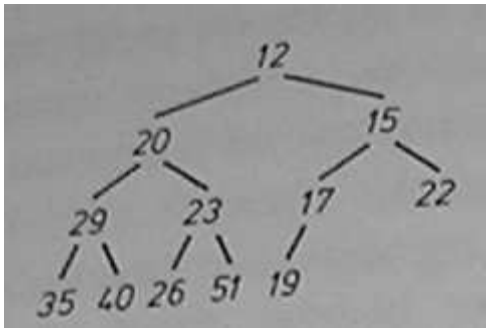
Sortowanie przez kopcowanie (sortowanie stogowe)

Kopiec (stóg, sterta), ang. heap

Struktura danych – reprezentacja zbioru elementów (np. liczb), mająca postać drzewa binarnego.
Zastosowania:
– sortowanie przez kopcowanie porządkuje n-elementową tablicę w czasie $\Theta(n \log n)$,
– kolejki priorytetowe: określanie operacji w zbiorze, służących do dodawania nowego elementu oraz usuwania elementu najmniejszego w czasie $O(\log n)$.

Przykład kopca:



Własności kopca:

1. Uporządkowanie.
Wartość każdego wierzchołka (ojca) jest nie większa niż wartości jego synów.

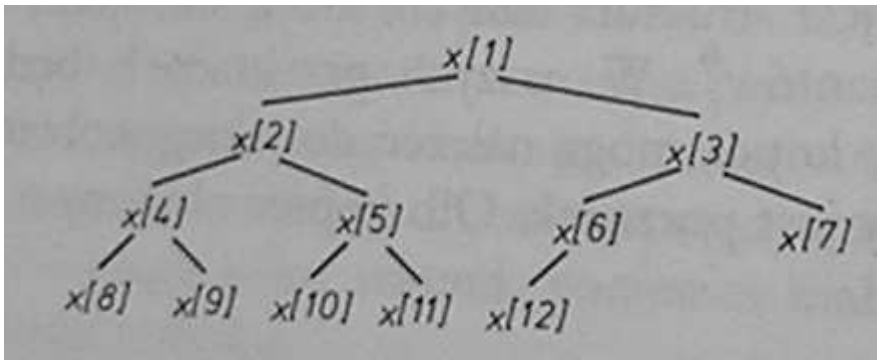
Wniosek: najmniejszy element zbioru znajduje się w korzeniu drzewa. Nic jednak nie wiemy o wzajemnym uporządkowaniu lewego i prawego syna.

2. Kształt
Liście (synowie) znajdują się na jednym lub więcej poziomach, a te na najniższym poziomie są przesunięte jak najbardziej w lewo.

Wniosek: jeżeli drzewo zawiera n wierzchołków, to żaden z nich nie jest bardziej oddalony od korzenia niż o $(\log n)$ węzłów.

Własności 1 i 2 są warunkami na tyle silnymi, żeby umożliwić szybkie odnalezienie elementu najmniejszego w zbiorze a jednocześnie umożliwiają szybką reorganizację struktury kopca po dodaniu lub usunięciu z niego elementu.

Realizacja kopca za pomocą tablicy:




```
korzeń       = 1
wartość(i)   = x[i]
lewysyn(i)   = 2*i
prawysyn(i)  = 2*i+1
ojciec(i)    = i div 2
```

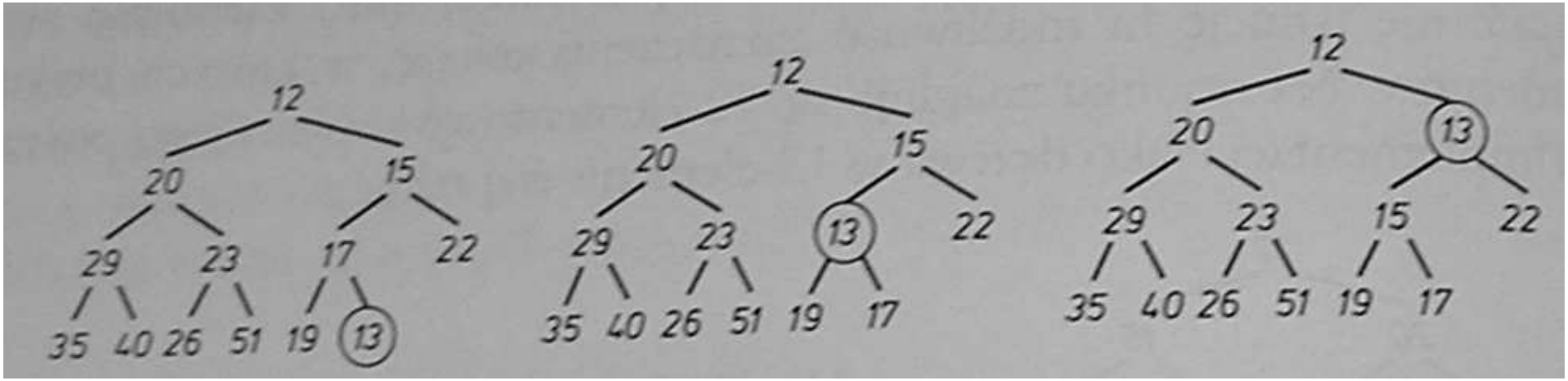
Tablica x={12,20,15,29,23,17,22,35,40,26,51,19}

Uwaga: w C/C++ tablice indeksujemy od zera a nie od jedynki!

