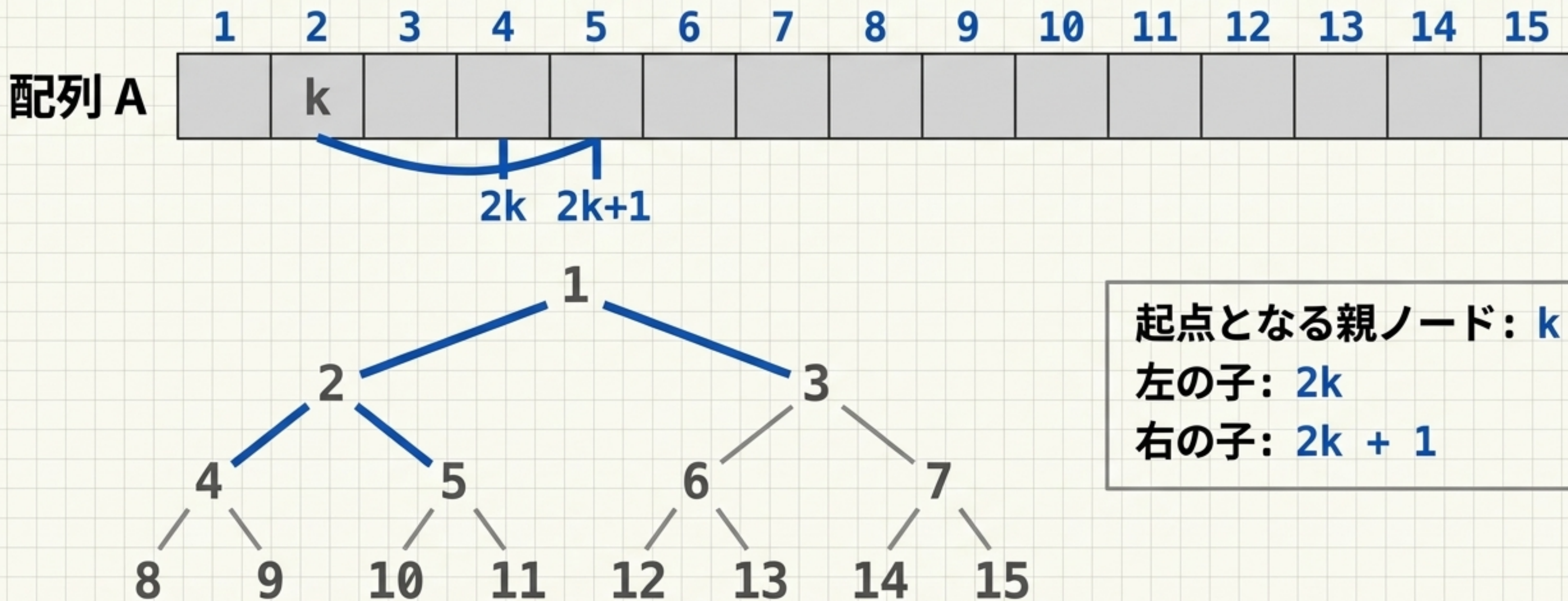
完全二分木。葉の深さは揃い、左から詰められる。

条件2: 価値のルール



親ノードの値 $\geq$ 子ノードの値。

1次元の配列は、計算式一つで2次元の木構造へ変換される



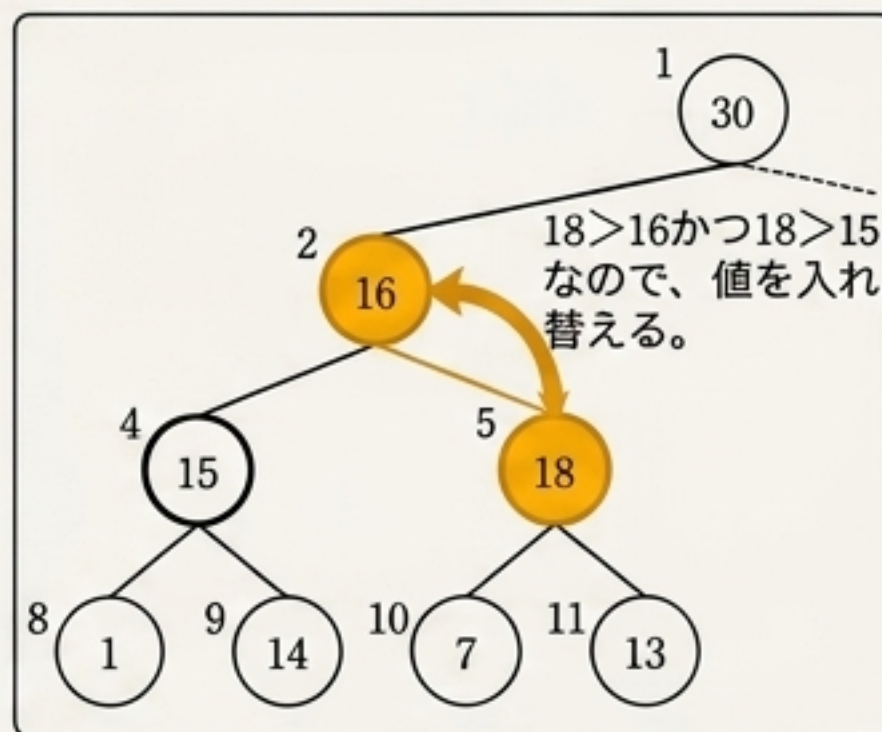
配列の添え字操作だけで、木構造の上下左右を自在に移動できる。
これがヒープソートの美しさである。

