

## 2-5 ヒープソート

データ構造のヒープを利用する

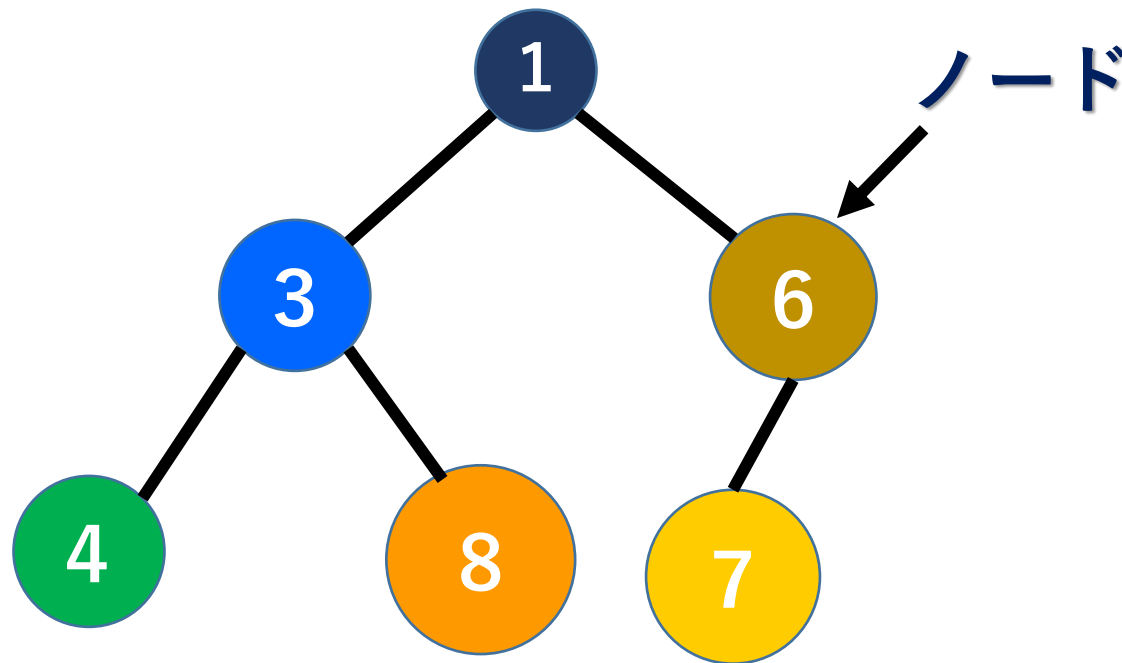


# 1-7 ヒープ

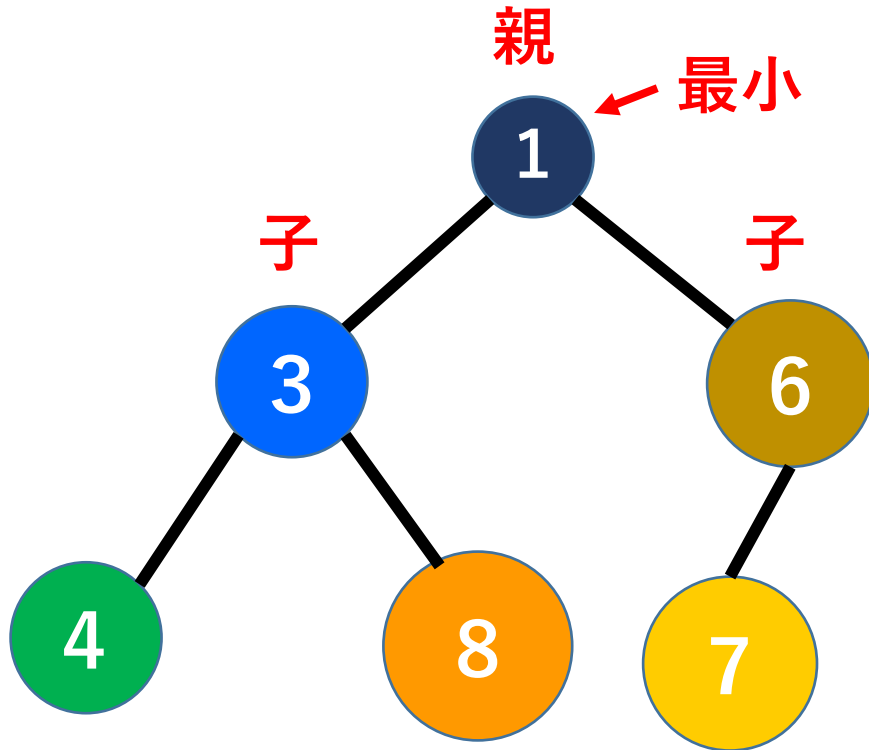
グラフの木構造の1つ

「プライオリティキュー（優先度付きキュー）」を実現するとき  
に使用

データを自由に追加でき、小さいものから取り出すことができる

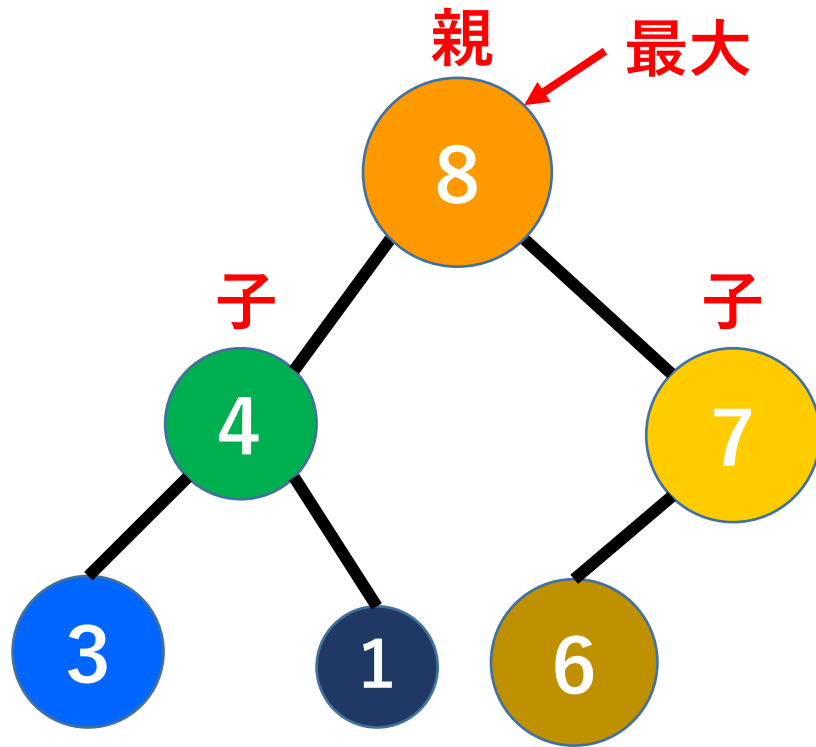


# 昇順ヒープ



1. ヒープの各ノードは最大2つの子どもを持つ
2. ノードは上に詰める
3. 同じ段では左に詰める
4. **子の数字は親の数字より大きい**
5. **一番上の数字が最小**
6. データの追加は、一番下の段に左詰に追加する
7. 一番下の段がすべて詰まっている場合は、新しい段を作り、左から追加

