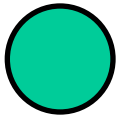
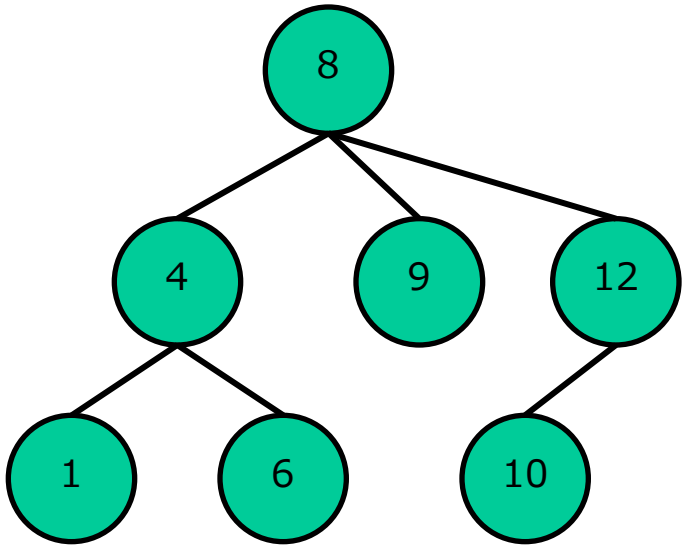
第１０週
ソート（２）
ヒープソートの準備

木構造

- データを木(ツリー)状に配置したデータ構造
  - ノード： 各節点(下図の )
  - キー： 各ノードが持っているデータ
    (下図の8, 4, 9, 12, ...)
  - ルート(根)： 一番上のノード(下図の )

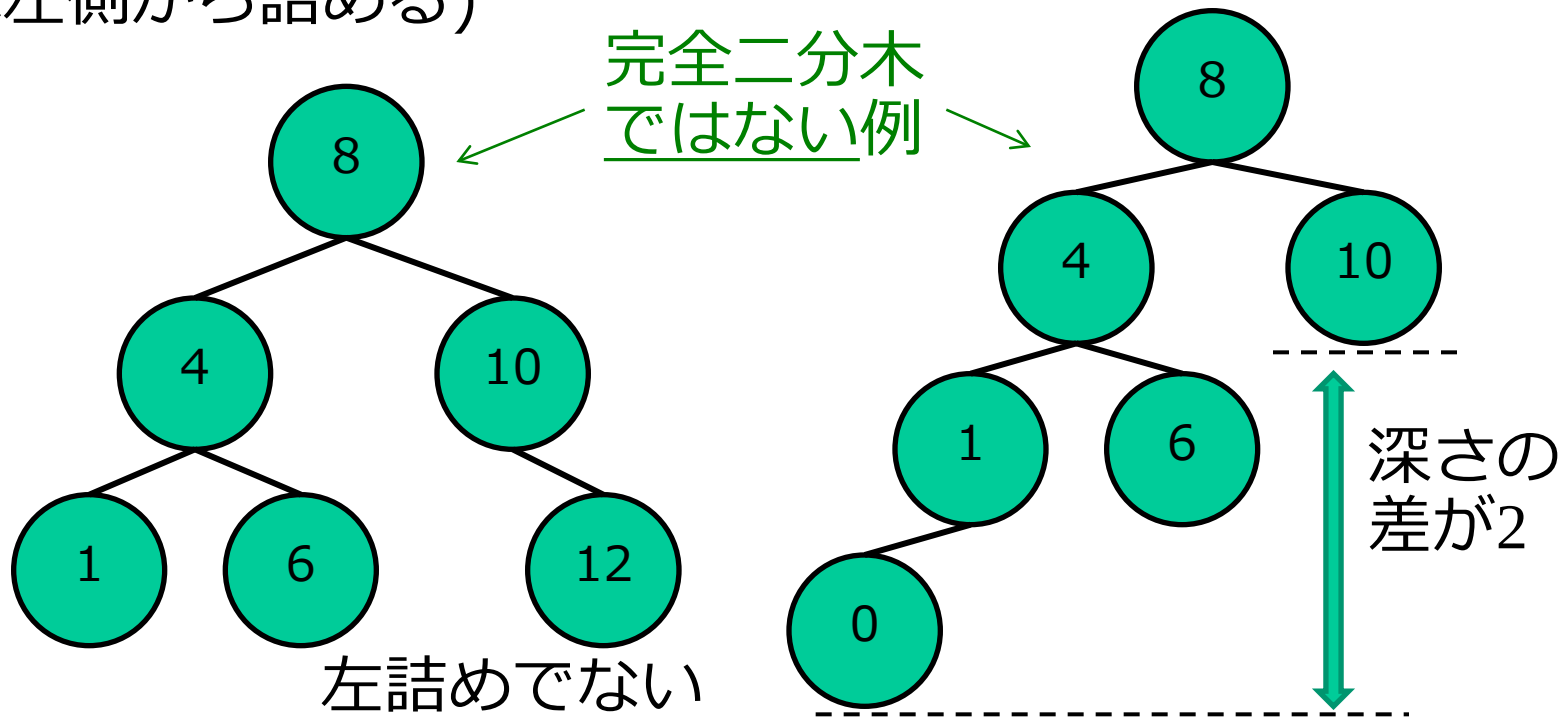
木構造

- 葉： 末端のノード(下図の 1 6 9 10)
- 親： 1つ上のノード(下図では 6 に対して 4)
- 子： 1つ下のノード(下図では 4 に対して 1 6)
- 兄弟： 同じ階層で同じ親を持つノード(1 と 6 等)
- 他にも、先祖、子孫、部分木などの用語がある