ミッション：違反したノードを沈めよ

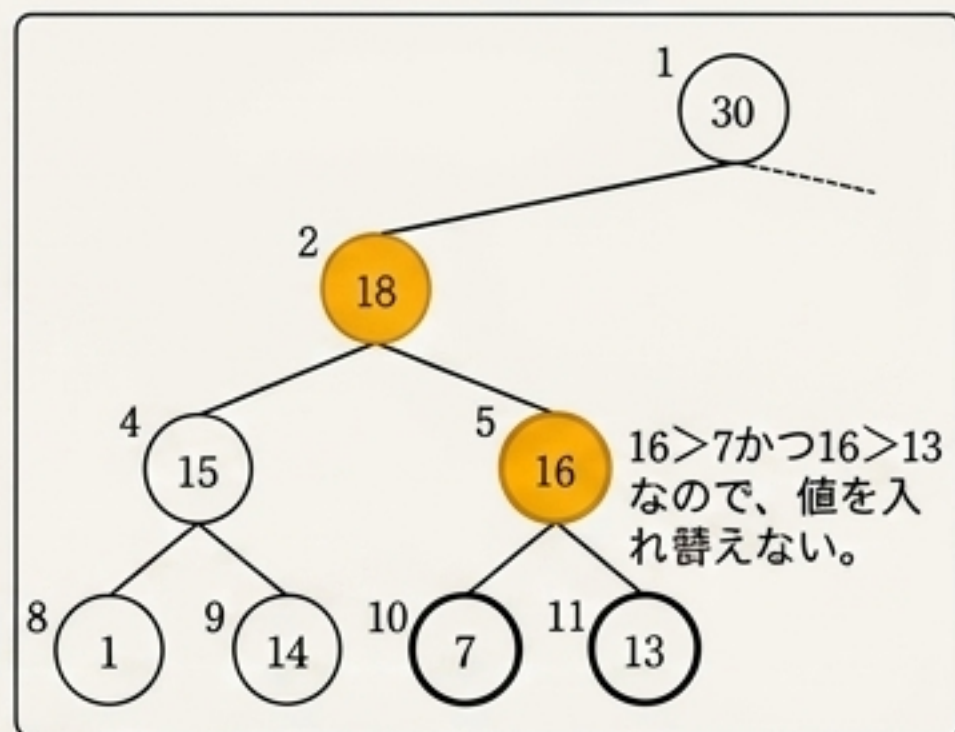
目的: 条件を満たさない親を、正しい位置まで「沈める」操作。

```
heap_correct(A, i)
  largest ← i
  L ← i * 2
  R ← i * 2 + 1
  if (L が LENGTH 以下である かつ A[L] が )
    largest ← L
  endif
  if (R が LENGTH 以下である かつ )
    largest ← R
  endif
  if (largest が i と等しくない)
    exchange(A, i, largest)
    heap_correct(A, largest)
  endif
```

Before (Fig 2)



After (Fig 3)



注目ポイント: 親・左の子・右の子による「三つ巴の戦い」。

最終的に一番上 (largest) に立つべきは、この3つのうち「誰」か？

【設問1への視点】 勝者を決める条件式を読み解く

```
if (LがLENGTH以下である かつ A[L]が [ ア ] )  
    largest ← L
```

思考のプロセス：

直前の処理で、変数 largest
には何が代入されているか？
(現在の「仮の勝者」は誰か？)