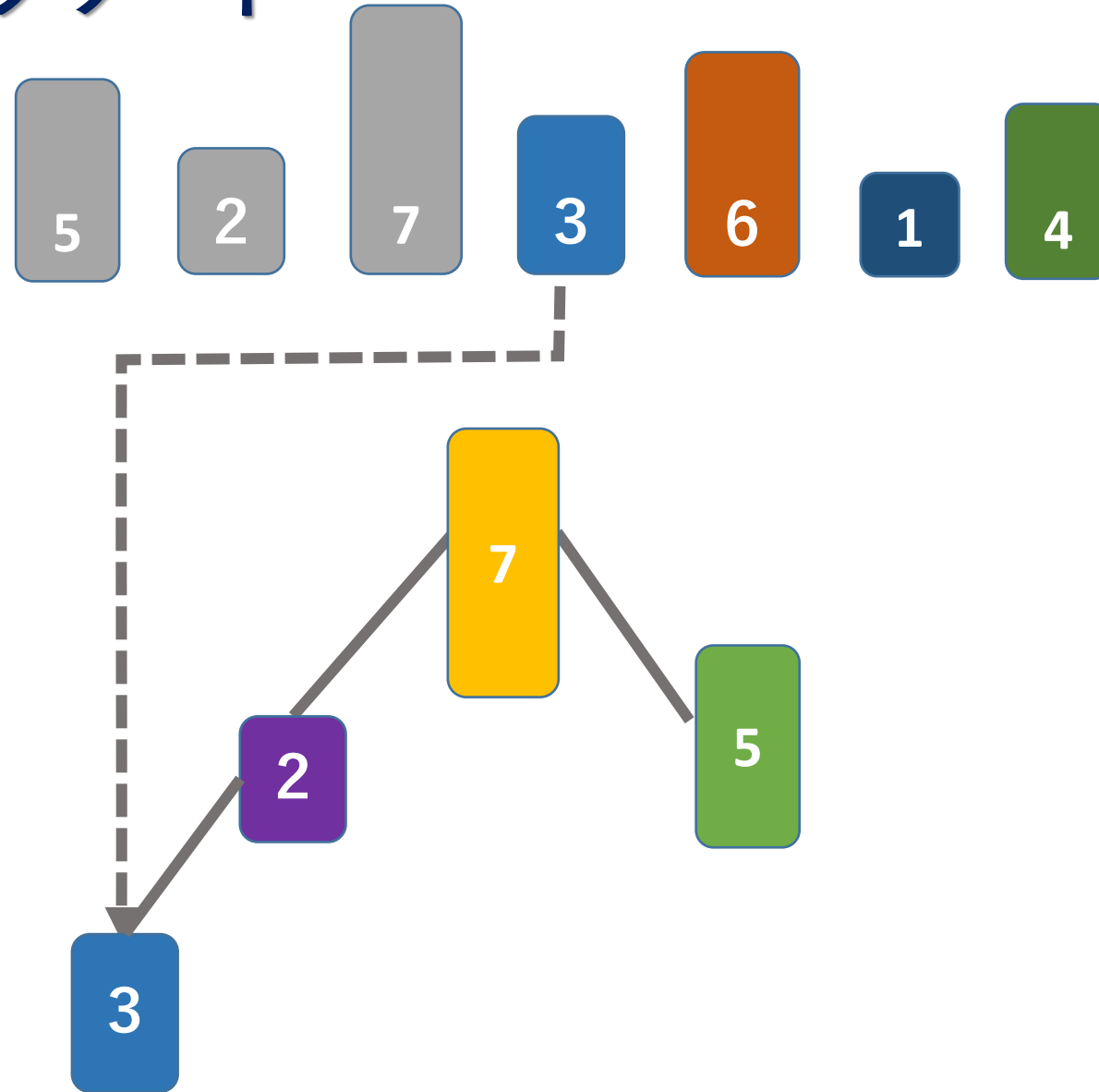
# 降順ヒープ



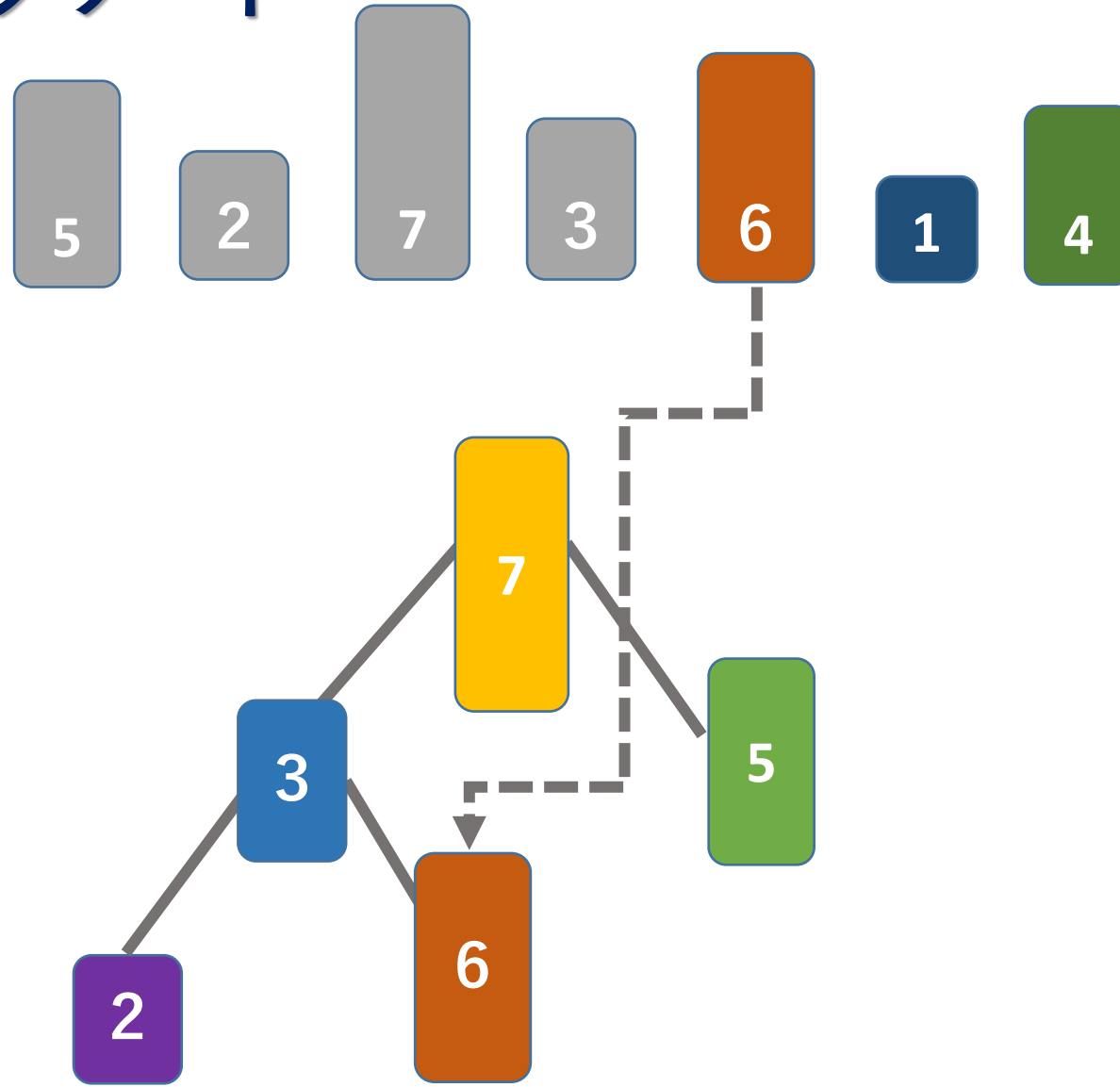
1. ヒープの各ノードは最大2つの子どもを持つ
2. ノードは上に詰める
3. 同じ段では左に詰める
4. **親の数字は子の数字より大きい**
5. **一番上の数字が最大**
6. データの追加は、一番下の段に左詰に追加する
7. 一番下の段がすべて詰まっている場合は、新しい段を作り、左から追加