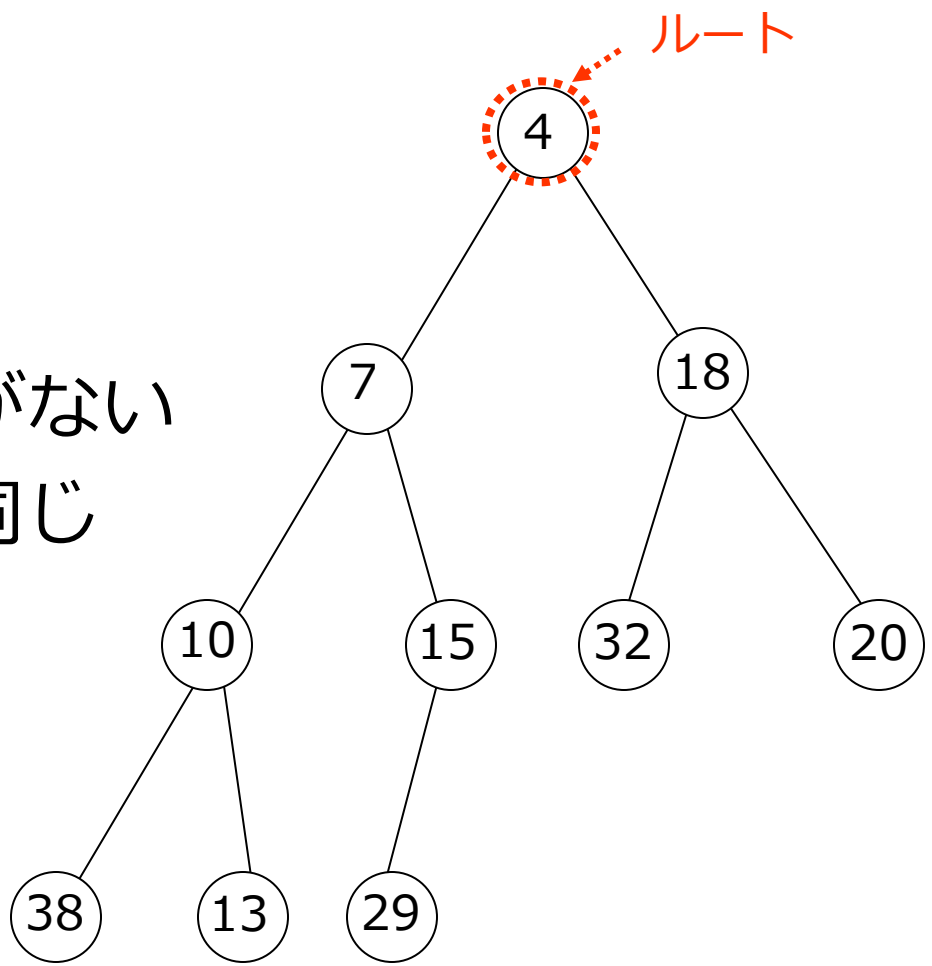
二分木

- 二分木(binary tree)：
  すべてのノードの子が2個以下の木
  - 右の子と左の子を区別
- 完全二分木：
  すべての葉の深さの差が1以下の二分木
  (葉は左側から詰める)



必須課題10-1：ヒープ木

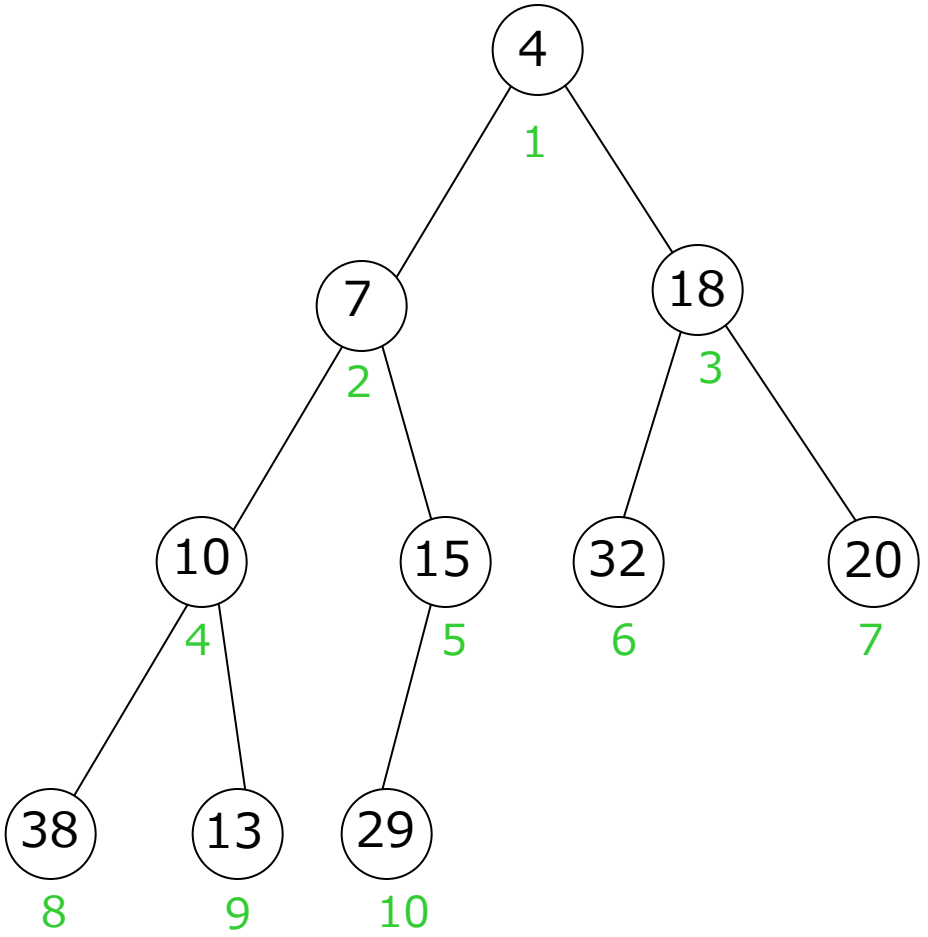
- 最小ヒープ
  - 完全2分木
    - 子供が２
    - 要素が左から空きがない
  - 親は子より小さいか同じ