この if 文の内部に入ると、勝
者 (largest) が L (左の子)
に更新される。

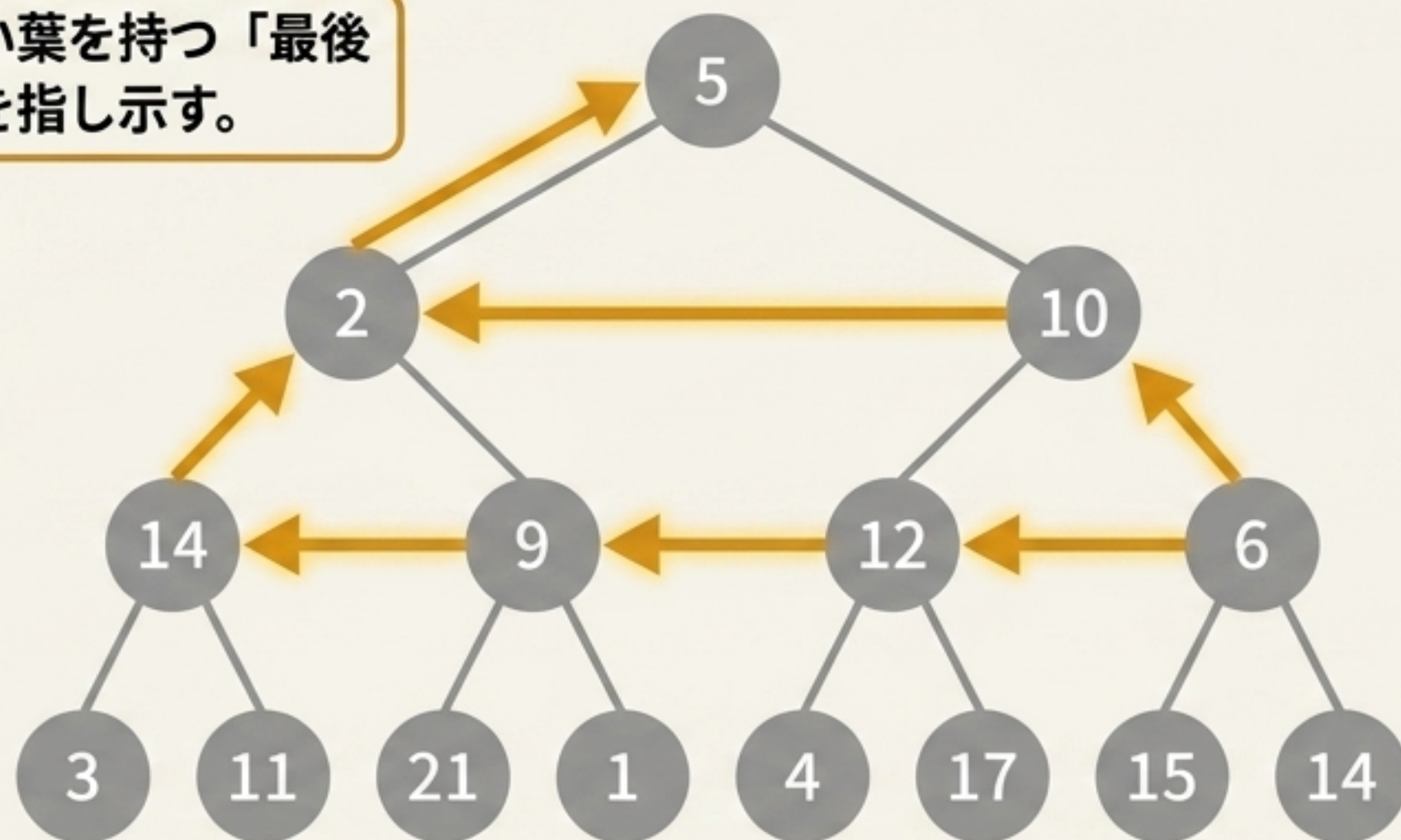
つまり、A[L] が「誰」に勝っ
た時に、この更新が発生するべ
きか？

プログラムは常に「現在の暫定チャンピオン」と「新たな挑戦者」を比較している。[イ] も全く同じ論理で導ける。

全体を再構築する軌跡（Bottom-Up Approach）

```
heap_make(A)
  n ← floor(LENGTH / 2)
  for(i を n から 1 まで 1 ずつ減らす)
    heap_correct( ウ , エ )
  endfor
```

意味：一番深い葉を持つ「最後の親ノード」を指し示す。

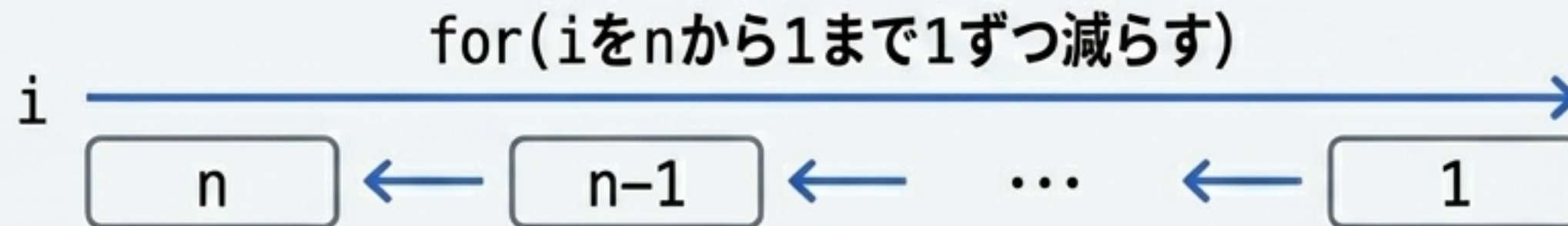


葉ノード単体は最初から条件を満たしている。
だからこそ、一番下層の「親」から順番に
上へ向かって修正（correct）をかけていく。

【設問2への視点】 ループ構造は何をコントロールしているか？

変数／サブルーチン	変数の意味／サブルーチンの機能
exchange(A, x, y)	配列Aのx番目とy番目の要素の値を入れ替える。

```
heap_correct(  ,  )
```