Ściśle:
Tablica $x[1...n]$ jest kopcem, jeżeli $\forall_{2 \le i \le n}\ x[i \text{ div } 2] \le x[i]$.
Mówimy, że zachodzi $\texttt{kopiec}(1,n)$.
Fragment tablicy $x[d...g]$ jest kopcem (czyli zachodzi $\texttt{kopiec}(d,g)$), jeżeli $\forall_{2d \le i \le g}\ x[i \text{ div } 2] \le x[i]$.

Procedury porządkowania kopca

1. Załóżmy, że $x[1...n-1]$ jest kopcem i dodajmy nowy element $x[n]$. Prawdopodobnie $x[1...n]$ nie jest już kopcem.
Procedura przywracania własności kopca dla tablicy $x[1...n]$:



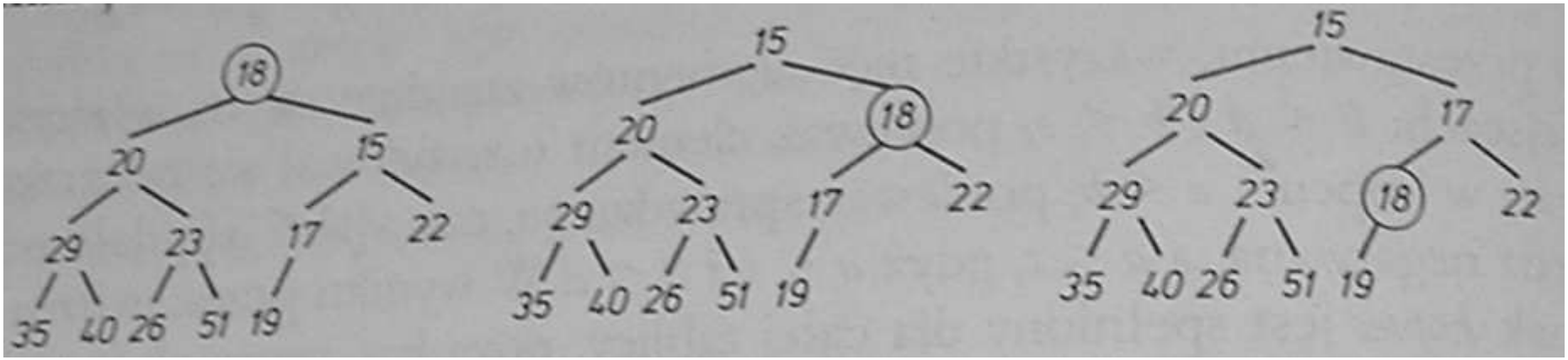
Procedura `doGóry(`*n*`)`:

– Przemieszczamy nowy element w górę drzewa tak daleko, jak powinien dotrzeć, zamieniając go po drodze z ojcem. Kończymy, gdy przemieszczany element stanie się większy lub równy ojcu.

– Uwaga: droga w górę drzewa to malejące indeksy w tablicy.

– Koszt operacji: O(log *n*).

2. Jeżeli w kopcu $x[1...n]$ na pozycji $x[1]$ przypiszemy nową wartość, to warunek $kopiec(2,n)$ pozostanie prawdziwy. Procedura przywracania własności $kopiec(1,n)$:



Procedura $naDół(n)$:

– Przemieszczamy element $x[1]$ w dół drzewa (indeksy rosną!), zamieniając go po drodze z mniejszym synem, aż do chwili kiedy nie ma on już żadnych synów albo jest od nich mniejszy lub równy.

– koszt operacji: O(log n).

Kolejki priorytetowe

Kolejka umożliwia operację dodania i usunięcia elementu z pewnej ich sekwencji, w naszym przypadku struktury kopca.

Początkowo kolejkę stanowi pusty zbiór S.

Procedura `Wstaw(`t`)` wstawia do kolejki nowy element t:

```
Wstaw(t):
   S = S ∪ {t}
   n++
   doGóry(n)
```

Procedura `usunMin()` usuwa najmniejszy element zbioru:

```
usunMin():
   S = S \ {t} i t = min(S)
   S[1] = S[n]
   n--
   naDół(n)
```

Ostateczna postać algorytmu sortowania przez kopcowanie:

```
pętla od i=1 do n
   wstaw(x[i])  // Utworzenie kopca
pętla od i=1 do n
   usunMin()    // Zdejmowanie z kopca el. min.
```

Pesymistyczny i średni koszt operacji: $\Theta(n \log n)$.