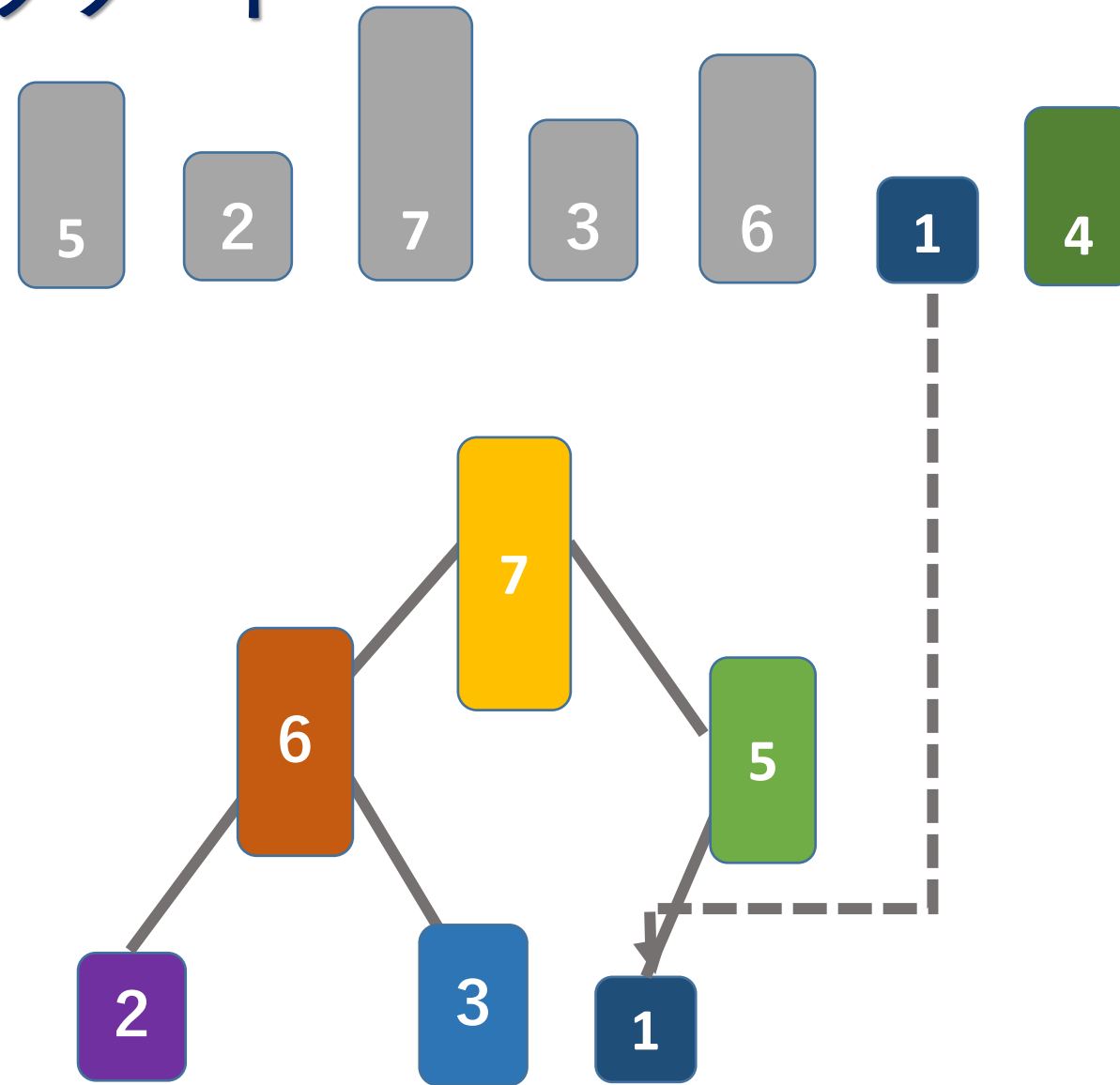
## 2-5 ヒープソート



## 2-5 ヒープソート

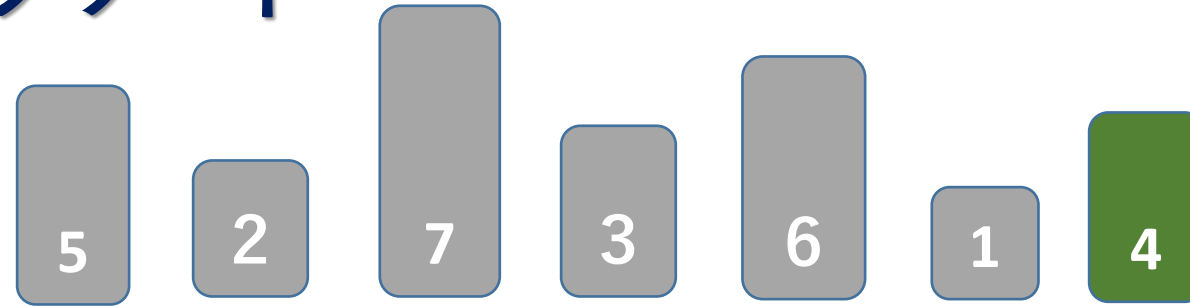


## 2-5 ヒープソート

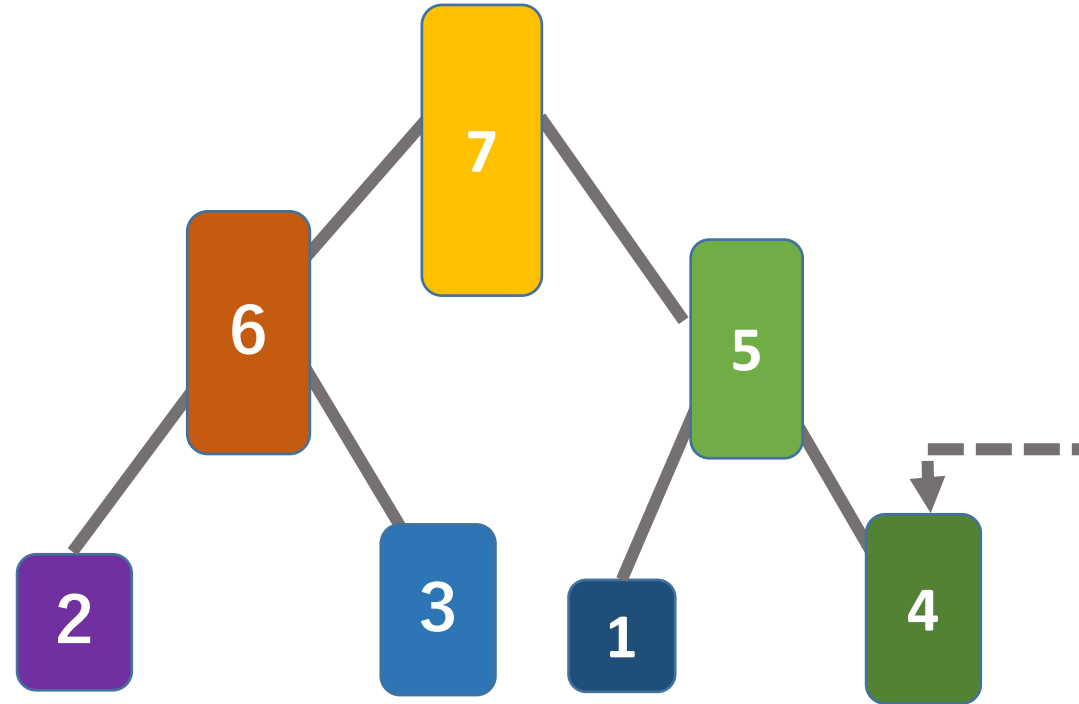




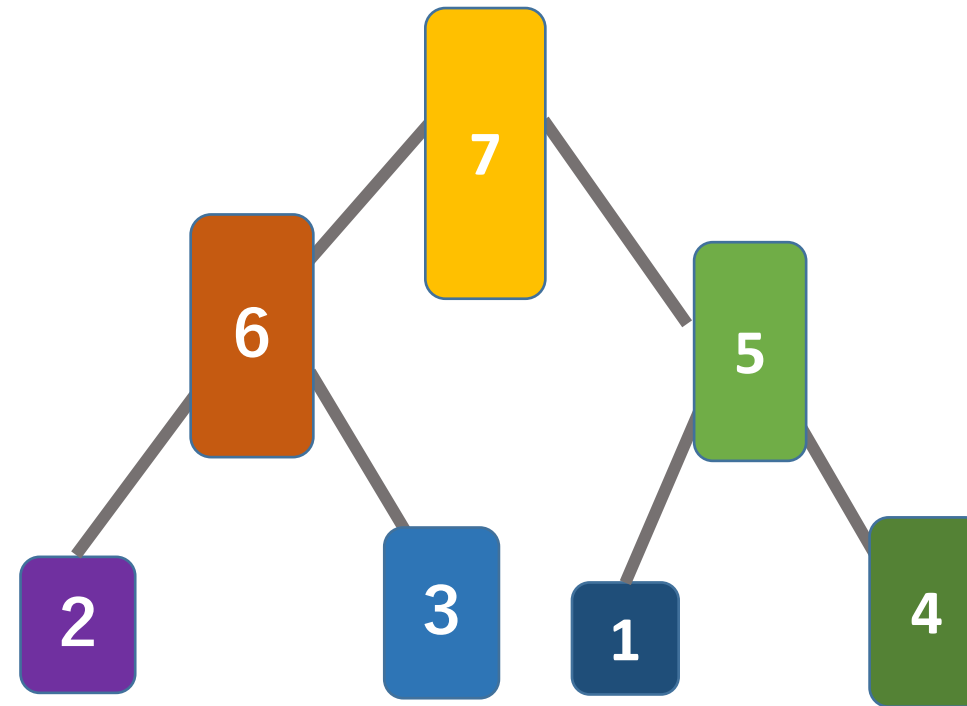
## 2-5 ヒープソート



ヒープ完成！



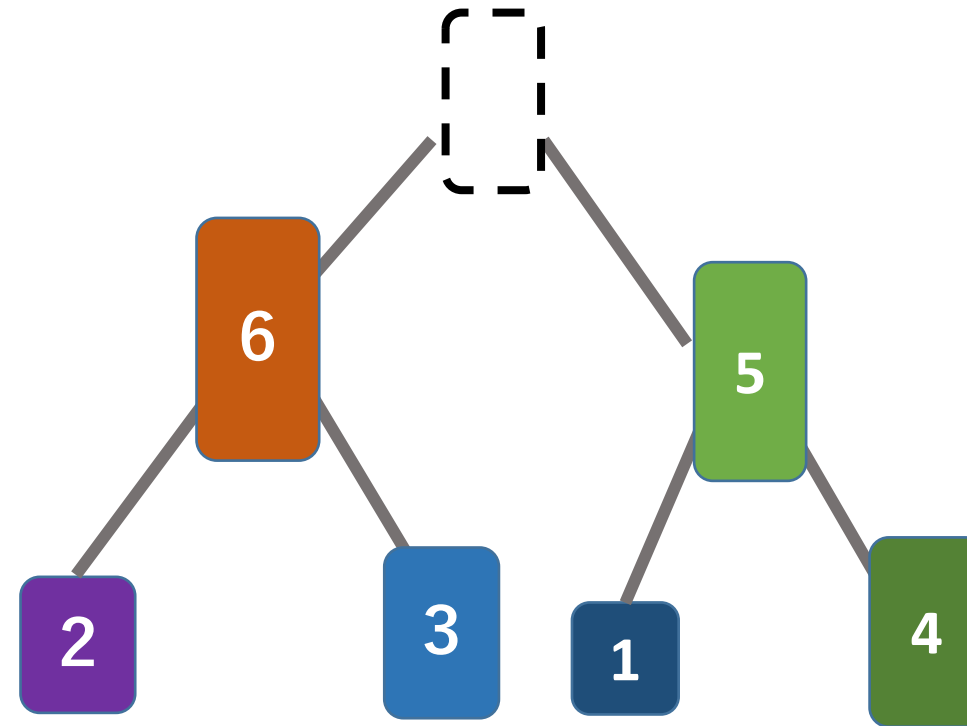
降順ヒープは大きいものから順にデータが取り出される  
一番上の **7** を取り出す