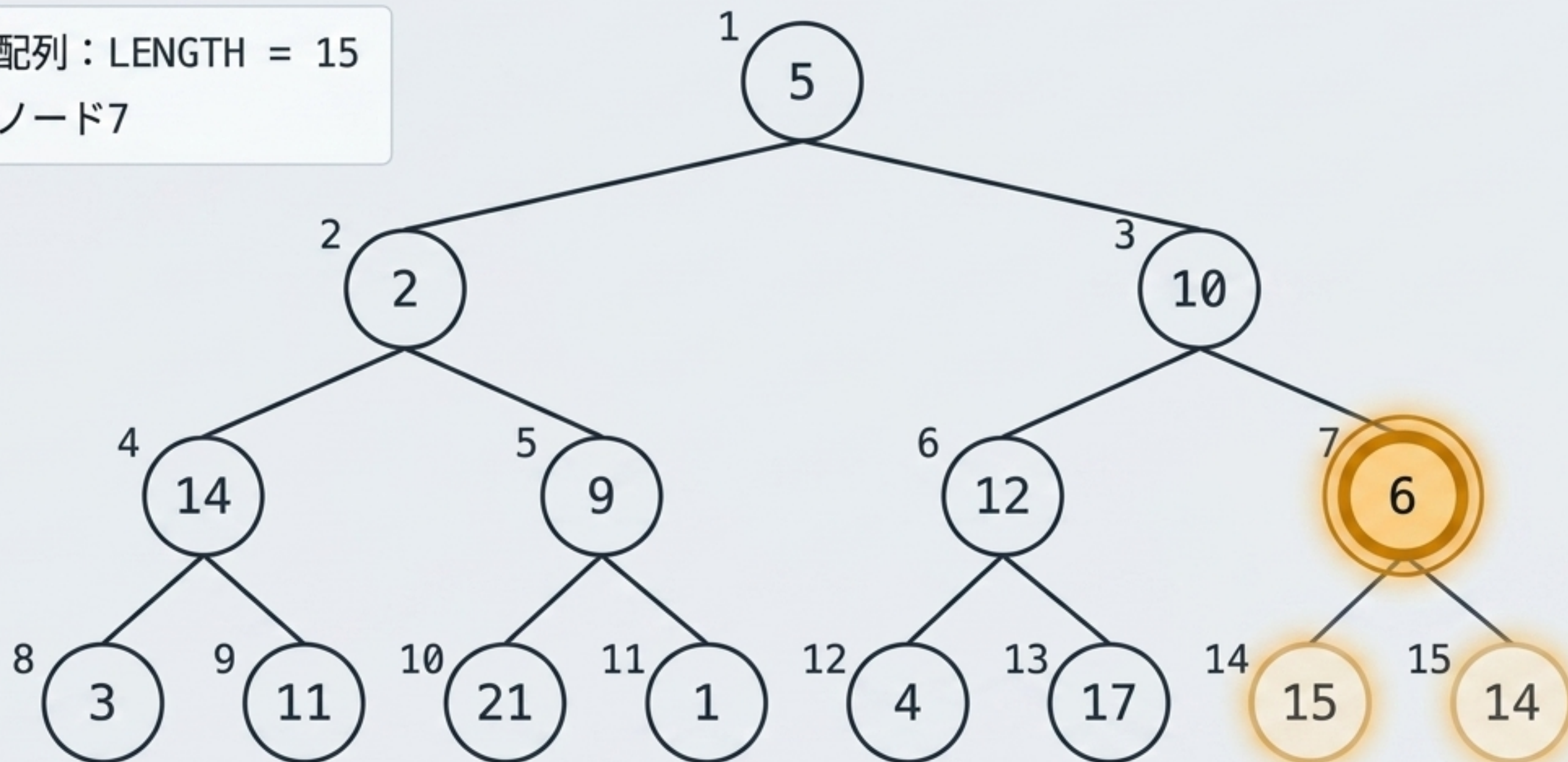
思考のプロセス：

1. heap_correct を呼び出すために必要な引数は何か？（表を確認しよう）
2. このループ構文 for(iをnから1まで1ずつ減らす) は、何を起点に処理を進めているか？
3. 今、どの「データ構造」に対して、どの「ノード番号」をターゲットに実行しようとしているのか？

思考の実験室：シミュレーション・トレースを開始せよ

ターゲット配列：LENGTH = 15

開始地点：ノード7



ここからは頭の中だけでは解けない。ノード7、6、5、4と順番に値を入れ替えながら、木の状態がどう変化するかを確実に追跡する準備をしよう。

【設問3(オ)への視点】 ヒープの極限状態を想像する

Root $\geq$ Level 2 ¹
 $\geq$
Level 2 $\geq$ Level 3
 $\geq$
Level 3 $\geq$ Level 4

思考のプロセス:

- すべての heap_correct が完了した時、この木は「完全なヒープ」になっている。
- 「親 $\geq$ 子」というルールを一番下から一番上まで連鎖させると、頂点(根)には必ずどのような「性質」を持った値が来る運命にあるか？
- 木全体を見渡し、一番強い値を見つけよう。