必須課題10-2：ヒープの1次元配列(1基準)

	4	7	18	10	15	32	20	38	13	29
0	1	2	3	4	5	6	7	8	9	10

1基準の場合は、
０番目の要素は利用しない



- ヒープ構造などの完全2分木は、配列の添字は右図のように親から順に割り当てられる

必須課題10-2：親子の参照方法(1基準の場合)

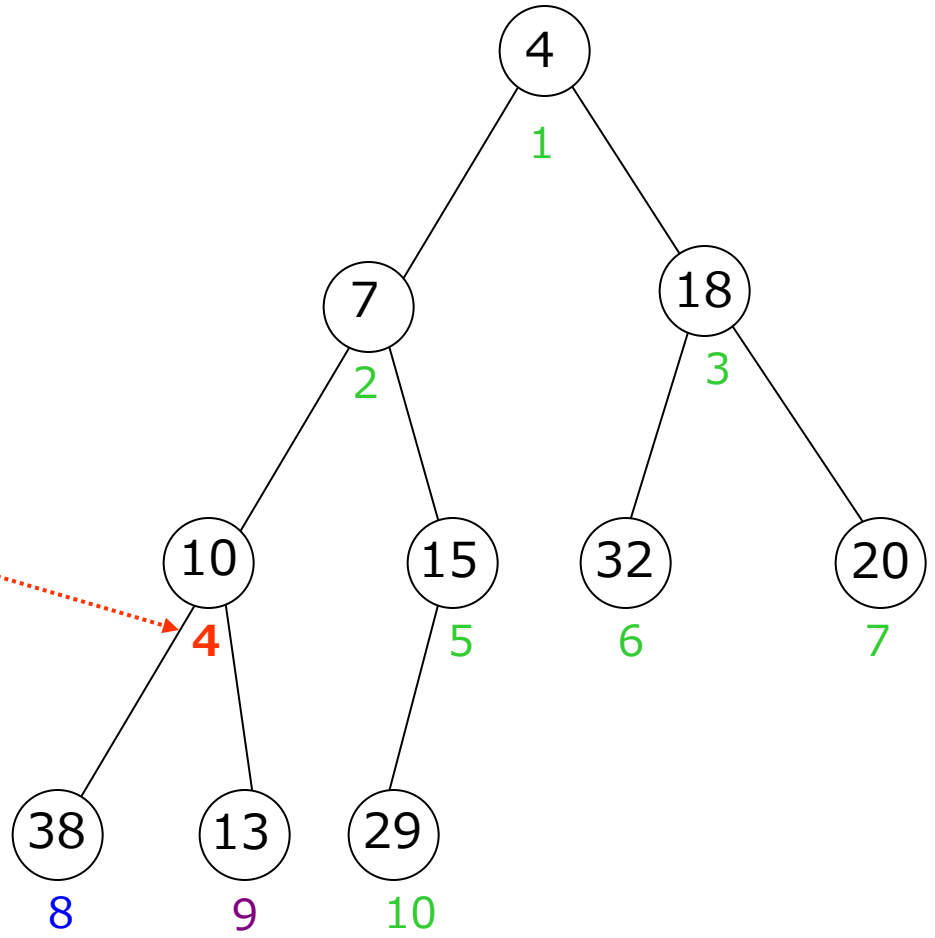
	4	7	18	10	15	32	20	38	13	29
0	1	2	3	4	5	6	7	8	9	10

●親子関係

- 親　　：n/2
- 子（左）：2*n
- 子（右）：2*n+1

●例：nが4の場合

- 親：　　4/2 = 2
- 子（左）：　2 * 4 = 8
- 子（左）：　2 * 4 + 1 = 9



必須課題10-2：check_heap（最小ヒープ）

```
int check_heap(int a[], int N)
```

一次元配列で
表現されたヒープ

ヒープのサイズ

返り値

1：最小ヒープの条件

すべてのノードで下記の条件を満たす

「ノードのキーはその親のキーよりも大きいか等しい」

0：上記の条件を満たさない場合

必須課題10-3：insert

```
void insert(int val, int a[], int *N);
```

挿入される値

一次元配列で
表現されたヒープ

ヒープのサイズ
(ポインタ渡し)

