HEAP: 14/03/22

A HEAP IS AN ABSTRACT DATA TYPES WHICH STORE TOTALLY ORDERED VALUES ($\preceq$)

PARTIAL ORDER IN WHICH ANY TWO ELEMENTS ARE COMPARABLE. IT IS A BINARY RELATION ON SOME SET X.

- $\preceq$ IS $\leq$, FOR a, b, c IN X:
  1. REFLEXIVE ← $a \leq a$
  2. TRANSITIVE ← $a \leq b$, $b \leq c$ THEN $a \leq c$
  3. ANTISYMMETRIC ← $a \leq b$, $b \leq a$ THEN $a = b$
  4. STRONGLY CONNECTED ← $a \leq b$ OR $b \leq a$

- SUPPORTED OPERATIONS →
  1. BUILD HEAP FROM SET OF DATA;
  2. FINDING THE MINIMUM ($\preceq$);
  3. EXTRACTING THE MINIMUM ($\preceq$);
  4. DECREASING ONE OF THE VALUES ($\preceq$);
  5. INSERT A NEW VALUE.

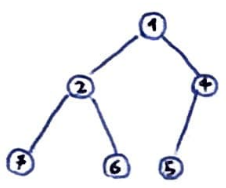
HEAPS ARE USEFUL TO IMPLEMENT PRIORITY QUEUES (SERIOUS PATIENTS MUST BE SERVED FIRST). IN OUR CASE THERE ARE TWO POSSIBLE HEAP:

- MIN-HEAP → HEAP WITH $\preceq \; = \; \leq$
- MAX-HEAP → HEAP WITH $\preceq \; = \; \geq$

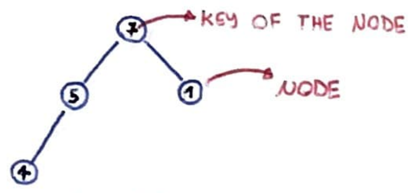
BINARY HEAP → IS A NEARLY COMPLETE BINARY TREE WHICH KEEPS THE HEAP PROPERTY.

COMPLETE UP TO THE SECOND LAST LEVEL AND ALL LEAVES OF THE LAST LEVEL ARE ON THE LEFT.

$\text{PARENT}(P).\text{KEY} \preceq P.\text{KEY} \quad \forall \, p$

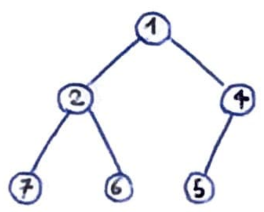


MIN-HEAP



MAX-HEAP

IS POSSIBLE TO USE AN ARRAY REPRESENTATION FOR A HEAP.



1	2	4	7	6	5		

OVERSIZE ARRAY

- ROOT → $i = 1$
- LEFT-CHILD → $2i$
- RIGHT-CHILD → $2i+1$
- PARENT → $\lfloor i/2 \rfloor$

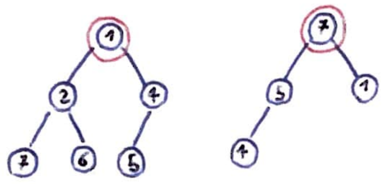
```
DEF PARENT(i):
    RETURN FLOOR(i/2)
ENDDEF

DEF LEFT(i):
    RETURN 2·i
ENDDEF

DEF RIGHT(i):
    RETURN 2·i +1
ENDDEF
```

OPERATIONS:

- FIND THE MINIMUM → THE MINIMUM WITH RESPECT TO $\preceq$ IS IN THE ROOT OF THE HEAP.



IN THIS CASE TO FIND THE MINIMUM WE CAN LOOK TO THE KEY VALUE OF THE ROOT NODE.

```
DEF FIND_MIN (H):
    RETURN H.ROOT.KEY
ENDDEF     H[1]
```

$\Theta(1)$ → IT IS COSTANT SINCE WE ARE JUST ACCESSING A POSITION OF THE ARRAY

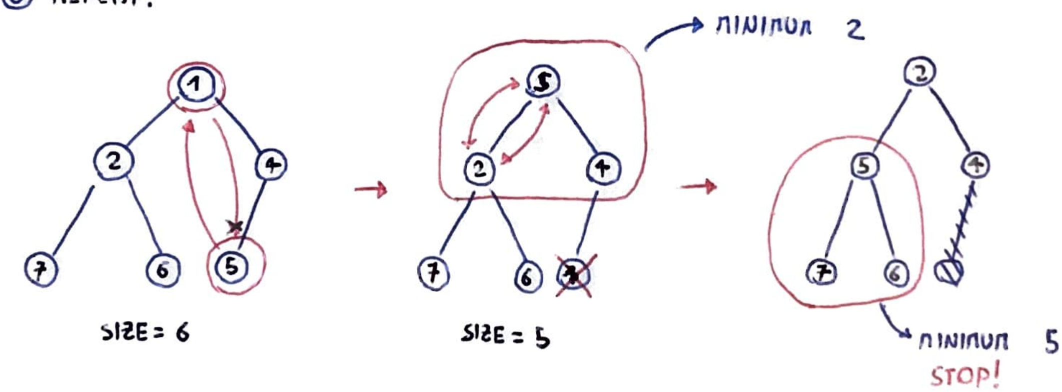
- EXTRACT THE MINIMUM → WHEN EXTRACT THE MINIMUM WE MUST PRESERVE THE TOPOLOGICAL STRUCTURE AND THE HEAP PROPERTY.