トレースを最後までやり切らなくても、木全体の数字を見渡せば「ある1つの値」が自動的に浮かび上がる。

【設問3(キ)への視点】 再帰の「連鎖」を正確にカウントする

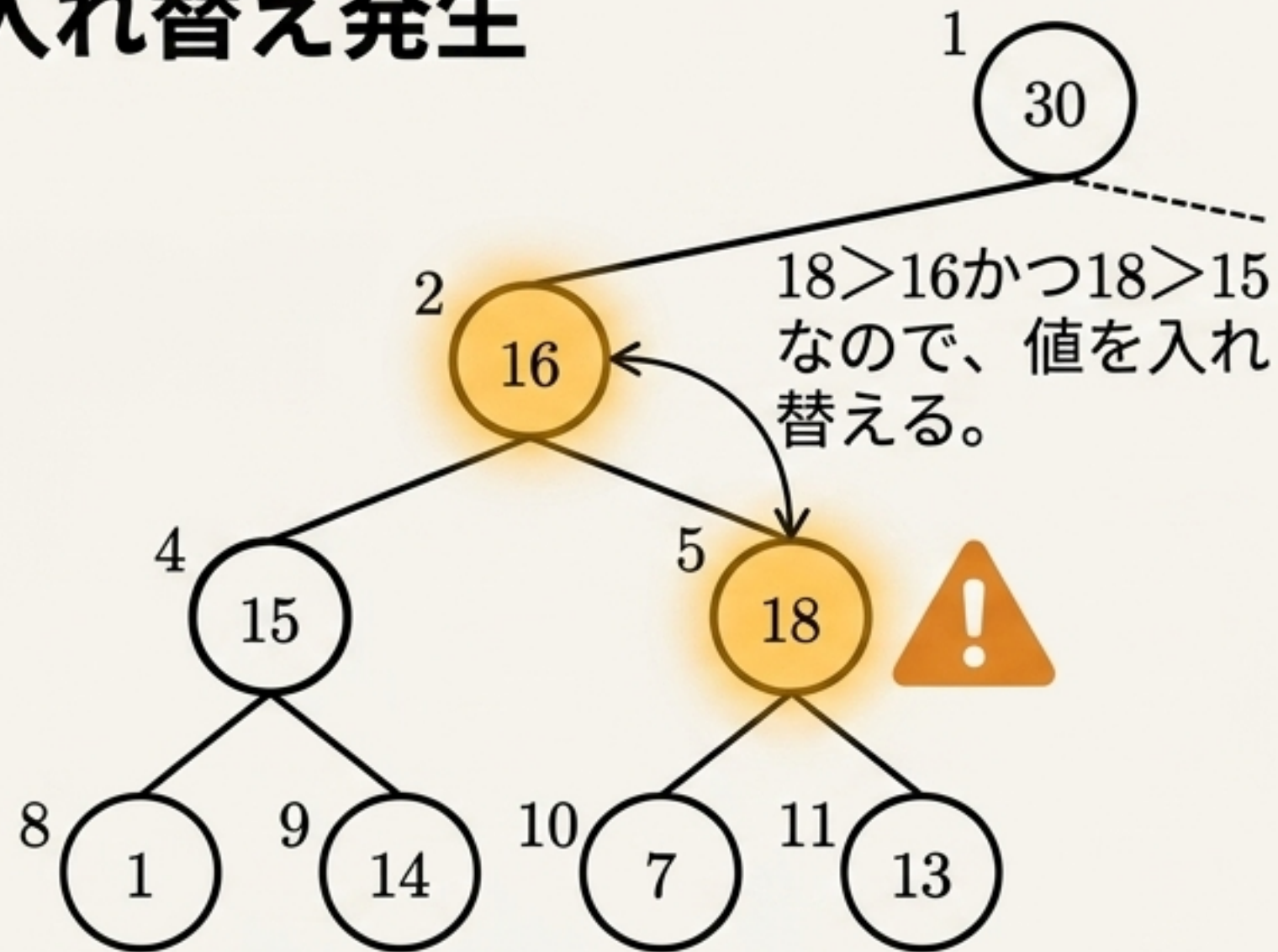
⚠ 再帰の絶対条件: 「値の入れ替え (Swap) 」が発生した時のみ、その先 (子ノード以下) へ連鎖する。

トレース・チェックリスト			
ターゲットノード	子ノードとの勝負	入れ替え発生？	再帰の連鎖？
[Node 7]	左右の子(14, 15)と勝負。	入れ替えは起きる？	その先の子はいる？
[Node 6]	左右の子(12, 13)と勝負。	勝者は誰？	
[Node 5]	左右の子(10, 11)と勝負。		
[Node 4]	左右の子(8, 9)と勝負。	入れ替えは起きる？	

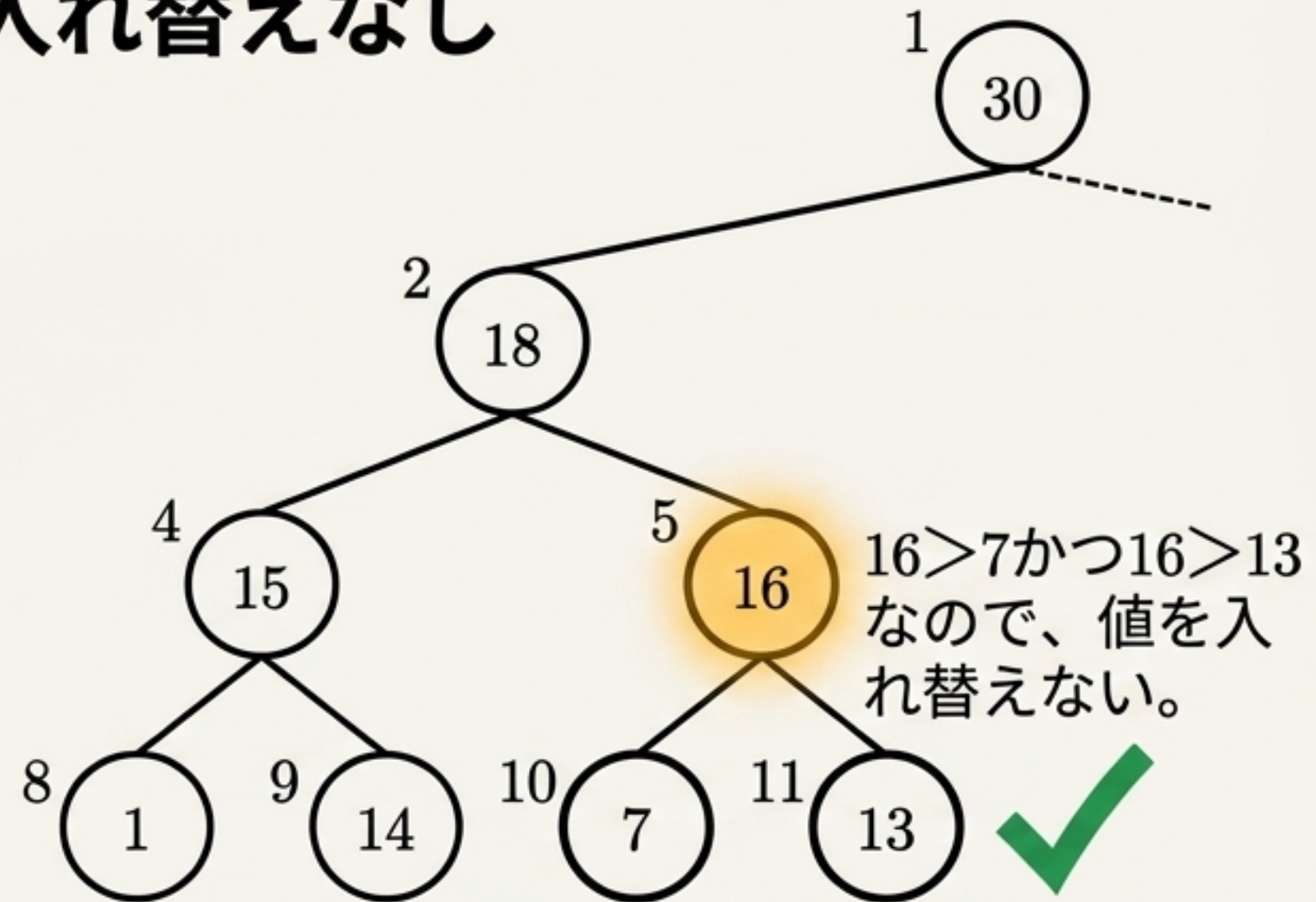
「最初の呼び出し (4回) 」 + 「連鎖した呼び出し (?回) 」の合計を算出しよう。