ＳＴＥＰ１：ヒープの最後に新しいノードを挿入する

ＳＴＥＰ２：親と比較する

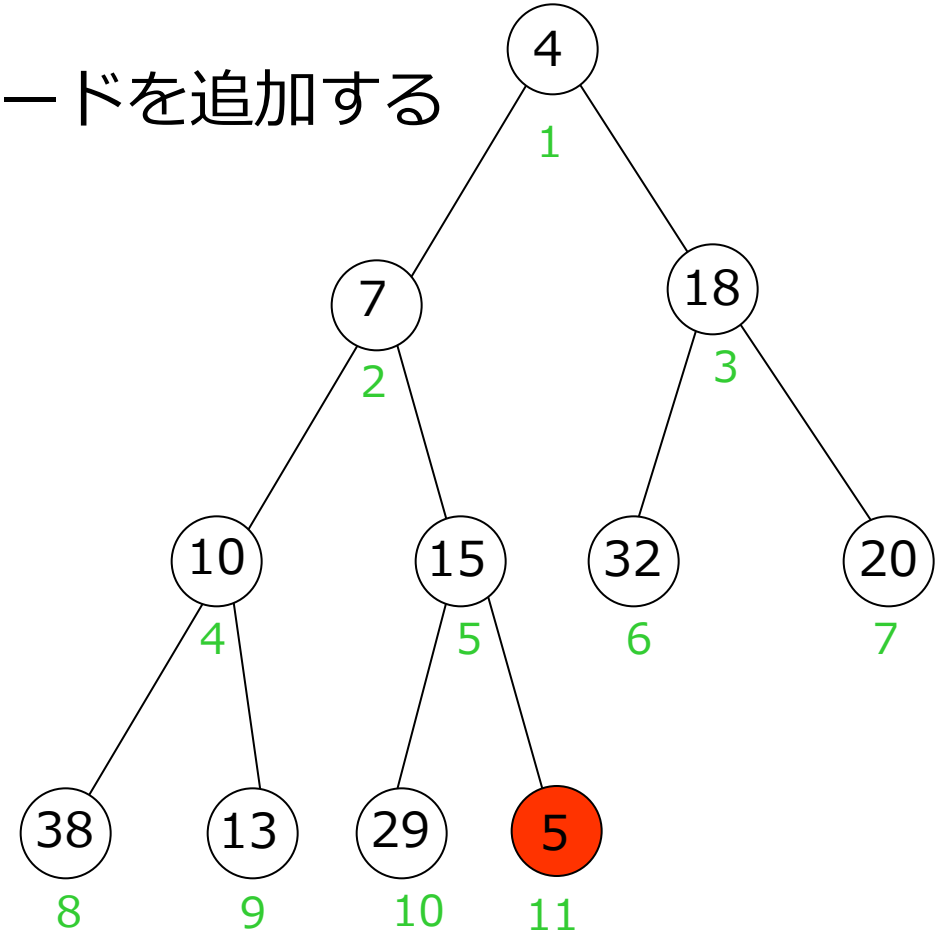
- 親より値が小さかったら場合：
  交換
- 同じか大きい場合：
  処理終了

