ALGORYTMY I STRUKTURY DANYCH – KOPIEC

- **Kopiec –** pełne drzewo binarne (z wyj. być może ostatniego poziomu)
  - **własność kopca**
    - dla każdego węzła x (x ≠ root) zachodzi: key[x] ≤ key[parent[x]]

Operacje na kopcu

- reprezentacja tablicowa

- przywracanie własności kopca
  - **dane**: indeks i
  - **wynik**: kopiec H

zatapianie(H,i)

```
l ← left[H[i]]
r← right[H[i]]
if(l≤ rozmiar[H] and H[l] > H[i])
    większy←l
else
   większy←i
if(r≤ rozmiar[H] and H[r] > H[większy])
    większy←r
if(większy ≠ i)
    zamień H[i] z H[większy]
    zatapianie(H, większy)
```

*i jest mniejszy od (jednego ze) swoich synów

wynurzanie(H,i)

```
if(i > 1)
  ojciec ← i/2
  if(H[i] > H[ojciec])
     zamień H[i] z H[ojciec]
     wynurzanie(H, ojciec)
```

*i jest mniejszy od (jednego ze) swoich synów

- **złożoność**: O(log(n))

- tworzenie kopca
  - **dane**: nieuporządkowana tablica H
  - **wynik**: kopiec H

```
for(i = n/2 downto 1)
   zatapiane(H, i)
```

- **złożoność**: O(n)

- sortowanie kopcowe
  - dane i wynik: jak dla pozostałych algorytmów sortowania

```
utwórz kopiec H
for (i ← n downto 2)
   zamień H[1] z H[i]
   n← n -1
   zatapianie(H,1)
```

- **złożoność**: O(n log(n))

✖ Operacje na kolejce priorytetowej

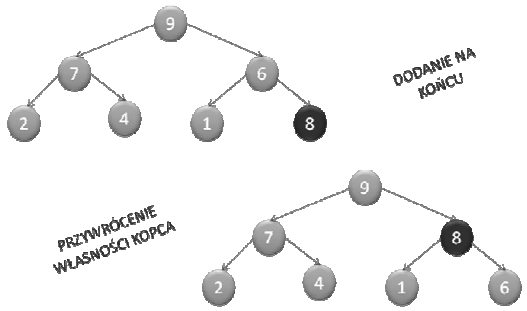
- ★ reprezentacja w postaci kopca binarnego

+ umieszczenie elementu w kolejce
  - × **dane**: kolejka priorytetowa (kopiec H) o długości n i element o kluczu x
  - × **wynik**: kolejka z dodanym elementem

insert(x)

H[n+1]←x
wynurzanie(H, n+1)
i←n+1
while(i >1 and H[ojciec[i]] < x)
  H[i]←H[ojciec[i]], i=ojciec[i]
H[i]←x

★ brak kontroli przepełnienia

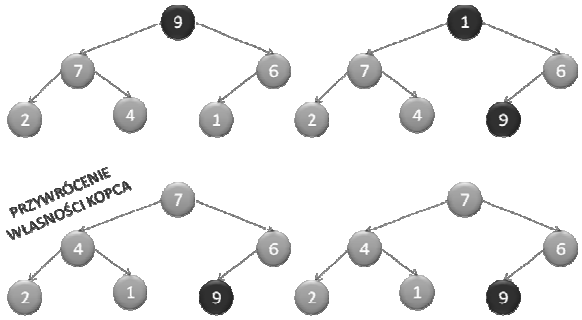


+ **złożoność**: O(log(n))

+ pobranie z kolejki elementu ***największego***
  - × **dane**: kolejka priorytetowa (kopiec H) o długości n
  - × **wynik**: element x (największy)

extract-max()

if(n<1)
  „kolejka pusta”
else
  zamień H[1] z H[n]
  rozmiar[H]←n-1
  zatapianie(H.1)
  ret(H[n])



+ **złożoność**: O(log(n))