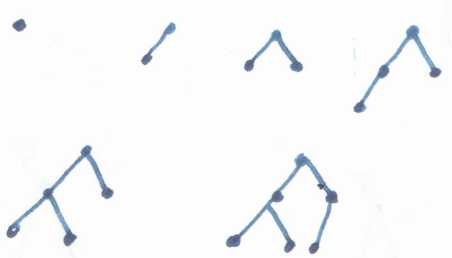
Kopiec

Kopiec – drzewo binarne, w którego wierzchołkach są liczby

Kolejka priorytetowa – struktura danych, która pamięta liczby i ma następujące operacje: insert, min, deletemin

Warunki na drzewo:

OK:

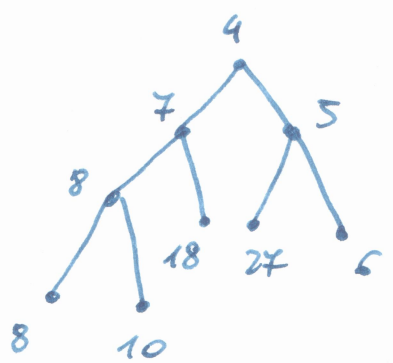


NIE OK:

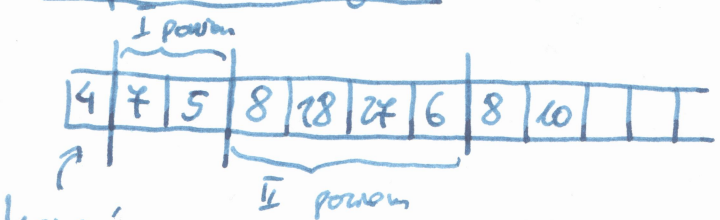
Warunek na porządek liczbowy (porządek kopcowy):

$\forall v \neq \text{korzeń} \quad \text{klucz}(v) \geq \text{klucz}(\text{ojciec}(v))$

Przykład

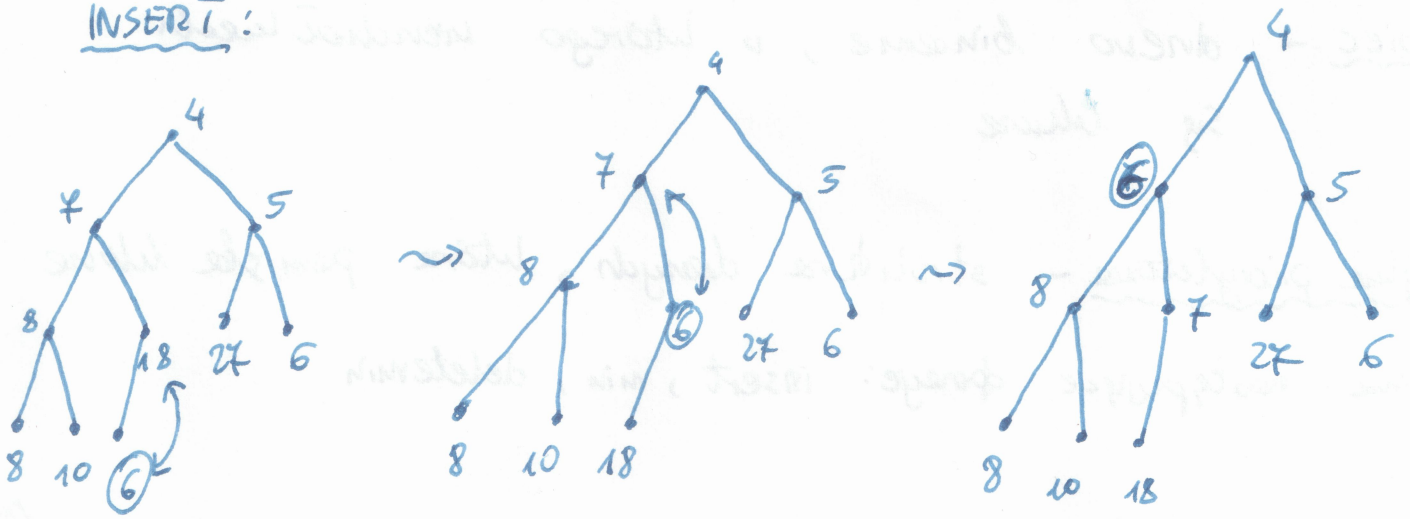


Implementacja:



MIN: return $H[1]$

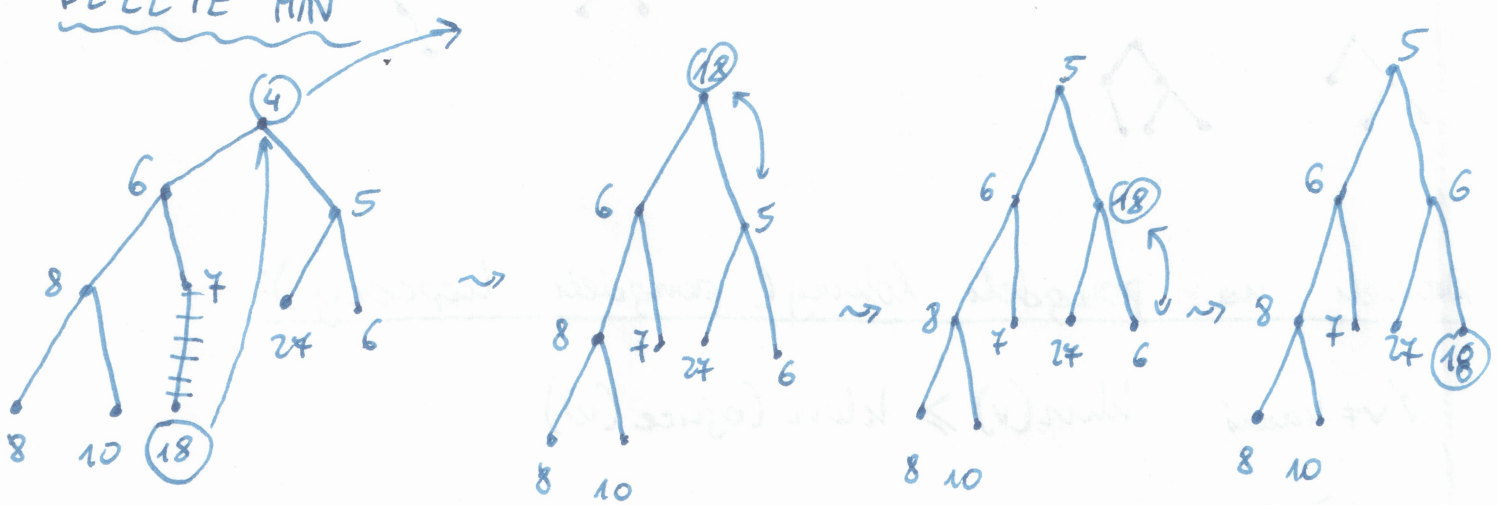
INSERT:



Obs: synowie $H[i]$ są w $2i$ oraz $2i+1$

ojciec $H[i]$ jest w $H[\lfloor i/2 \rfloor]$

DELETE MIN



Zamieniamy (18) z mniejszym synem.

delete_min:

- zamień $H[1]$ z $H[\text{rozmiar}--]$
- "zgrzeb" z $H[1]$ w dół

Obs:

W kopcu o wysokości h jest $\Theta(2^h)$ wierzchołków.

n

Podwójne kolejki priorytetowe

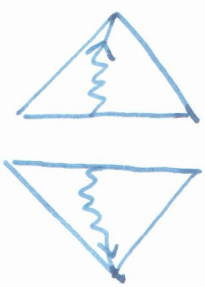
Dostępne operacje:

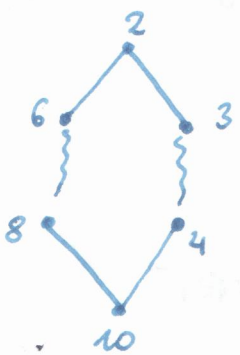
INSERT, MIN, DEL-MIN, MAX, DEL-MAX

Pomysł:

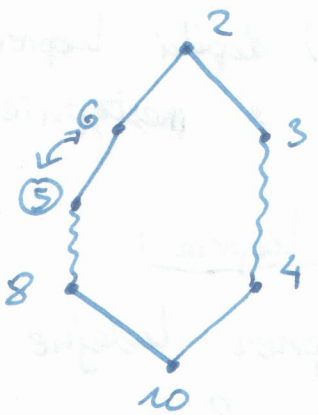
Budujemy dwa kopce: klucze małe i klucze duże

Dodawanie:

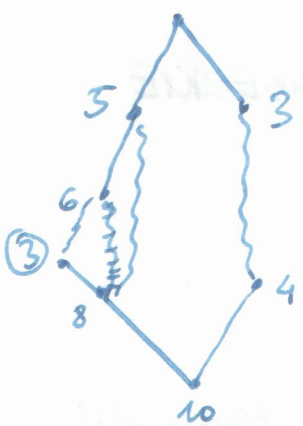




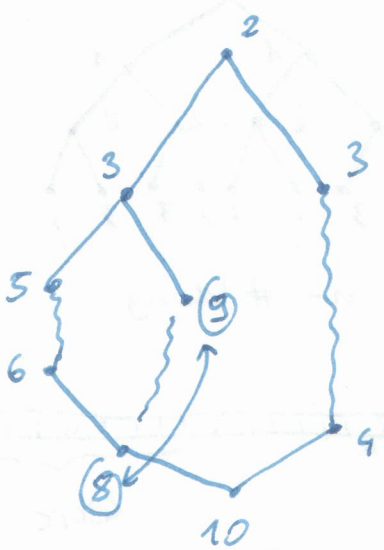
Dorzucamy ⑤



Dorzucamy ③ na dole



no i przepychamy teraz



Usuwanie:

Podobnie, ale jak z górnego usuwamy i się zmniejsza, to przerzucamy z jednego do drugiego. Jak usuwamy max, to bierzemy albo z górnego albo z dolnego — to zależy który jest większy

Koszt operacji kopca:

MIN → $\Theta(1)$

INSERT → $O(\text{wys. kopca}) = O(\log n)$, gdzie n to # kluczy

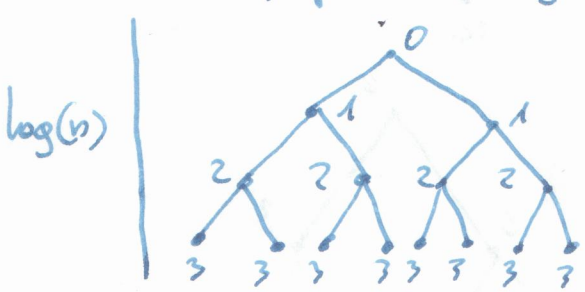
DELETE_MIN → $O(\log n)$

Inne zastosowania kopca:

- Heap sort
  (i) umieść wszystkie klucze w kopcu
  (ii) dopóki kopiec jest niepusty wyjmuj MIN,
  a następnie DELETE_MIN

Budowa kopca:

- poprzez kolejne operacje INSERT

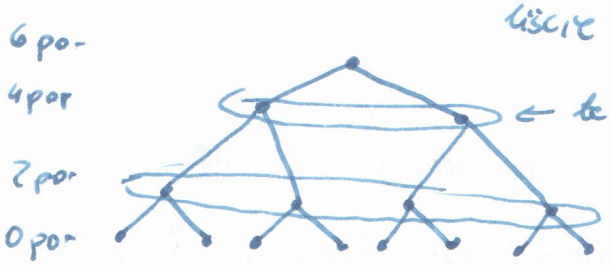


← # porównań przy INSERCIE

Oraz: n - # kluczy

liście

← dla kopca o wysokości h mamy $2\log n$ porównań



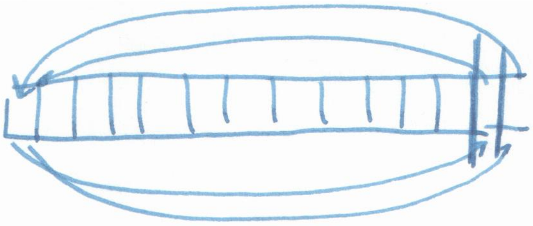
Koszt $\Theta(n)$

$n/2$ klucze

$\Omega(n \log n)$ ∧ $O(n \log n)$

⇓

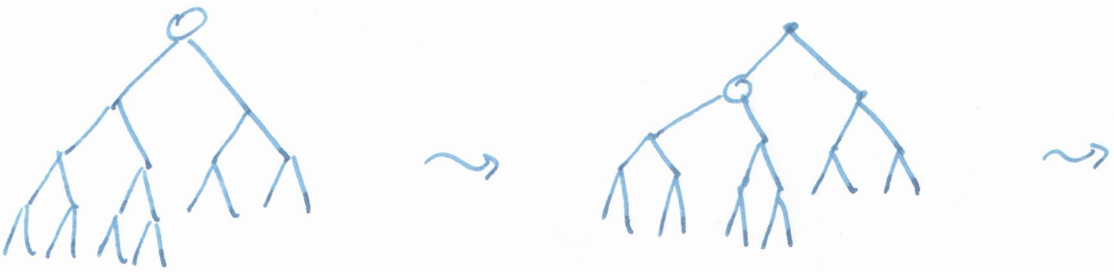
$\Theta(n \log n)$



przepisujemy i skręcamy kopiec i możemy tam usunąć pośrednie elem (małego)

Przyspieszenie delete:

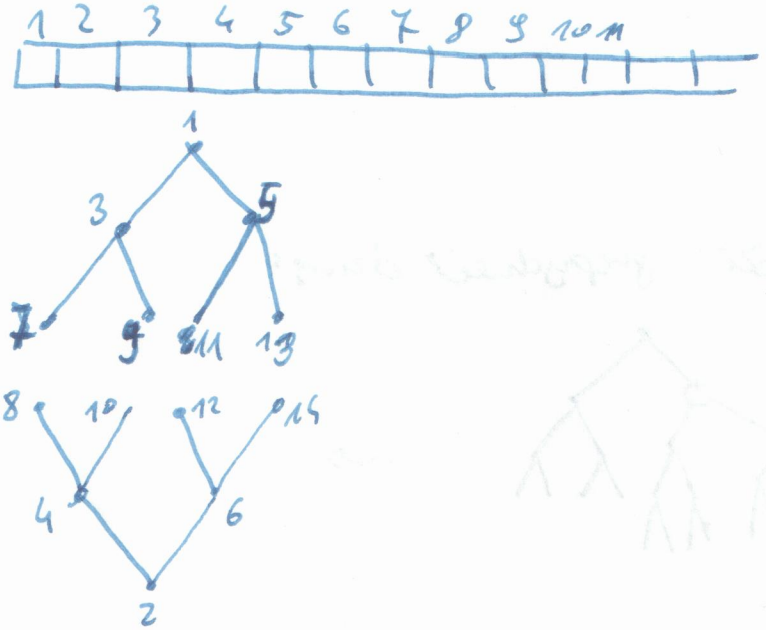
Będziemy zamiast wartości przepychać dziurę:



Więc poprawimy złą stronę drzewa. Jest możliwe, że strzeliliśmy porządek kopcowy, zatem trzeba przepchnąć w górę. Zysk jest na tym, że przepychając dziurę wykonamy 1 porównanie. Okazuje się, że wartość oczekiwana przepychania ledwie to 2. Zatem mamy $\log n + 2$. Jest tak dlatego, że w dwóch ostatnich poziomach jest

ok $n/2$ dzieci liści.

Pomysł na implementację:



Pomysł jest, aby trzymać kopiec w jednej tablicy, ale elementy z dwóch kopców są ułożone na przemian. Wówczas dodając element zawsze dodamy na wolnym kopiec dodając za aktualne ostatnim elementem w tablicy.

Usunięcie jest podobnie proste. Wtedy tylko pamiętać przy usuwaniu lub dodawaniu, aby przepychać tak długo aż będzie wszystko dobrze. Czasem trzeba będzie przechodzić z jednego kopca do drugiego i przepychać dalej.