降順ヒープは大きいものから順にデータが取り出される

4 を最上位に移動(一番後ろ)

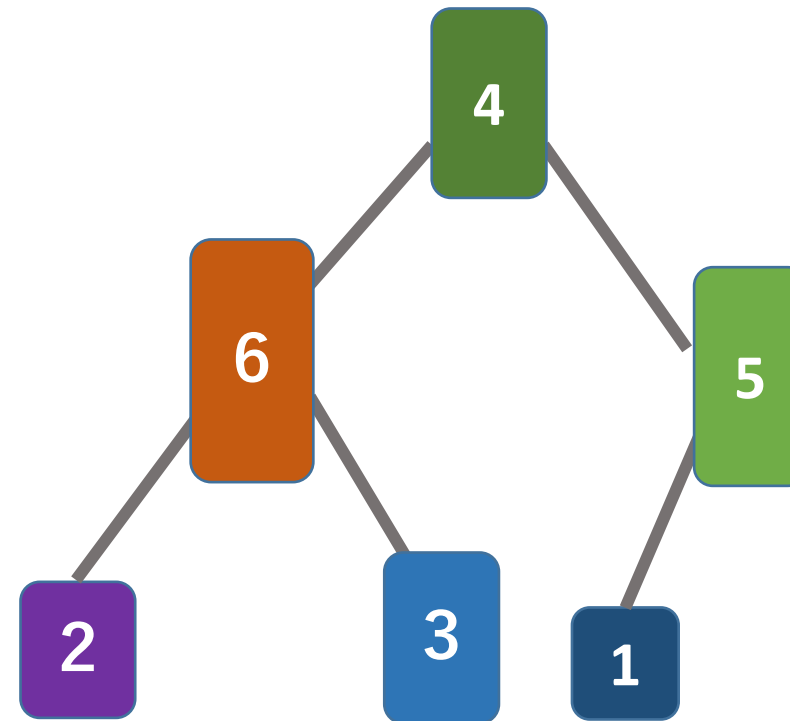
7



降順ヒープは大きいものから順にデータが取り出される

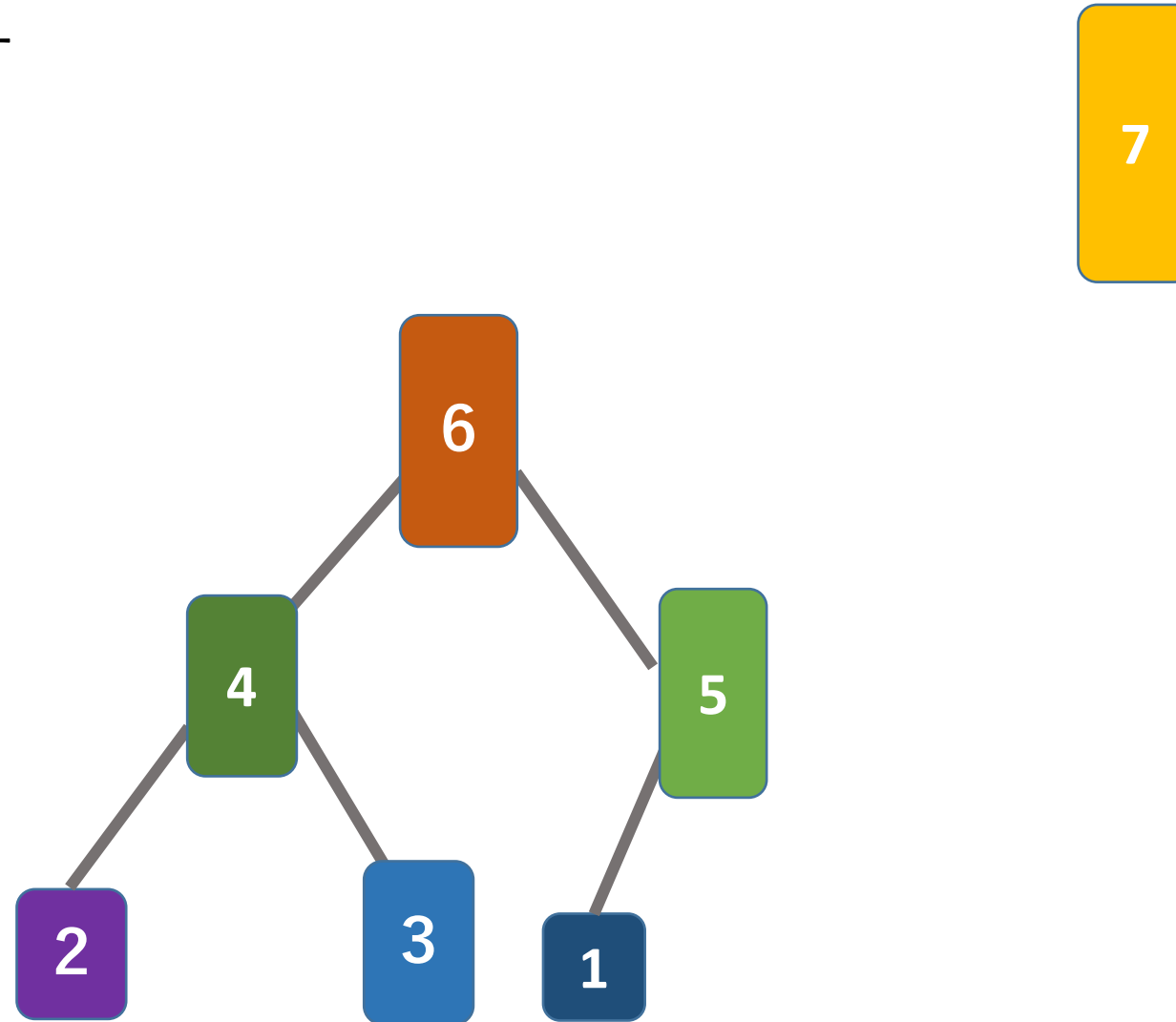
ヒープの再構築  
**6**と**4**の入替え

7



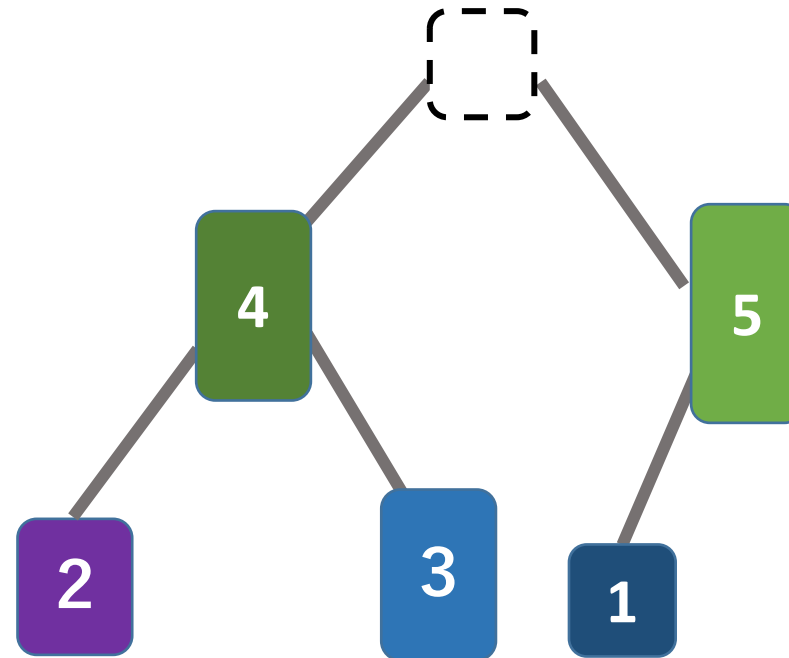
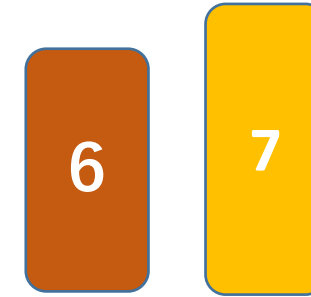
降順ヒープは大きいものから順にデータが取り出される

