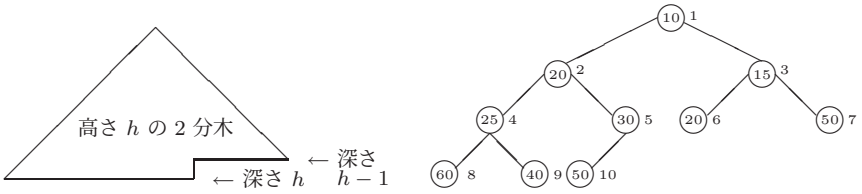
4.5.5 優先順位キュー

最小全域木を求める Kruskal/Prim のアルゴリズムや最短道を求める Dijkstra のアルゴリズムを効率よくプログラム化するためには，アルゴリズム実行の各段階で重みが最小 (最大) の辺を効率的に見つけることが必要である。そのためのデータ構造として有用なものの１つが優先順位キューである．**優先順位キュー***とは，大小関係 (線形順序) をもったデータを保持し，その中の最大値を取り出すことがいつでも効率よく行なえるようなデータ構造のことをいう．その具体的な実現法の１つとして，ヒープ†と呼ばれるデータ構造を用いた方法を以下で述べよう．

● **ヒープ**　ヒープとは，次の図



のように，その形が「どの葉の深さも木の高さ h に等しいかまたは $h-1$ であり，かつ，深さ h の葉はどれも最も左から隙間なく存在している」ようになっている 2 分順序木 (問 4.116 参照) で，各頂点にはデータがラベル付けされていて，次の性質を満たしているもののことである：

> **ヒープ特性**: どの頂点においても，親に付けられたラベルの値は子に付けられたラベルの値以下である．

ヒープは，その形ゆえに，どんなプログラミング言語も備えているデータ表現法である 1 次元配列 $\langle H[1], H[2], \ldots, H[n]\rangle$ (n はノード数) を用いて容易に表わすことができる．$H[1]$ を根とし，ノード $H[i]$ の左の子を $H[2i]$ で，右の子を $H[2i+1]$ で表わせばよい．$2i > n$ は $H[i]$ に左の子がないことを，$2i+1 > n$ は $H[i]$ に右の子がないことを意味する．$H[i]$ の親は $H[\lfloor i/2 \rfloor]$ である．上右図で，各ノードの右に小文字で示した数字は配列の要素 $H[i]$ の添字 i である．

---

*プライオリティキュー (「優先順位が付いたデータの並び」の意): priority queue.

†ヒープ：heap.「積み上げたもの」の意.

● **最小値**　ヒープの根のラベルは，すべてのラベルの中の最小値であることに注意する．したがって，全データの中の最小値を $O(1)$ 時間で取り出すことができる．最小値を取り出した結果空いた根には，ヒープ末尾 (最も右端の葉) のラベルを移す (ヒープのノード数も 1 減らす)．それによってヒープ特性が満たされなくなったら，以下のように修正を施し，ヒープに戻しておく：

procedure reheapify(H, n); / $*$ n はヒープのノード数 $*$/
begin
1. $i \leftarrow 1$ とする．
2. $H[i]$ に子があり，かつ，$H[i]$ のラベルが 2 人の子のどちらかのラベルより大きい限り，以下のことを根から葉に向かって進める．
   2.1. 小さい方のラベルを持つ子を $H[j]$ とする．
   2.2. $H[j]$ のラベルと $H[i]$ のラベルを入れ替える．
   2.3. $i \leftarrow j$ として 2.1 へ戻る．

end

● **ヒープへの挿入**　一方，与えられたデータ $x_1, \ldots, x_n$ に対するヒープ H は以下のように作る．空のヒープから始め，heap_insert$(H, i-1, x_i)$ を $i = 1 \sim n$ について実行する．heap_insert(H, n, x) では，ノード数 n の既存ヒープ H の末尾 (右端の葉のところ) にデータ x を追加挿入し，その結果崩れるヒープ特性を，葉から根に向かって修正していく：

procedure heap_insert(H, n, x);
begin
1. $i \leftarrow n+1$; $H[i] \leftarrow x$; /* データ x を挿入 */
2. $H[i]$ に親 ($H[j]$ とする) があり，そのラベルが $H[i]$ のラベルより大きい限り，以下の 2.1 を繰り返す (葉から根へ向かって進行する)．
   2.1 $H[j]$ と $H[i]$ を入れ替え，$i \leftarrow j$ とする．

end

reheapify は木の高さ $O(\lfloor \log_2 n \rfloor)$ に比例する時間で実行でき，heapfy は $O(n)$ 時間で実行できる効率的な方法である．ヒープを使って，ソーティングを効率よく行なうこともできる (問 4.117 参照)．