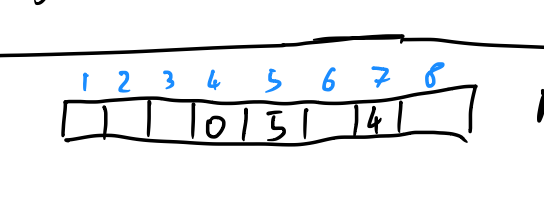
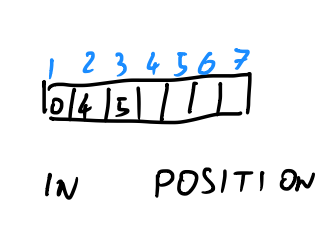
HEAPs

A HEAP MAKES IT POSSIBLE TO STORE N (KEY, VALUE) PAIRS (WHERE N IS FIXED A PRIORI), SO THAT:

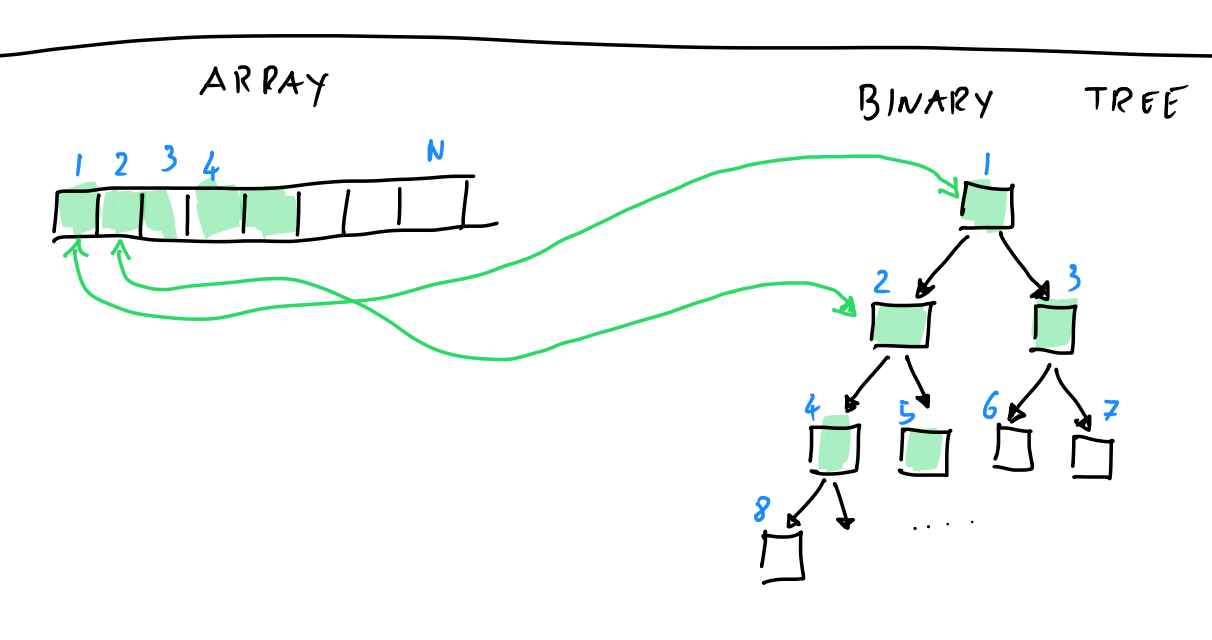
- THE ITEM WITH THE LOWEST KEY CAN BE ACCESSED IN $O(1)$ TIME;
- THE ITEM IN POSITION i OF THE HEAP CAN BE DELETED FROM THE HEAP IN $O(\lg n)$ TIME (HERE n IS THE CURRENT NUMBER OF PAIRS IN THE HEAP; THEN $O(\lg n) \le O(\lg N)$).
- A NEW ITEM CAN BE INSERTED IN THE HEAP IN $O(\lg n)$.

1-DIM. ARRAY  N

- REMOVE THE ITEM IN POSITION i TAKES $O(1)$ TIME
  ADD AN
- FINDING THE LOWEST KEY (FIND-MIN) TAKES $O(n)$ TIME.