**6** を取り出す



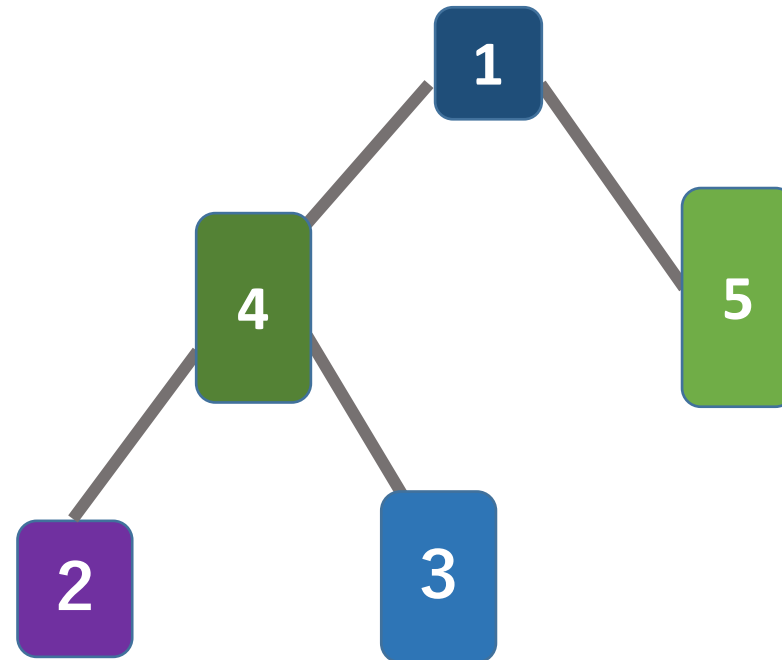
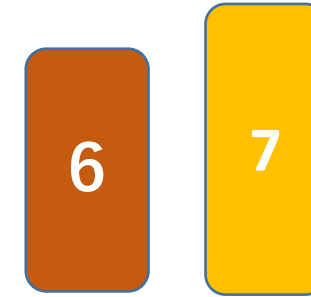
降順ヒープは大きいものから順にデータが取り出される

**1** を最上位に移動(一番後ろ)



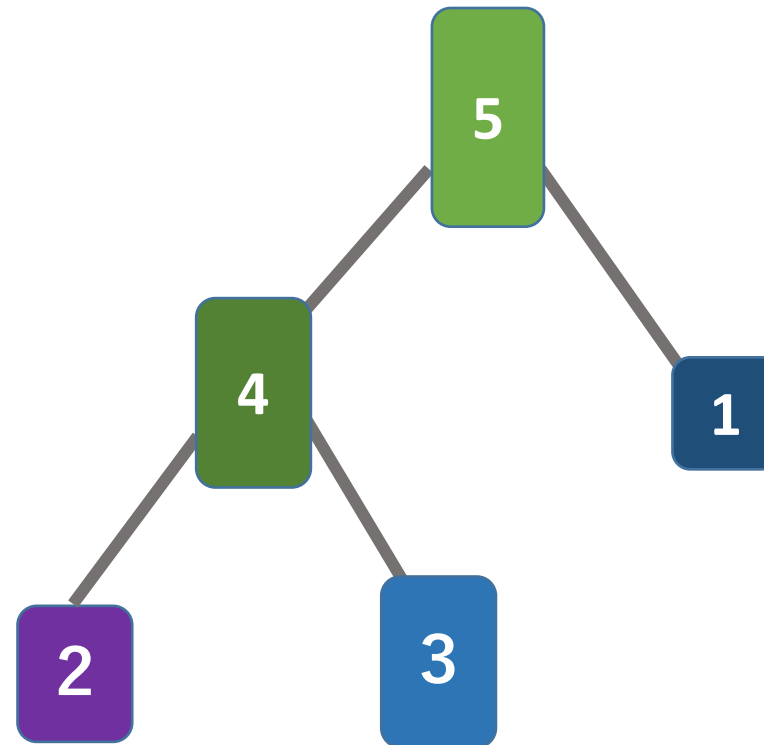
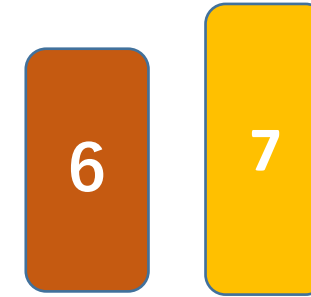
降順ヒープは大きいものから順にデータが取り出される

ヒープを再構築  
**1**と**5**を入替え



降順ヒープは大きいものから順にデータが取り出される

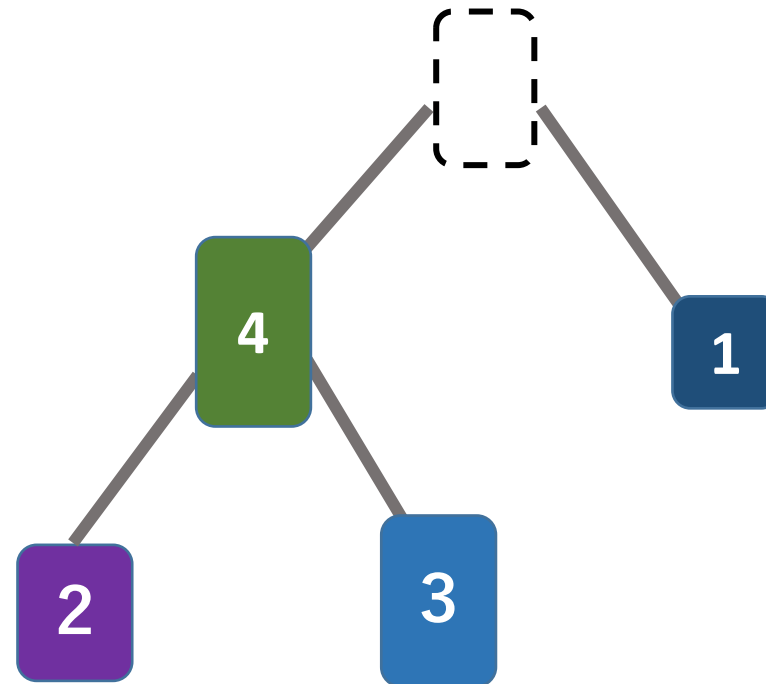
**5** を取り出し





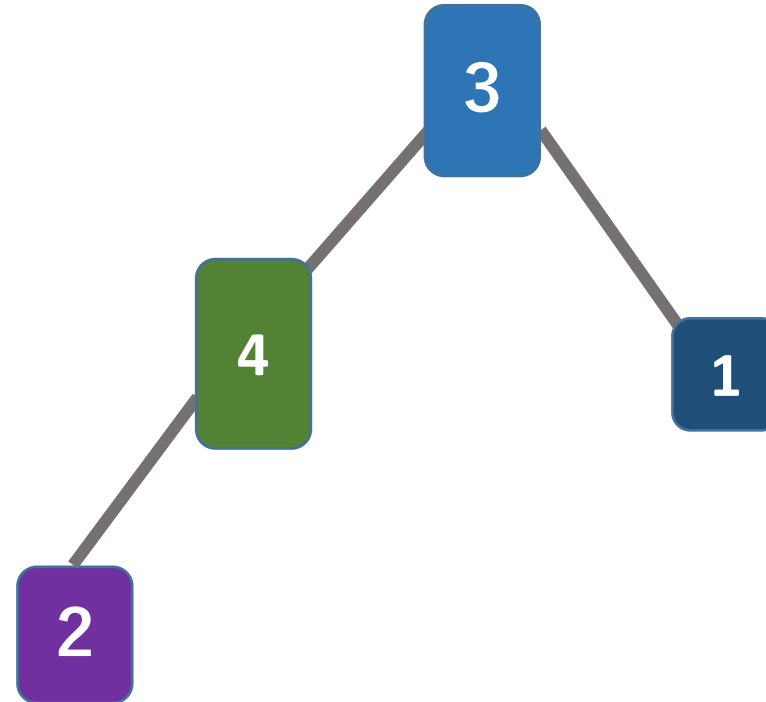
降順ヒープは大きいものから順にデータが取り出される

ヒープを再構築  
**3**を移動(一番後ろ)