交換がされ、ルートノードでない場合はSTEP２を繰り返す

必須課題10-3：insertの例： 1 / 4

ＳＴＥＰ１：ヒープの最後に新しいノードを挿入する

この場合、添字１１に値５のノードを追加する

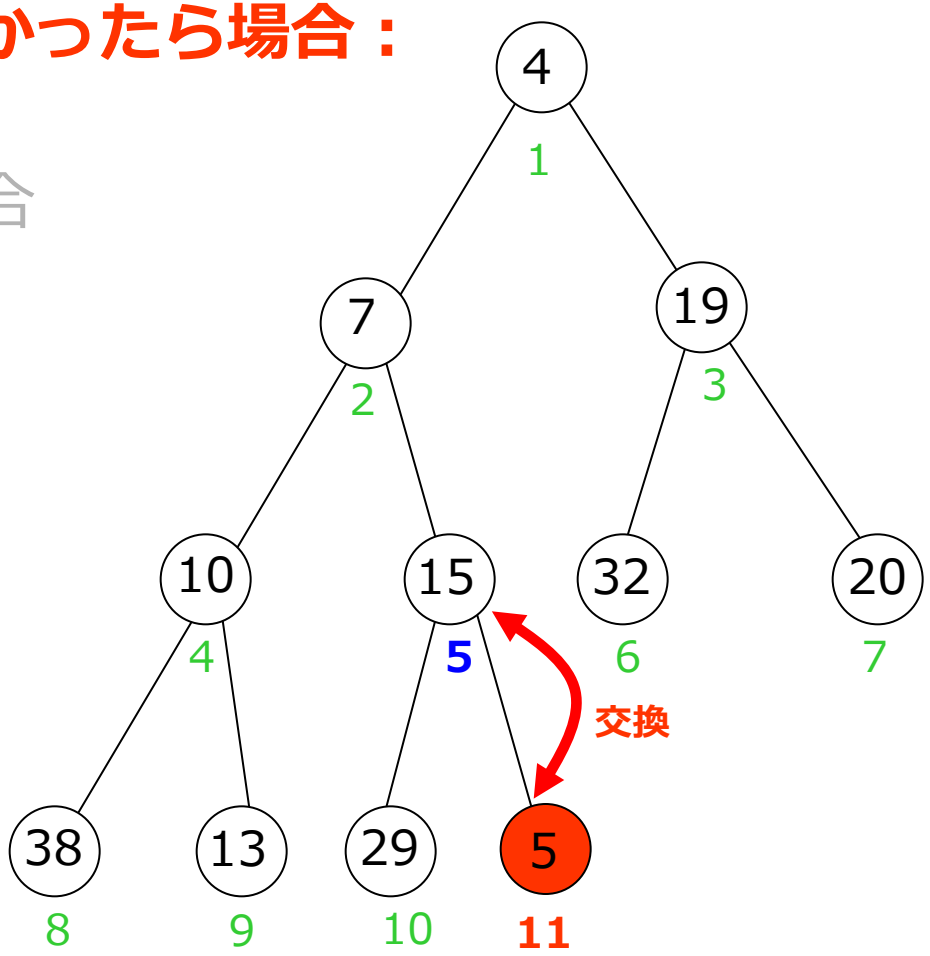


必須課題10-3：insertの例： 2 / 4

ＳＴＥＰ２：親と比較する

- **親より値が小さかったら場合：交換**
- 同じか大きい場合
  処理終了

親　：11/2 ＝ **5**

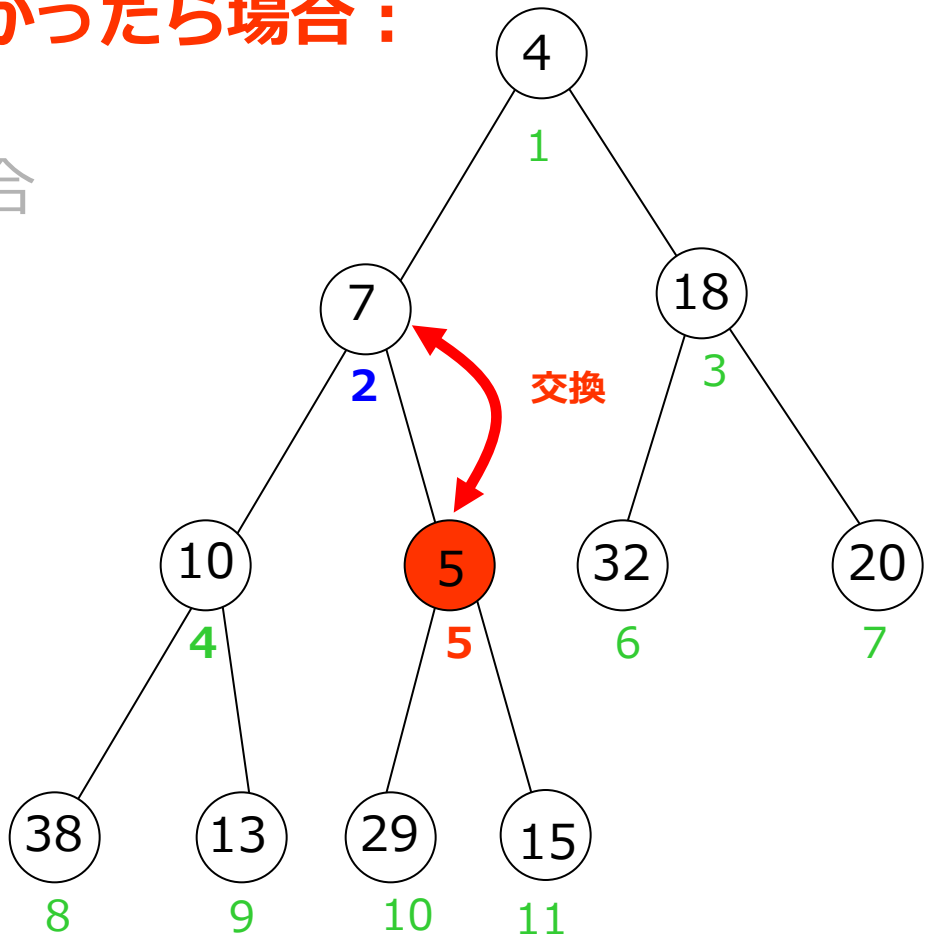


必須課題10-3：insertの例： 3 / 3

ＳＴＥＰ２：親と比較する

- **親より値が小さかったら場合：交換**
- 同じか大きい場合
  処理終了

親　：5/2 ＝ 2

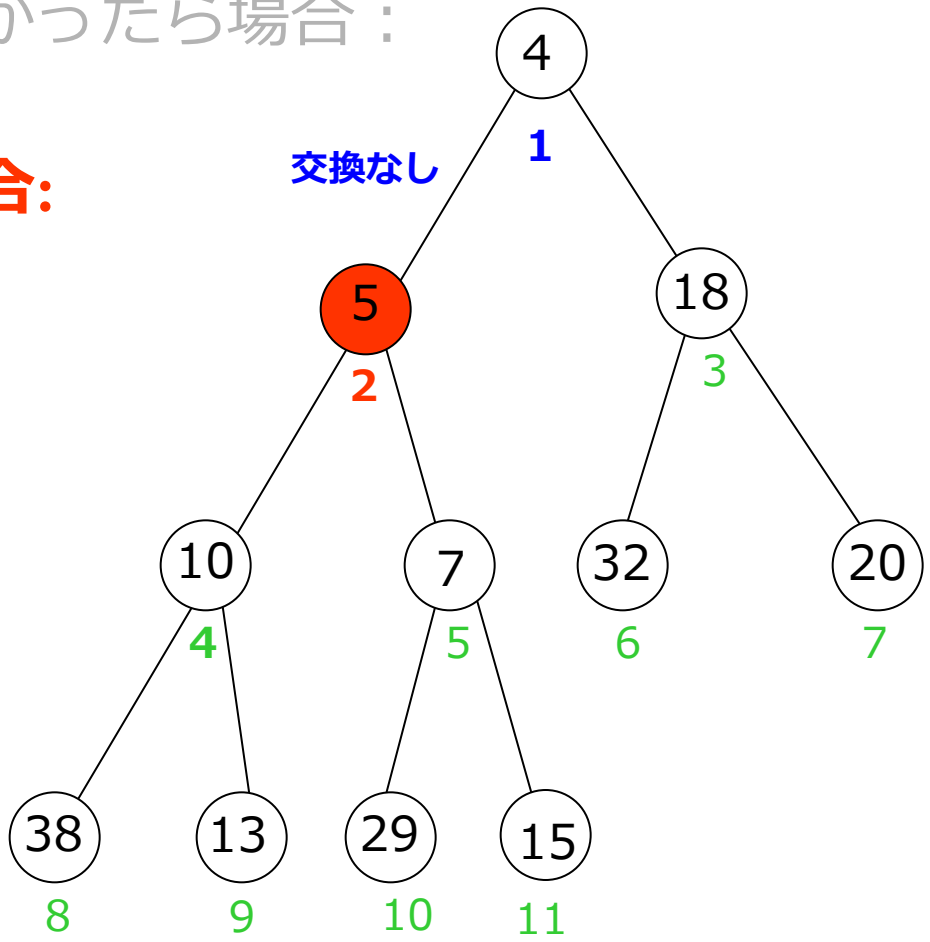


必須課題10-3：insertの例：４/４

ＳＴＥＰ２：親と比較する

- 親より値が小さかったら場合：交換
- **同じか大きい場合:処理終了**

親　：2/2 ＝ 1



必須課題10-3：insert:参考プログラム

```
int main(){
        int heap[HEAP_SIZE+1];
        int heap_size = 0;
        /* 略：変数宣言、ヒープの初期化 */

        insert(      , heap, &heap_size);

        //ヒープ条件が成立しているかどうかチェックして出力
        printf( "HEAP_CHECK = %d¥n", check_heap(heap, heap_size)  );
        //ヒープ領域の内容表示
        printf("HEAP = [ ");
        for(i = 1; i <= heap_size; i++){
                printf("%d ", heap[i]);
        }
        printf("]¥n");

        return 0;
}
```

ヒープの一次元配列：１基準なので＋１

現在のヒープのサイズ

この処理（挿入）を
複数回繰り返す

挿入する値

10-2の関数

必須課題10-4：deletemin

```
int deletemin(int a[], int *N);
```

取り出されてた値

一次元配列で
表現されたヒープ

ヒープのサイズ
(ポインタ渡し)