1-DIM <u>SORTED</u> ARRAY 

- REMOVE THE ITEM IN POSITION i TAKES $O(n)$ TIME
  ADD AN
- FINDING THE LOWEST KEY TAKES $O(1)$ TIME



THIS LOGICAL REPRESENTATION OF THE TREE HAS THE FOLLOWING PROPERTIES:

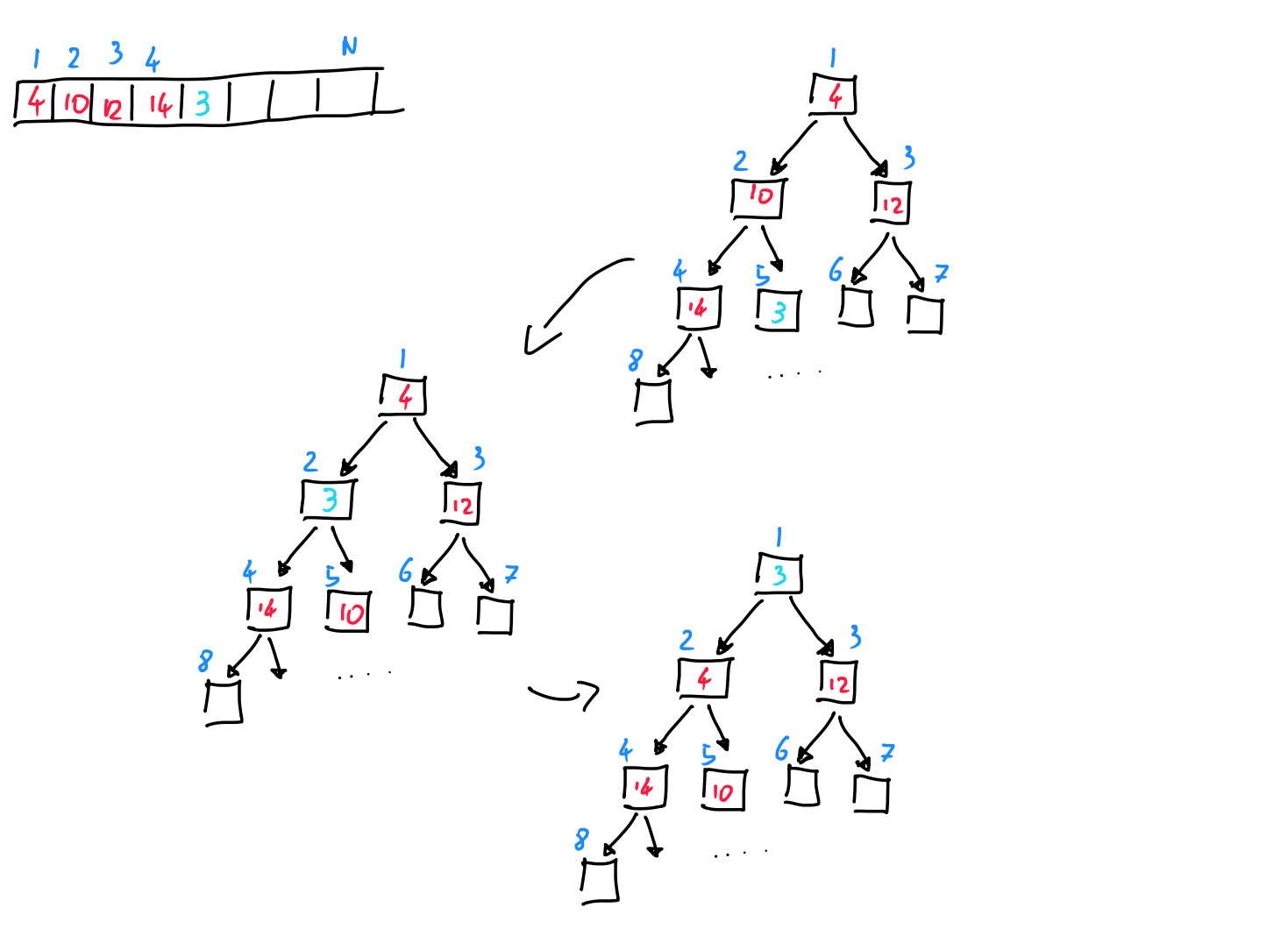
- THE LEFT CHILD OF NODE i HAS INDEX $2i \triangleq left(i)$
- " RIGHT " " " i " " $2i+1 \triangleq right(i)$
- THE PARENT " " i " " $\lfloor \frac{i}{2} \rfloor \triangleq parent(i)$.

THE HEAP HAS TO BE INITIALIZED A PRIORI (WE NEED TO KNOW N WHEN WE CREATE THE HEAP).