降順ヒープは大きいものから順にデータが取り出される

ヒープを再構築  
**3**と**4**を入替え

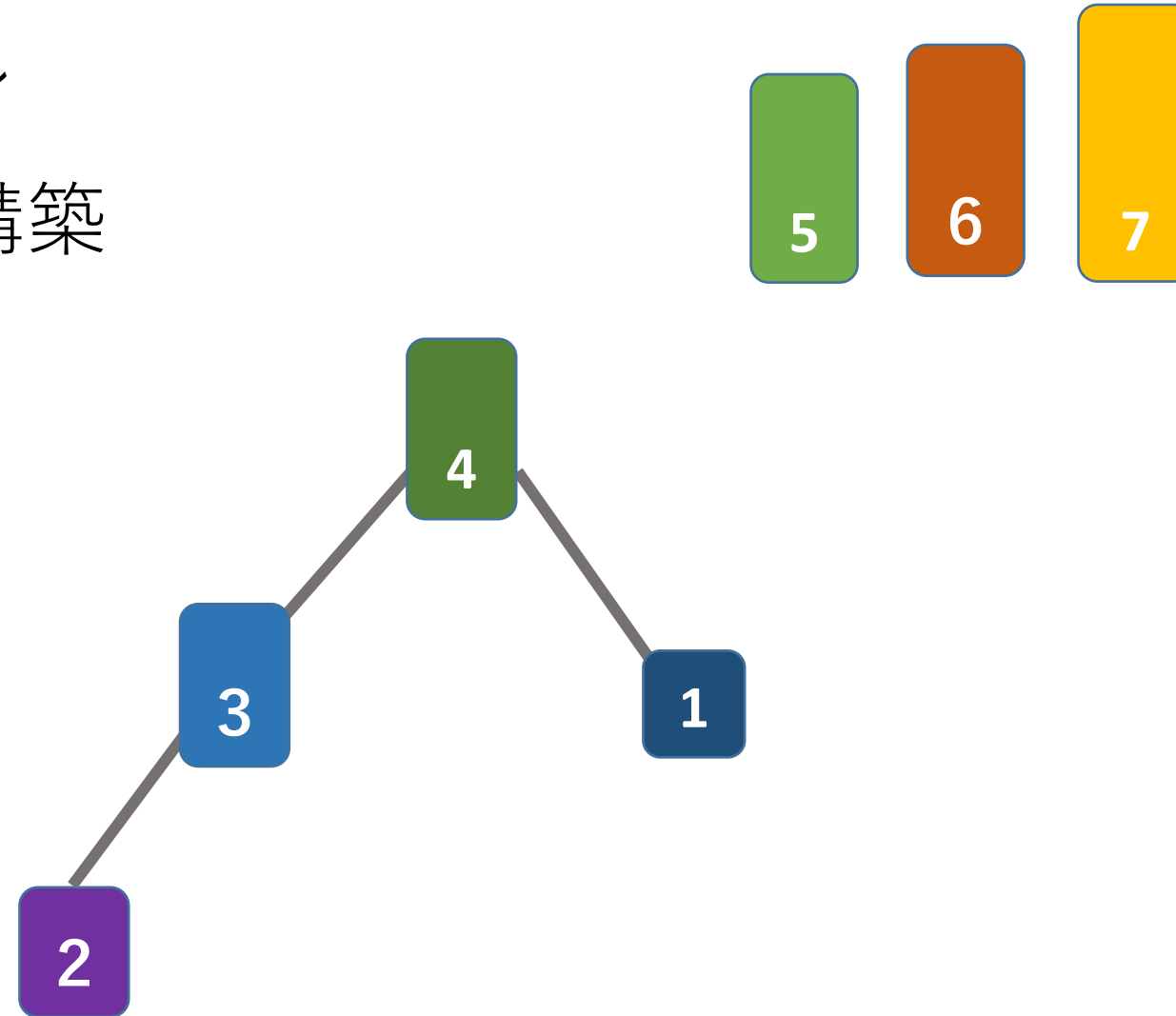


降順ヒープは大きいものから順にデータが取り出される

**4** を取り出し

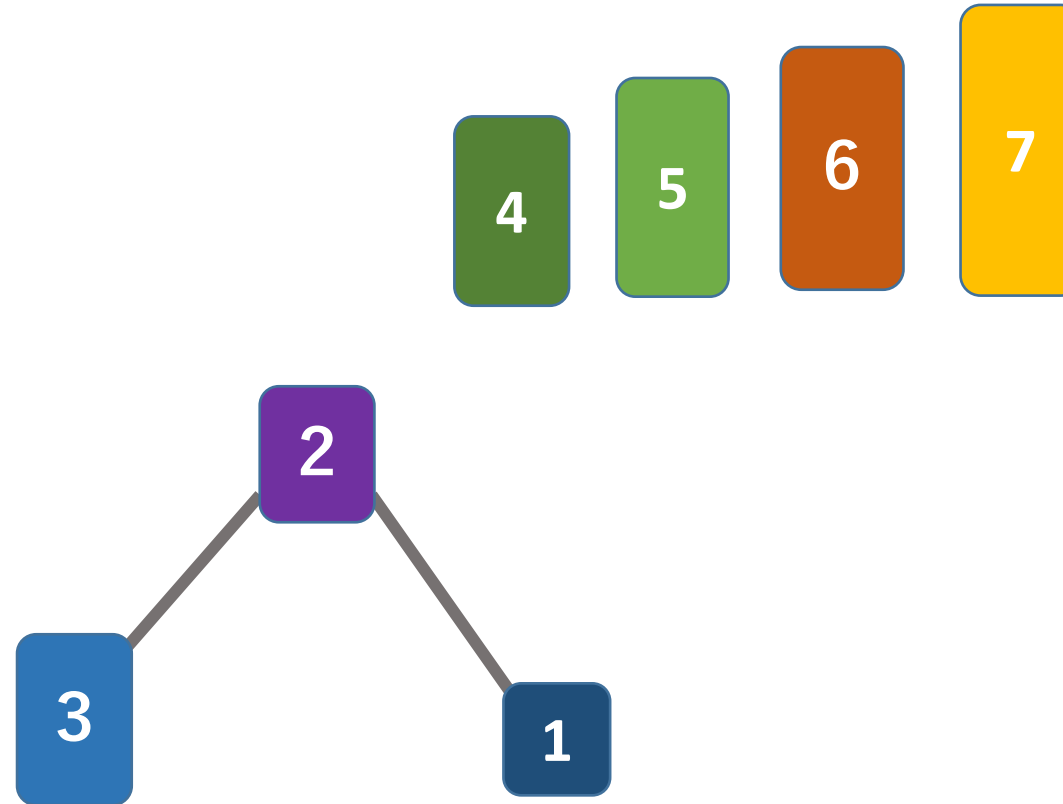
ヒープを再構築

**2** を移動



降順ヒープは大きいものから順にデータが取り出される

ヒープを再構築  
**2**と**3**を入替え

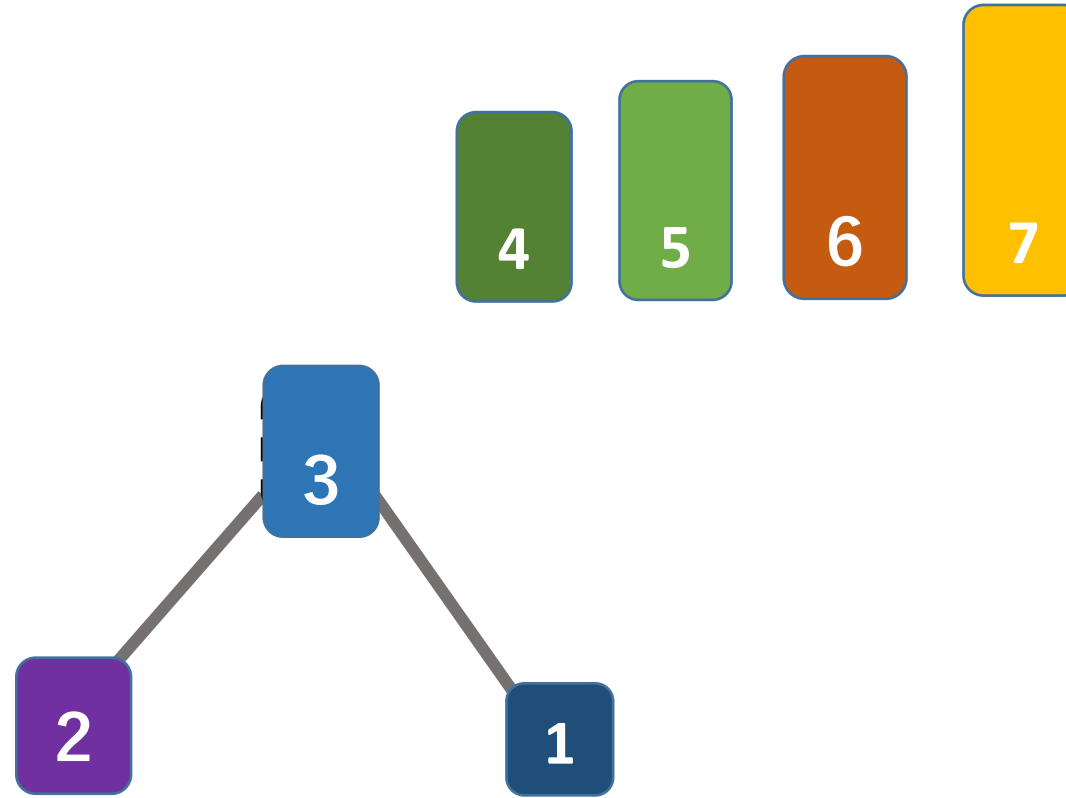


降順ヒープは大きいものから順にデータが取り出される

**3** を取り出し

ヒープを再構築

**2** を移動

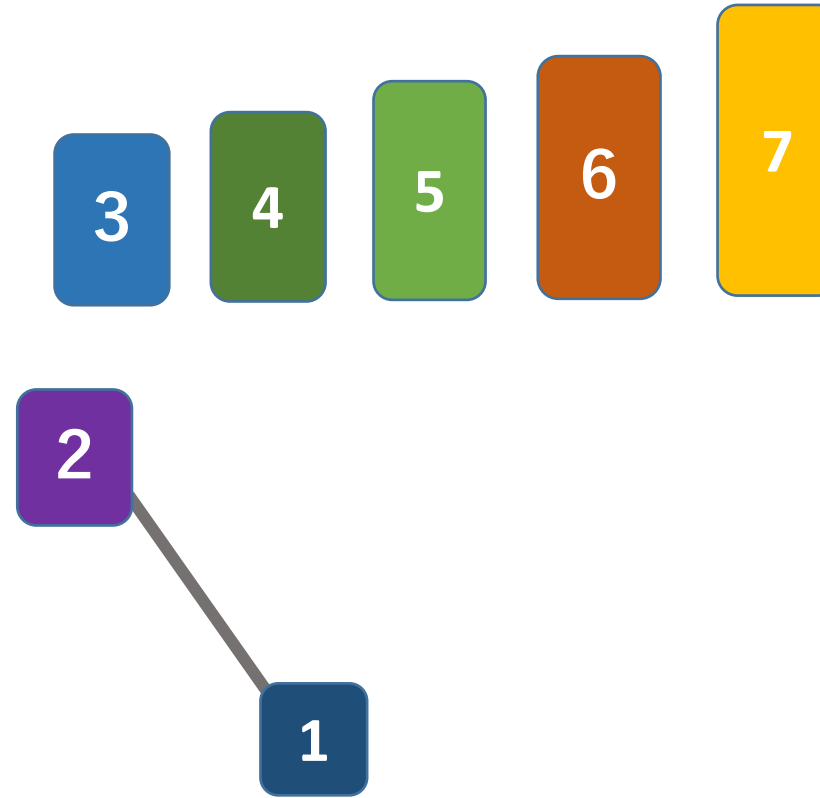


降順ヒープは大きいものから順にデータが取り出される

**2**を取り出し

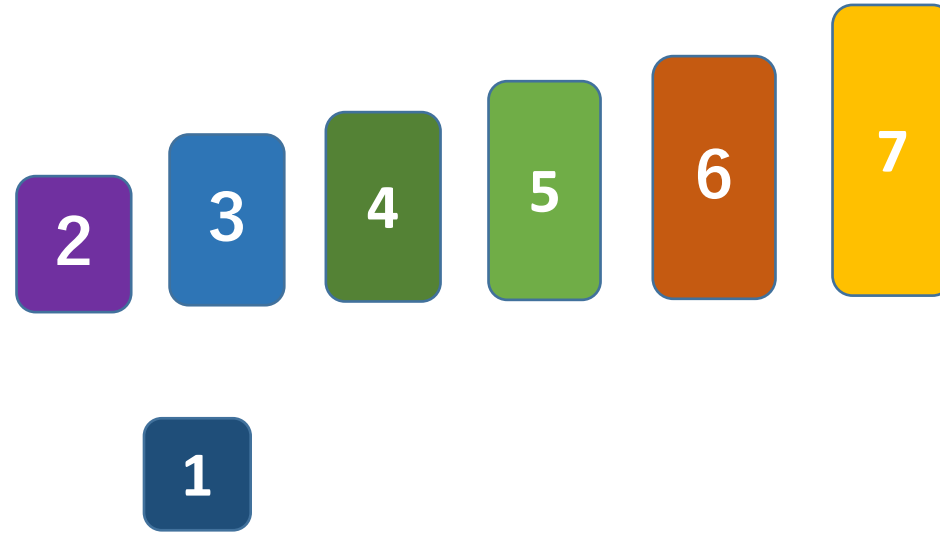
ヒープを再構築

**1**を移動



降順ヒープは大きいものから順にデータが取り出される