【設問4への視点】なぜ「追跡調査（再帰）」が必要なのか？

入れ替え発生



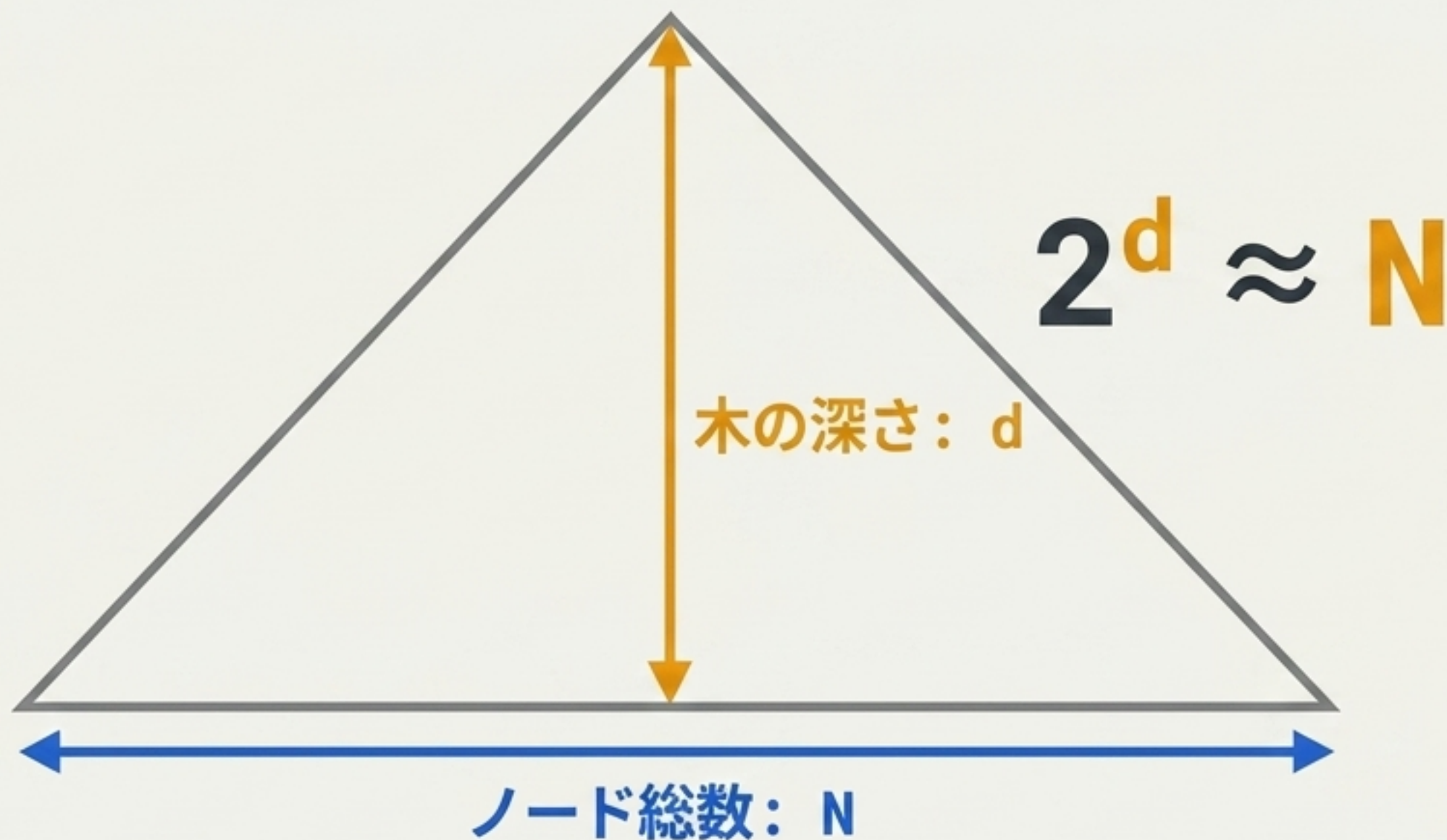
入れ替えなし



思考のプロセス：

- ・ノードが下に沈んだ（入れ替わった）直後、そのノードの「新しい部下たち（子ノード）」との間に、ヒープの条件は保証されているだろうか？
- ・子が親を上回っていた場合（＝入れ替えが必要な場合）、新しい配置先でも再度ルール違反がないか確認し続けなければならない。