IF, AT SOME POINT, THE HEAP CONTAINS $n \le N$ ITEMS, THEN ONLY THE FIRST n POSITIONS OF THE ARRAY WILL BE FILLED.

HEAP PROPERTY: A HEAP SATISFIES THE HEAP PROPERTY IF, FOR EACH FILLED POSITION $i \ge 2$ OF THE HEAP, IF v IS THE ITEM IN POSITION i, AND IF w IS THE ITEM IN POSITION $parent(i) \left(= \lfloor \frac{i}{2} \rfloor\right)$ THEN:

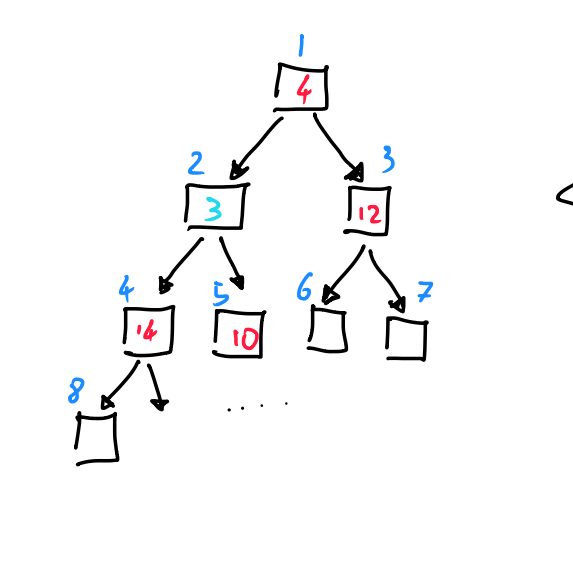
$$v.VALUE \ge w.VALUE$$



HEAPIFY-UP (H, i):
// THIS FUNCTION "PUSHES UP" THE ERROR IN POSITION i

```
IF i ≥ 2:
    // i HAS A PARENT
    j = parent(i)
    IF H[i].VALUE < H[j].VALUE:
        // THE HEAP PROPERTY FAILS AT i
        SWAP H[i], H[j]
        HEAPIFY-UP(H, j).
```

DEF.: WE SAY THAT H IS AN ALMOST HEAP WITH i TOO SMALL (AHWiTS) IF IT IS POSSIBLE TO INCREASE THE VALUE OF $H[i]$ TO SOME VALUE $\alpha \ge H[i].VALUE$ SO THAT THE RESULTING HEAP HAS THE HEAP PROPERTY.



← AHW$_2$TS

L: THE FUNCTION HEAPIFY-UP (H, i) FIXES THE HEAP PROPERTY OF H (PROVIDED IT NEEDS FIXING) IF H HAS THE AHWiTS PROPERTY.

THE RUNTIME OF HEAPIFY-UP(H, i) IS $O(\lg n)$.

P: WE PROVE IT BY INDUCTION ON i.

IF $i = 1$, THEN THE PROPERTY HOLDS (i HAS NO PARENT).