**1** を取り出し



**ソート完了！**

# ヒープソート

バブルソート、選択ソート、挿入ソートよりも高速

n個の数字をヒープに格納する時間  $O(n \log n)$

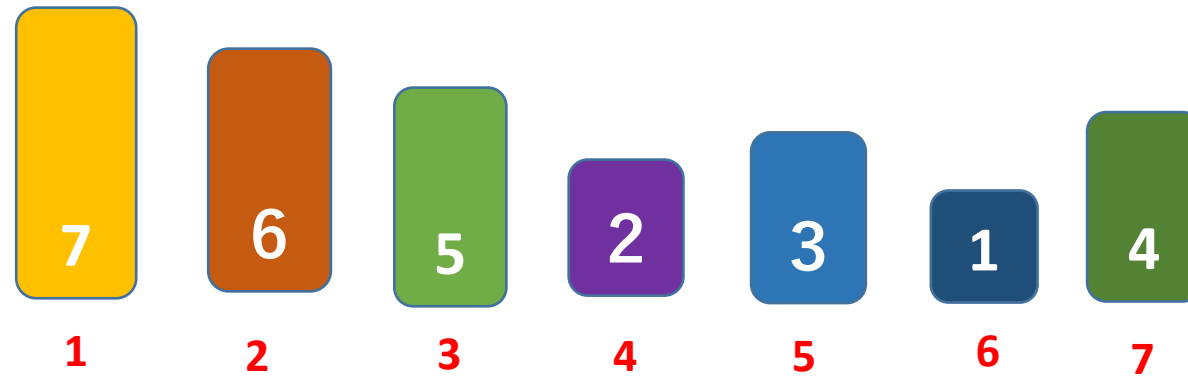
ヒープを再構築する時間  $O(\log n)$

ヒープ再構築の後、ソートする時間  $O(n \log n)$

したがって、ヒープソートの時間は  $O(n \log n)$



# ヒープ

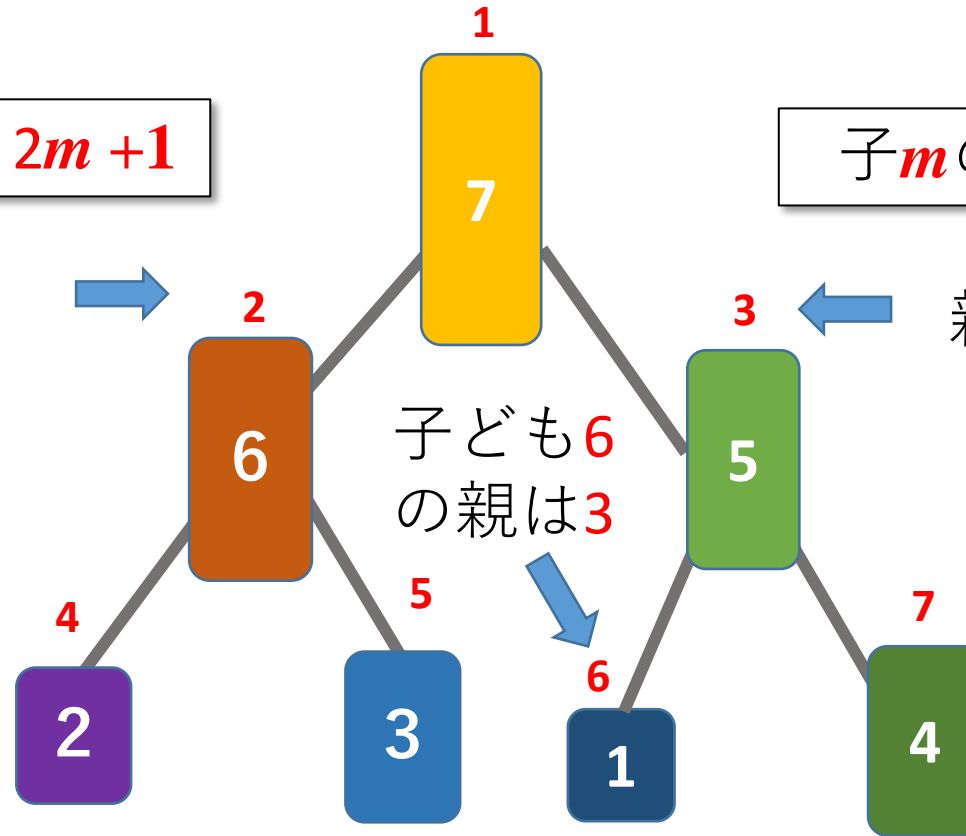


親 $m$ の子どもは、 $2m$ と $2m+1$

子 $m$ の親は、 $m/2$ か $(m-1)/2$

親2の子どもは、4と5

親3の子どもは、6と7



子ども7の親は3

## ヒープ構築 (降順)

```
Sub heap_descending_order()  
Dim n, i, j, a, b As Integer  
Dim s_sumi As Integer
```

```
n = Cells(3, 20)
```

```
If n <= 1 Then Exit Sub  
s_sumi = 1
```