1. REPLACE THE ROOT KEY WITH THE KEY OF THE LAST NODE;
2. DECREASE THE SIZE OF THE NUMBER OF NODES (-1);

WE ARE NOT PRESERVING THE HEAP PROTERTY

HEAPIFY

3. FIND THE NODE m, AMONG THE ROOT AND ITS CHILDREN, WHOSE KEY IS MINIMUM W.R.T. $\preceq$;
4. IF THE ROOT'S KEY IS MINIMUM, STOP!
5. SWAP m'S AND ROOT'S KEYS;
6. REPEAT.



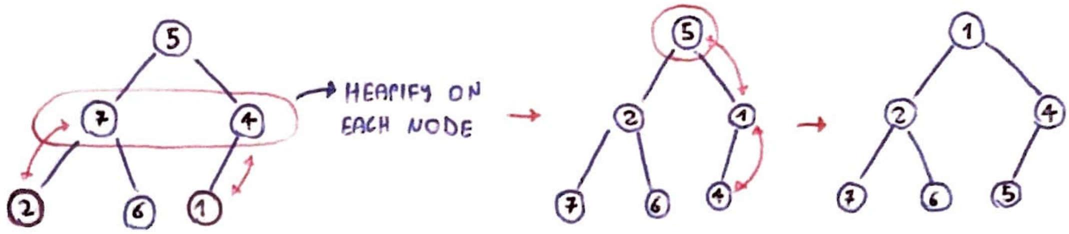
HEAPIFY FIXES THE HEAP PROPERTY IN THE ROOT.

```
DEF HEAPIFY (H, i):
    IF IS_VALIDE_NODE (H, LEFT(i)):
        IF IS_VALIDE_NODE(H, RIGHT(i)):
            m ← MIN (LEFT(i), RIGHT(i))
        ELSE:
            m ← LEFT(i)
    ELSE:
        m ← i
    IF i != m:
        SWAP(H, i, m)
        HEAPIFY(H, m)
ENDDEF
```

```
DEF REMOVE_MIN (H):
    H[1] ← H[H.SIZE]
    H.SIZE ← H.SIZE -1
    HEAPIFY (H,1)
ENDDEF
```

$O(\log(n))$ → IN THE WORST CASE WE NEED TO SWAP IT FROM THE NODE UNTIL A LEAF.

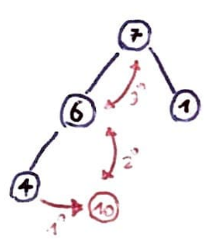
- **BUILD HEAP** → BUILD AN HEAP IS POSSIBLE USING HEAPIFY ON THE PARENT OF THE SECOND LEVEL UNTIL THE ROOT.




```
DEF BUILD_HEAP(A):
    A.SIZE ← |A|
    FOR i ← PARENT(A.SIZE) DOWN TO 1:
        HEAPIFY(A, i)
    ENDFOR
    RETURN A
ENDDEF
```

$$T_{bh}(n) \le \sum_{h=0}^{\log_2 n} \underbrace{\left\lceil \frac{n}{2^{h+1}} \right\rceil}_{\text{N° OF NODES AT LEVEL } h} * \overbrace{(c*h)}^{\text{HEAPIFY COST}} \le \sum_{h=0}^{\log_2 n} \frac{n}{2^h} * (c*h) \le c\cdot n \cdot \sum_{h=0}^{\log_2 n} \frac{h}{2^h} \le c \cdot n \cdot \frac{\frac{1}{2}}{(1-\frac{1}{2})^2} = 2\cdot c\cdot n \in O(n)$$

- **DECREASE KEY** → DECREASE W.R.T. $\preceq$.



WHEN WE DECREASE A KEY OF A NODE, THE PROBLEM IS FORWARD IN THE UPPER NODES.

1. IF KEY NODE $\le$ W.R.T $\preceq$ OF P(NODE) KEY, SWAP!
2. REPEAT UNTIL THE HEAP PROPERTY IS RESTORED OR IN THE WORST CASE WE REACH THE ROOT.

```
DEF DECREASE_KEY(H, i, VALUE):
    H[i] ← VALUE
    WHILE NOT (IS_ROOT(i) OR H[PARENT(i)] ⪯ H[i]):
        SWAP(H, i, PARENT(i))
        i ← PARENT(i)
    ENDWHILE
ENDDEF
```

$O(\log(n))$ → WORST CASE WE REACH THE ROOT FROM A LEAF

- **INSERTING A NEW VALUE** →
  1. ADD A NEW NODE NEXT TO THE LAST POSITION WITH THE KEY = NEW VALUE;
  2. DECREASE IT UNTIL THE HEAP PROPERTY IS FIXED.

```
DEF INSERT(H, VALUE):
    H.SIZE ← H.SIZE + 1
    H[H.SIZE] ← ∞ ⪯
    DECREASE_KEY(H, H.SIZE, VALUE)
```

$O(\log(n))$