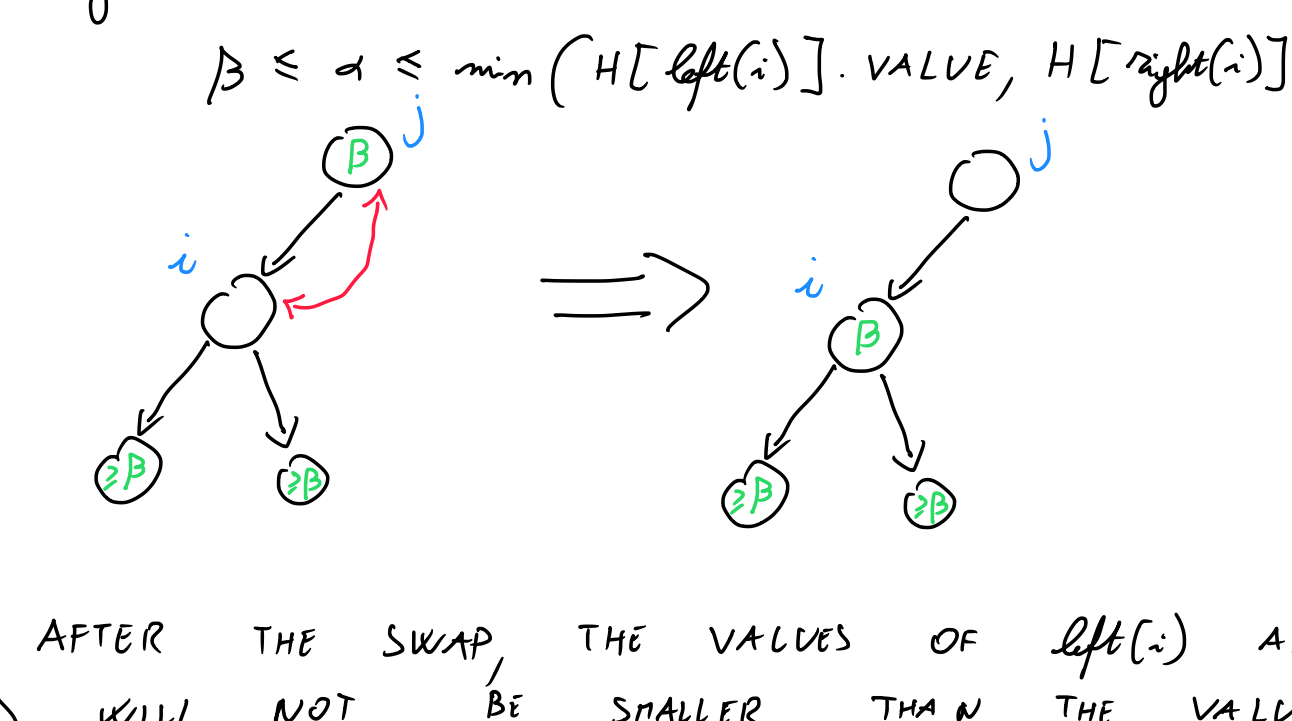
IF $i \ge 2$, THEN THE ALG. SWAPS $H[i]$ AND $H[j]$ (WITH $j = parent(i) = \lfloor \frac{i}{2} \rfloor < i$) IF $H[i].VALUE < H[j].VALUE$.

LET β BE EQUAL TO $\beta = H[j].VALUE$, <u>BEFORE</u> THE SWAP.

BECAUSE OF THE AHWiTS PROPERTY $\exists\ \alpha \ge H[i].VALUE$ (WHERE $H[i].VALUE$ IS TAKEN BEFORE THE SWAP) SUCH THAT IF THE VALUE OF $H[i]$ BECOMES α, THE HEAP IS FIXED.

BUT, THEN, $\alpha \ge \beta$ AND $\alpha \le H[left(i)].VALUE$ AND $\alpha \le H[right(i)].VALUE$. THUS,

$$\beta \le \alpha \le \min\left(H[left(i)].VALUE,\ H[right(i)].VALUE\right).$$



THUS, AFTER THE SWAP, THE VALUES OF $left(i)$ AND OF $right(i)$ WILL <u>NOT</u> BE SMALLER THAN THE VALUE OF i.

SINCE WE MADE THE SWAP, THE NEW VALUE OF i WILL NOT BE SMALLER THAN THE NEW VALUE OF j.

THUS, AFTER THE SWAP, i WILL BE CORRECTLY POSITIONED WITH RESPECT TO $left(i)$, $right(i)$ AND $j = parent(i)$.

WHAT ABOUT j? j MIGHT NOT BE CORRECTLY POSITIONED WRT ITS PARENT.

BUT, CLEARLY, THE NEW HEAP SATISFIES THE AHW$_j$TS PROPERTY. THE RECURSIVE CALL TO $j < i$ THEN SOLVES THE PROBLEM BY INDUCTION.

THE AMOUNT OF WORK IN EACH CALL TO HEAPIFY-UP IS $O(1)$ PLUS AN EXTRA CALL TO A NODE W A LEVEL CLOSER TO THE ROOT. GIVEN THAT THE BINARY TREE IS BALANCED, IT HAS AT MOST $O(\lg n)$ LEVELS. THE RUNTIME IS THEN

$$O(1) \cdot O(\lg n) = O(\lg n). \ \square$$

USING HEAPIFY-UP WE CAN IMPLEMENT THE ADD OPERATION

```
ADD(H, v, n):
    // n IS THE # OF ITEMS CURRENTLY IN THE HEAP
    IF n ≤ N-1:        // IF THERE IS SPACE FOR v
        n += 1
        H[n] = v
        // NOW, THE HEAP SATISFIES THE AHWnTS PROPERTY
        HEAPIFY-UP(H, n)
```

HOW TO REMOVE ITEMS?