← 3行目k列のセルの値を n とする  
nは1列目にある数列の個数

Do

$i = s\_sumi + 1$

Do

$i$  行目1列の値を $a$ とする

$a = \text{Cells}(i, 1).\text{Value}$

$i$  行目1列の値の親  $j$  行目

If  $i \bmod 2 = 0$  Then

$j = i / 2$

Else

$j = (i - 1) / 2$

End If

$j$  行目1列の値を $b$ とする

$b = \text{Cells}(j, 1).\text{Value}$

親 $i$ の子どもは、 $2i$ と $2i + 1$



子 $j$ の親は、 $j/2$ か $(j-1)/2$

'a>b' ならばセルの値を入れ替える。そうでなければDoループを抜ける

If a > b Then

Cells(i, 1) = b

Cells(j, 1) = a

i = j      ← 親jを子iとする

Else

Exit Do

End If

'時間をs秒あける

Application.Wait [Now() + "0:00:00.5"]

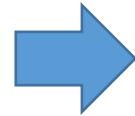
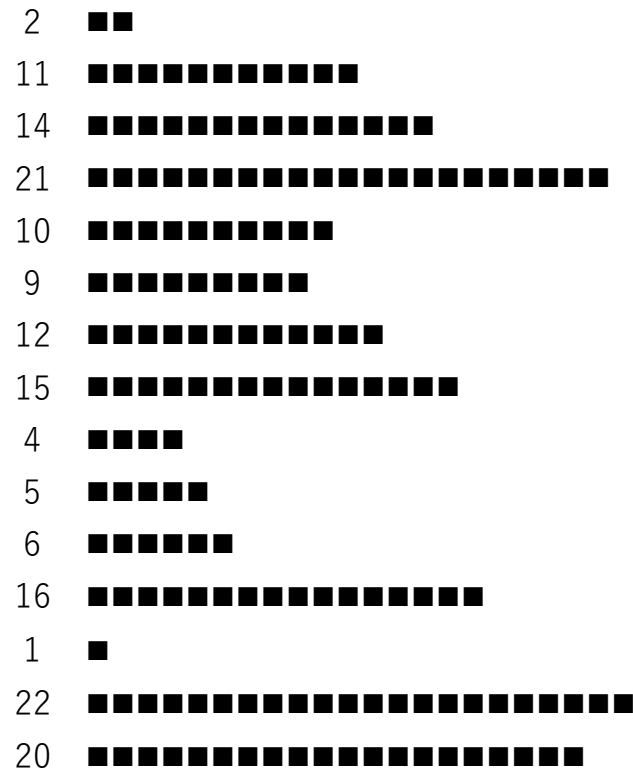
Loop Until i = 1

s\_sumi = s\_sumi + 1

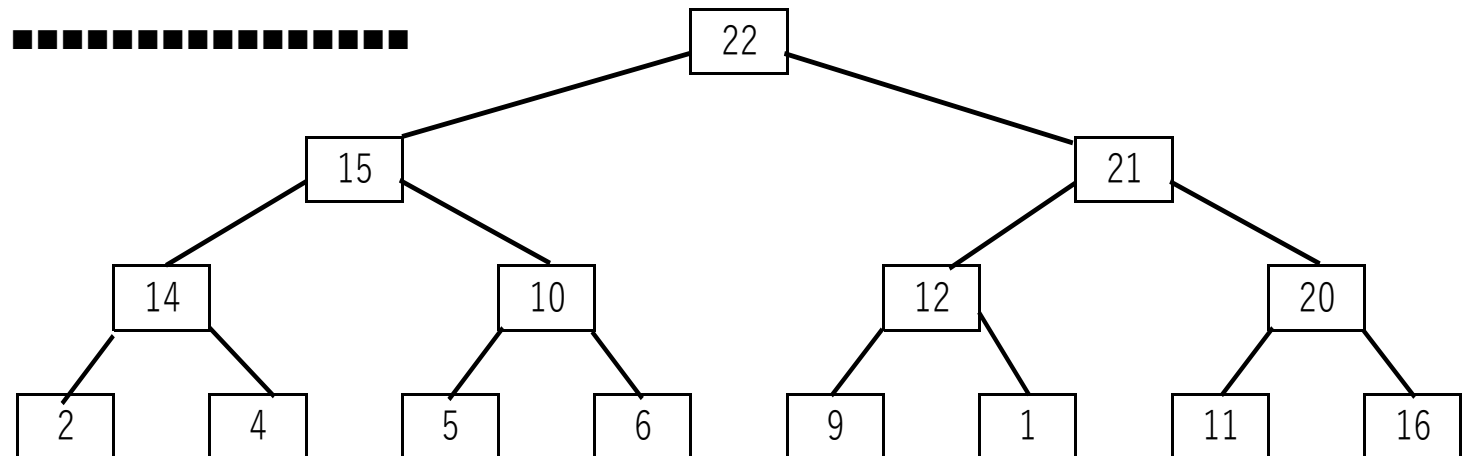
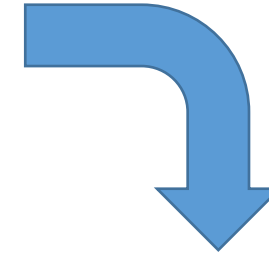
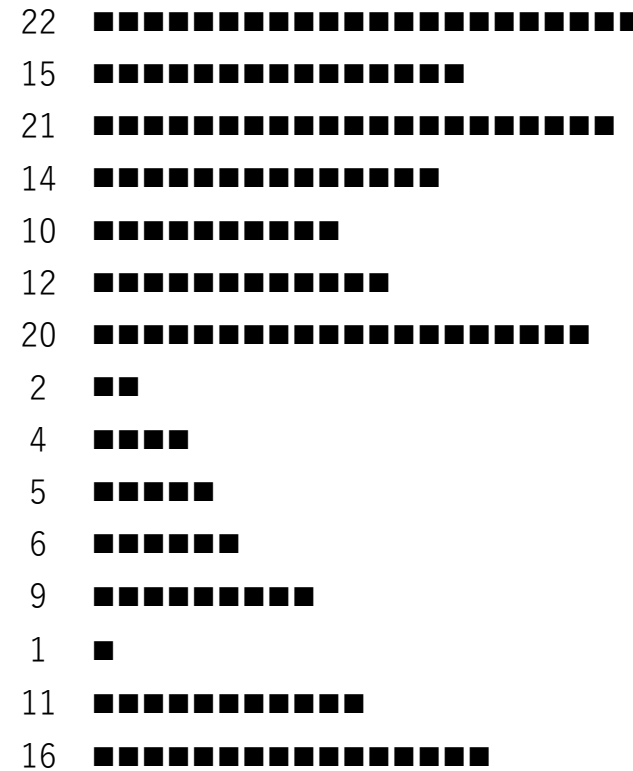
Loop Until s\_sumi = n

End Sub

## 元の数列




## 降順ソート



# ヒープソート (降順)

入力して  
ください



```
Sub Heap_Sort_Descending_Order()
```

```
Dim n, i As Integer
```

```
n = Cells(3, 20)
```

```
‘降順ヒープ
```

```
Call heap_descending_order
```

```
‘データをK列に昇順に並べる (K列=11列)
```

```
i = n
```

```
Do
```

```
Cells(i, 11) = Cells(1, 1)
```

```
Cells(1, 1) = Cells(n, 1)
```

```
Cells(n, 1) = ""
```

```
n = n - 1
```

```
Cells(3, 20) = n
```

```
Call heap_descending_order
```

```
i = i - 1
```

```
Loop Until n = 0
```

```
Cells(3, 20) = WorksheetFunction.Count(Range("K:K"))
```

```
End Sub
```

1行目1列の値ををi行目K列に  
n行目1列の値を1行目1列に  
N行目1列のセルを空白に

## ヒープ構築（昇順） `heap_ascending_order`

ヒープ構築（降順）のプログラム  
（`heap_descending_order`）をコピーして、  
昇順に作り変えてください。

（変更するのは1か所です）

## ヒープソート (昇順) Heap\_Sort\_Ascending\_Order

ヒープソート (降順) のプログラム  
( Heap\_Sort\_Descending\_Order ) をコピーして、  
昇順に作り変えてください。

(変更するのは4か所です)