計算量 (Big-O) の直感的なイメージ構築



木構造の特性:

ノード数 N が倍になっても、深さ d は1段しか増えない。

直進経路:

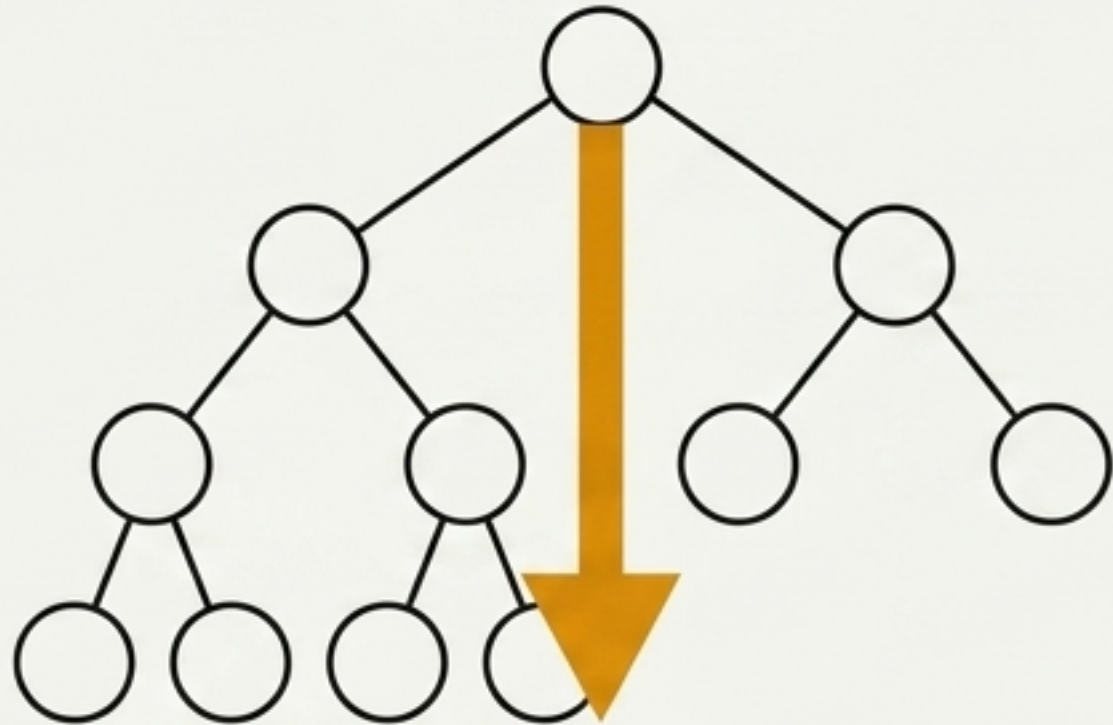
一番上から一番下まで真っ直ぐ辿るステップ数は、全体の数 N ではなく、深さに比例する。

Insight:

全ノードを舐めるのか ($O(N)$)、一直線に下るだけなのか ($O(\log N)$) を見極める。

【設問5への視点】 最悪のケースと、最良の初期状態

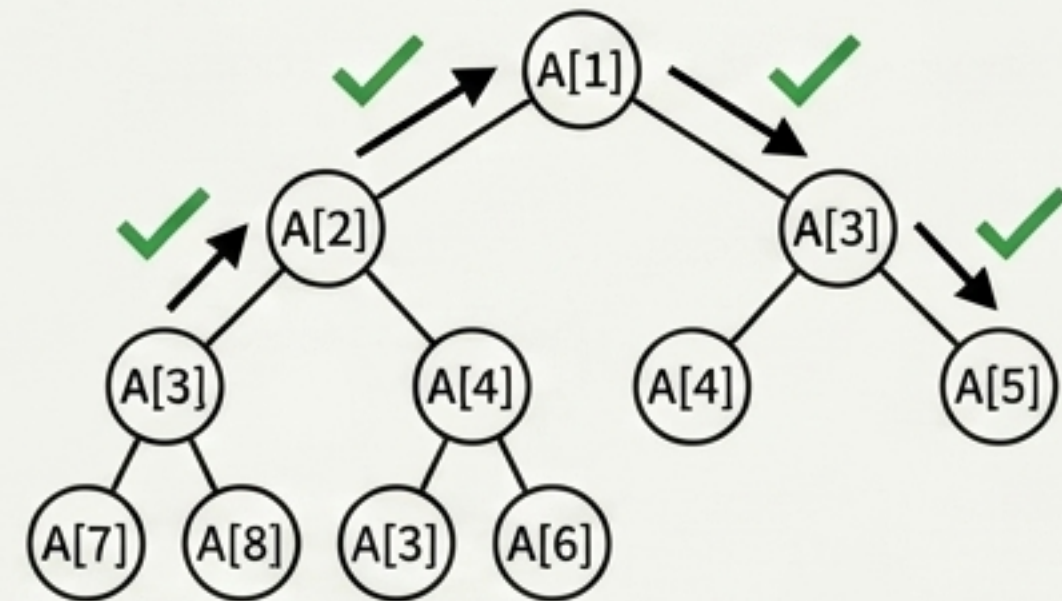
ケース①: 根からの heap_correct



根が一番下まで沈んでいく最悪のケース。
木を上から下へ辿る最大の回数は、前のスライドの
「深さ」と「総数」どちらに依存するか？

ケース②: 降順ソート済み配列での heap_make

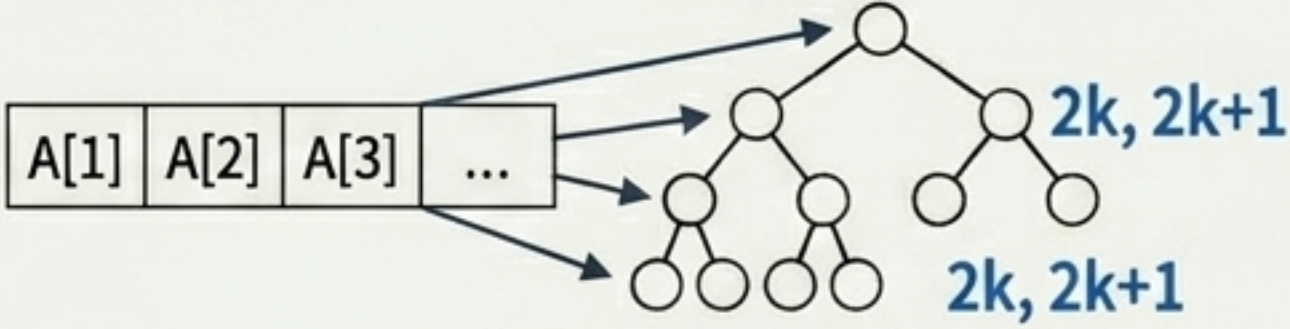
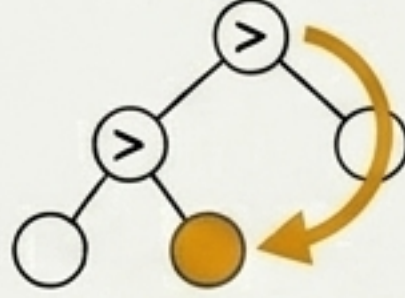
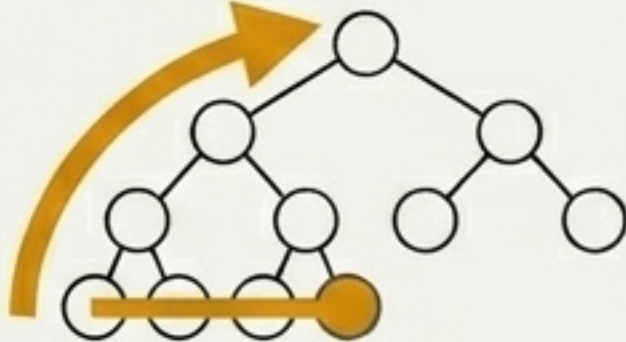
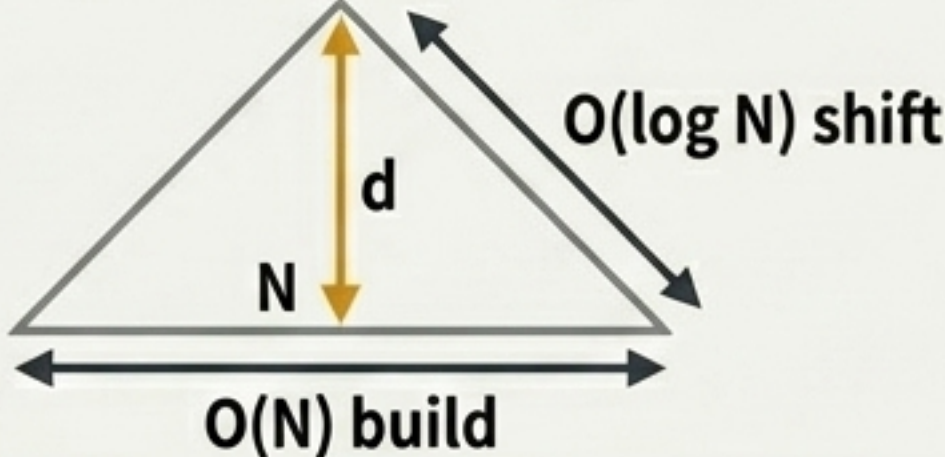
A[1]	>	A[2]	>	A[3]	...
------	---	------	---	------	-----



配列がすでに「降順（大きい順）」に並んでいる時、
木構造の「親 > 子」のルールはどうなっているか？

最初から条件が満たされていれば、**再帰（Swap）は一度も起きない。**
では、**forループの回数**だけで計算量はどうなる？

複雑なアルゴリズムも、局所的なルールの積み重ねに過ぎない

Architecture Map	
Q1. Data Mapping  Array to Tree Indexing	Q2. Local Rule  heap_correct (違反を沈める)
Q3. Global Execution  heap_make (下から上へ再構築)	Q4. Complexity  $O(\log N)$ shift $O(N)$ build

法則を理解し、局所的な動きをトレースし、全体へ抽象化する。
視点を持てば、どんなコードの意図も必ず読み解ける。