DEF: WE SAY THAT H IS AN ALMOST HEAP WITH i TOO LARGE IF $\exists\ \alpha \le H[i].VALUE$ SUCH THAT IF WE DECREASE THE VALUE OF $H[i]$ TO α, WE HAVE THE HEAP PROP.

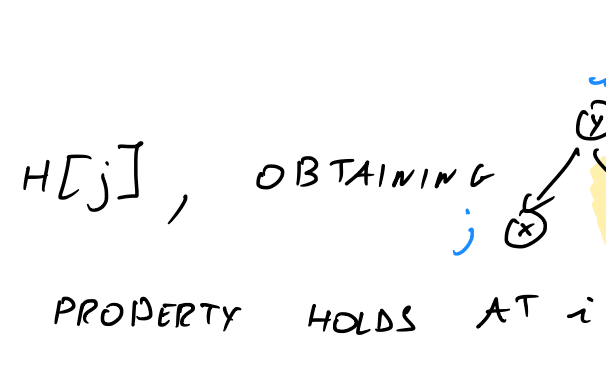
```
HEAPIFY-DOWN(H, i):
    // LET n BE THE CURRENT # OF ITEMS IN H, 1 ≤ i ≤ n
    IF 2i > n:
        // i HAS NO CHILDREN
        RETURN
    ELIF 2i == n:
        // i HAS ONLY THE LEFT CHILD
        j = 2i
    ELSE:
        // i HAS TWO CHILDREN
        IF H[left(i)].VALUE < H[right(i)].VALUE:
            j = left(i)
        ELSE
            j = right(i)
    IF H[j].VALUE < H[i].VALUE:
        SWAP H[i] AND H[j]
        HEAPIFY-DOWN(H, j)
```

L: HEAPIFY-DOWN (H, i) FIXES THE HEAP PROPERTY OF H, IF H HAS THE AHWiTL PROPERTY IN TIME $O(\lg n)$.

P: WE PROVE THE CLAIM BY REVERSE INDUCTION (OUR BASE CASE IS $i = n$).

IN GENERAL, IF $2i > n$, THEN i HAS NO CHILDREN, SO THERE IS NOTHING TO FIX.

O/W, $2i \le n$. IF THE ALGORITHM SWAPS $H[i]$ WITH $H[j]$, EITHER j IS THE ONLY CHILD OF i, OR i HAS A SECOND CHILD K, WITH $H[K].VALUE \ge H[j].VALUE$.

BEFORE THE SWAP, WE HAVE  .

THEN, $y = H[j].VALUE \le \min(x, z)$.

THE ALGORITHM SWAPS $H[i]$ AND $H[j]$, OBTAINING

GIVEN THAT $y \le \min(x, z)$, THE HEAP PROPERTY HOLDS AT i.

SINCE $j > i$, BY IND. THE ALGORITHM DEALS CORRECTLY WITH j IN ITS RECURSIVE CALL (SINCE THE HEAP NOW SATISFIES THE AHW$_j$TL PROPERTY).

THE RUNTIME IS $O(\lg n)$ SINCE EACH CALL TAKES $O(1)$ TIME, AND THERE ARE $O(\lg n)$ CALLS. $\square$

```
REMOVE(H, i):    // REMOVE THE ITEM IN POSITION i
    LET n BE THE NUMBER OF ITEMS IN H
    SWAP H[i] WITH H[n]
    H[n] = NONE
    n -= 1
    IF H[i].VALUE < H[parent(i)].VALUE:
        // AHWiTS
        HEAPIFY-UP(H, i)
    ELSE H[i].VALUE > min(H[left(i)].VALUE, H[right(i)].VALUE):
        // AHWiTL
        HEAPIFY-DOWN(H, i)
```

T: ADD(...) ADDS AN ITEM IN TIME $O(\lg n)$.
REMOVE(...) REMOVES THE ITEM IN A POSITION IN TIME $O(\lg n)$.

```
DEF HEAPSORT(V):    // LET V BE AN ARRAY OF LENGTH N
    INITIALIZE A HEAP H OF SIZE N
    FOR i = 0, 1, ..., N-1
        ADD(H, V[i])              O(lg N) } O(N lg N)
    FOR i = 0, 1, ..., N-1
        V[i] = H[1]
        REMOVE(H, 1)              O(lg(N)) } O(N lg N)
    RETURN V.
```

$O(N \lg N)$