ＳＴＥＰ１：
ルートノードを取り出し
末尾のノードをルートに持ってくる

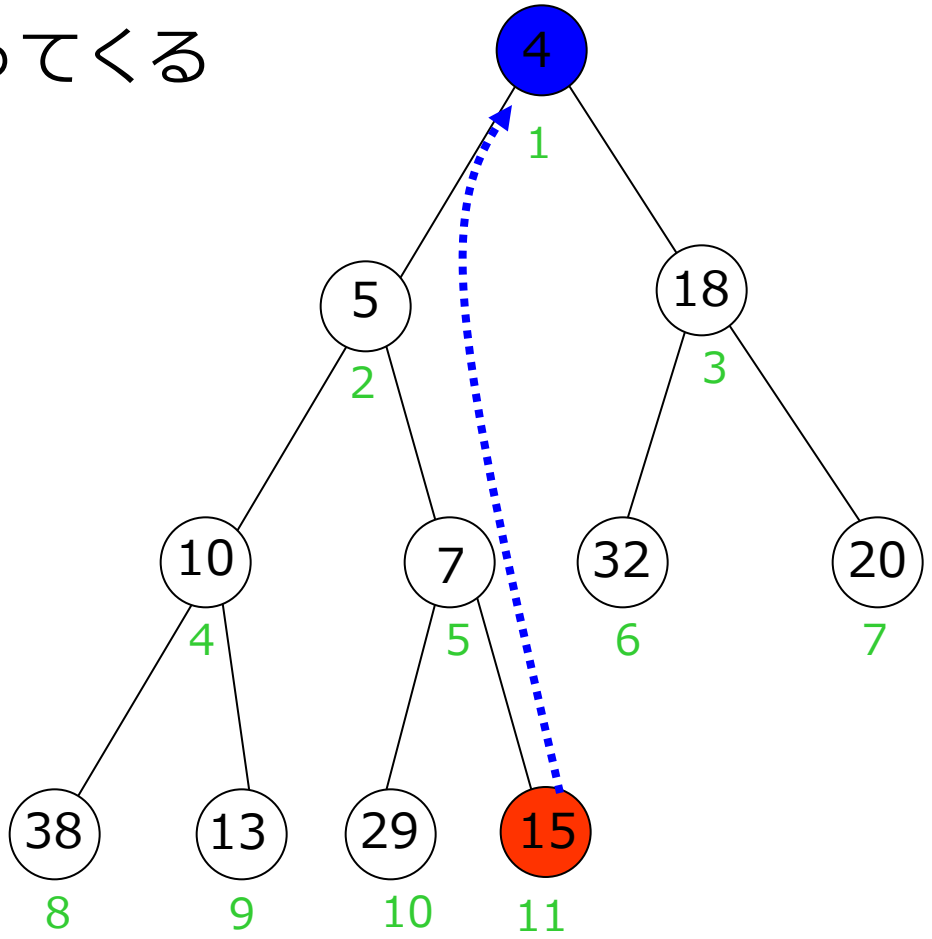
ＳＴＥＰ２：
左右の子供の値の小さい方と比較

- 子供の方が小さい場合：
  交換
- 親の方が大きいか同じ場合：
  処理終了

交換され一番下のノードでない場合はSTEP２を繰り返す

必須課題10-4：deleteminの例： 1 / 4

ＳＴＥＰ１：
ルートノードを取り出し
末尾のノードをルートに持ってくる

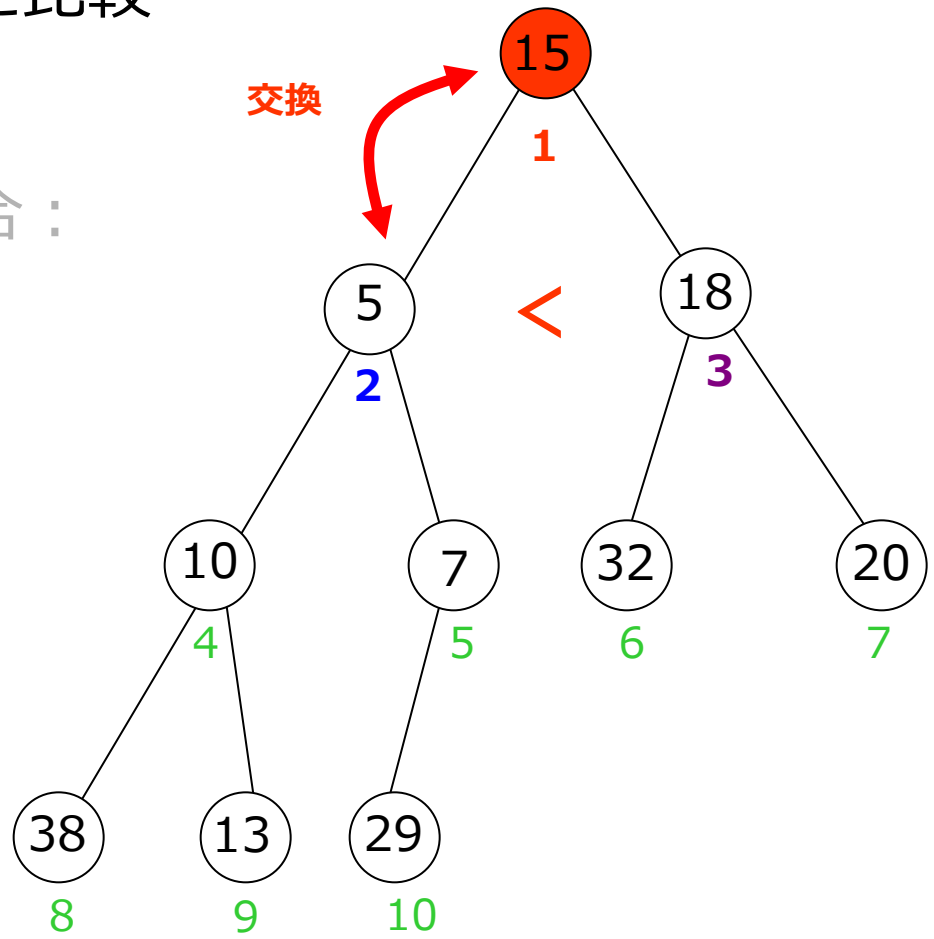


必須課題10-4：deleteminの例： 2 / 4

ＳＴＥＰ２：
左右の子供の値の小さい方と比較

- 子供の方が小さい場合：
  交換
- 親の方が大きいか同じ場合：
  処理終了

子（左）：2*1　＝ 2
子（右）：2*1+1 ＝ 3

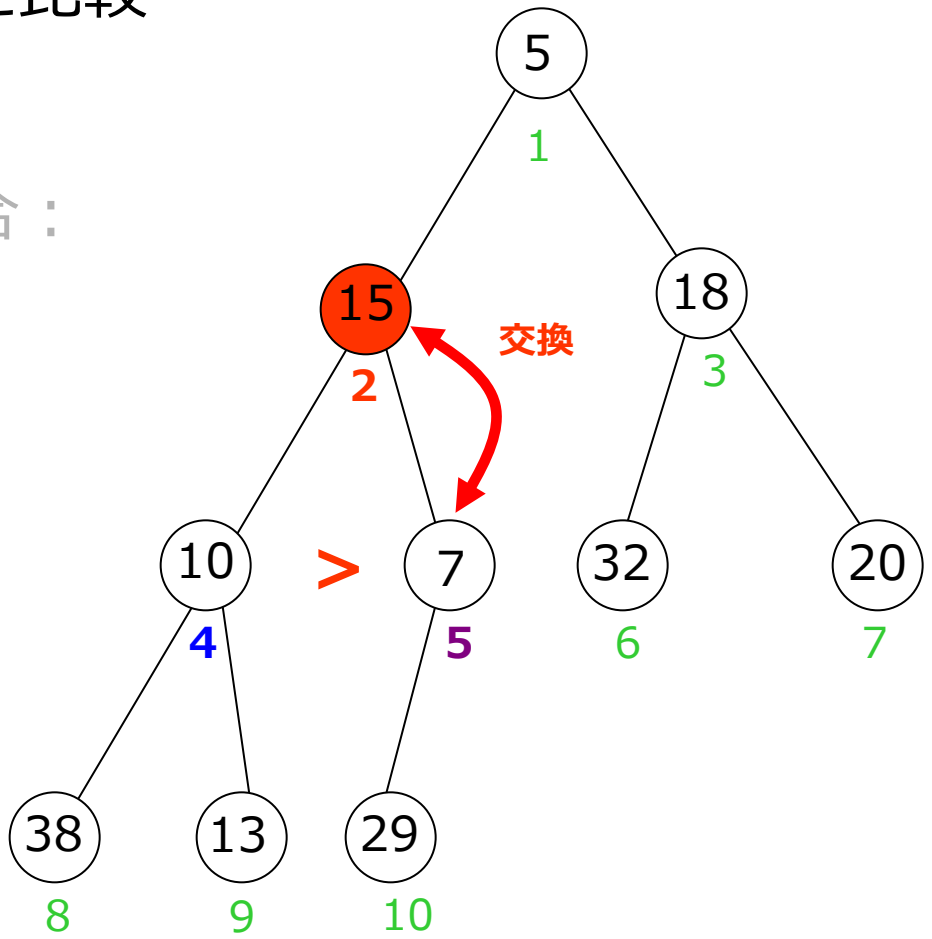


必須課題10-4：deleteminの例： 3 / 4

ＳＴＥＰ２：
左右の子供の値の小さい方と比較

- 子供の方が小さい場合：
  交換
- 親の方が大きいか同じ場合：
  処理終了

子（左）：2*2 ＝ 4
子（右）：2*2+1 ＝ 5



必須課題10-4：deleteminの例： 4 / 4

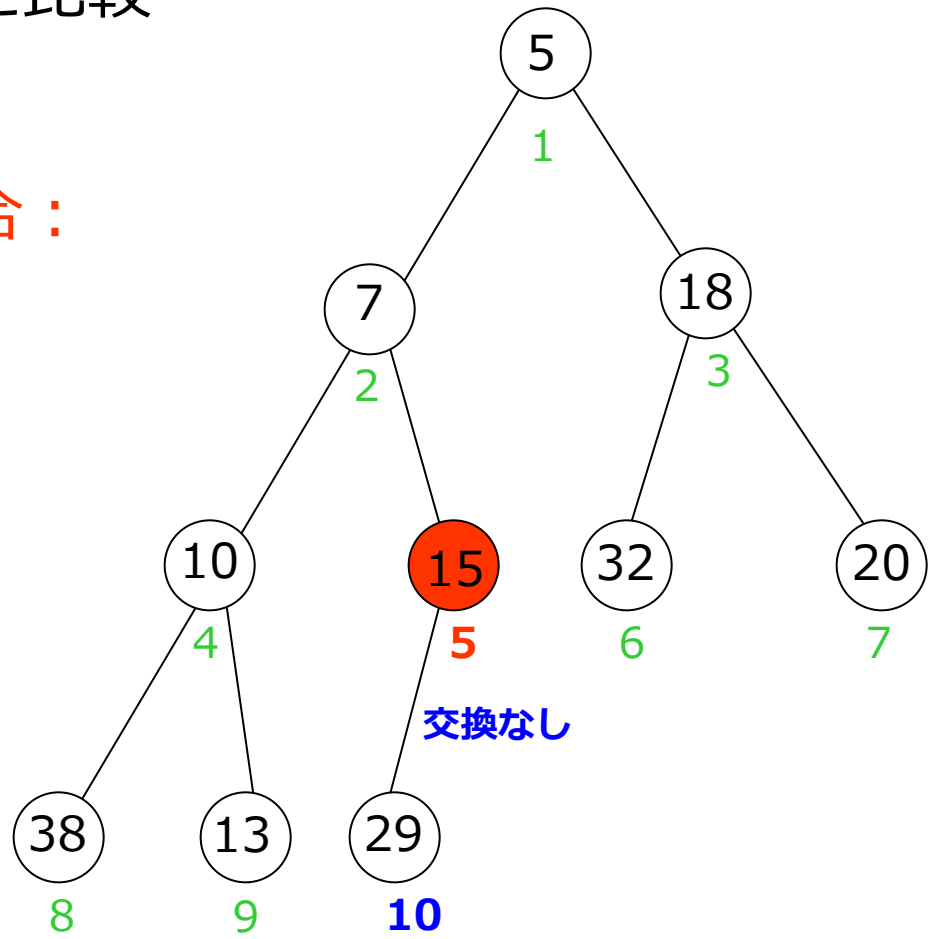
ＳＴＥＰ２：
左右の子供の値の小さい方と比較

- 子供の方が小さい場合：
  交換
- 親の方が大きいか同じ場合：
  処理終了

子（左）：2*5 = 10

子（右）：2*5+1 = 11

右の子供はいなので
省略



必須課題10-4：deletemin確認用プログラム

```
int main(){
        int heap[HEAP_SIZE+1];
        int heap_size = 0;
        /* 略：変数宣言、ヒープの初期化 */
            //ヒープ領域の内容表示
            printf("HEAP = [ ");
            for(i = 1; i <= heap_size; i++){
                        printf("%d ", heap[i]);
            }
            printf("]¥n");

            for(i = 0; i <      i++){
                        printf(“DELETED = %d¥n”, deletemin(heap, &heap_size)   );
                        //ヒープ領域の内容表示
                        printf("HEAP = [ ");
                        for(j = 1; j <= heap_size; j++){
                                    printf("%d ", heap[j]);
                        }
                        printf("]¥n");
            }
        return 0;
}
```

deleteminを実行する回数

その他の注意事項

- 10週目の課題では、
  動的メモリ確保(malloc)が不要
  - 固定長の配列で実現可能

- 必須課題10-2

```
int check_heap(int *a, int N);
```

int a[]でも同じ

著者リスト

1. 安積 卓也（情報システム学科）
2. 大森 隆行（情報システム学科）