DSA [02/14/24]

PRIORITY QUEUE

NEED DEFAULT CONSTRUCTOR TO MAKE EMPTY HEAP

HEAP - - - HEAP SORT

ARRAY ⟵ FIXED SIZE. DON'T INSERT PAST SIZE!

BASIC OPERATIONS:

- HEAPIFY ↓ $O(\lg n)$ PRESERVE HEAP PROPERTY
- BUILD HEAP* $O(n)$ ARRAY ⟶ HEAP
- HEAP SORT $O(n \lg n)$ HEAP ⟶ SORTED ARRAY
- HEAP INSERT $O(\lg n)$ ADD NEW VALUE
- EXTRACT MAX $O(\lg n)$ REMOVE TOP/FRONT (MAX HEAP)
- INCREASE KEY ↑ $O(\lg n)$ PROMOTE ELEMENT; SUPPORT FOR INSERT

PUT COMPLEXITY ON SUPPORT

SUPPORT FUNCTIONS:

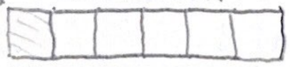
- SWAP(i, j) ⟶ SWAPS A[i] AND A[j]
- PARENT(i) ⎤
- (i*2+2) ⟶ LEFT(i) ⎥ RETURN INDEX
- (i*2+1) ⟶ RIGHT(i) ⎦
- INDEX OF MAX(i, j, k) ⟵ RETURN INDEX OF MAX A[i], A[j], A[k]

(UP TO 3 ELMT) GARUNTEED GOOD INDEX POSSIBLY INVALID IF LEGAL/in BOUNDS

REVIEW: HEAP!

HEAP PROPERTY

PQ MAX HEAP:



↑ HIGHEST (NUMERIC PRIORITY

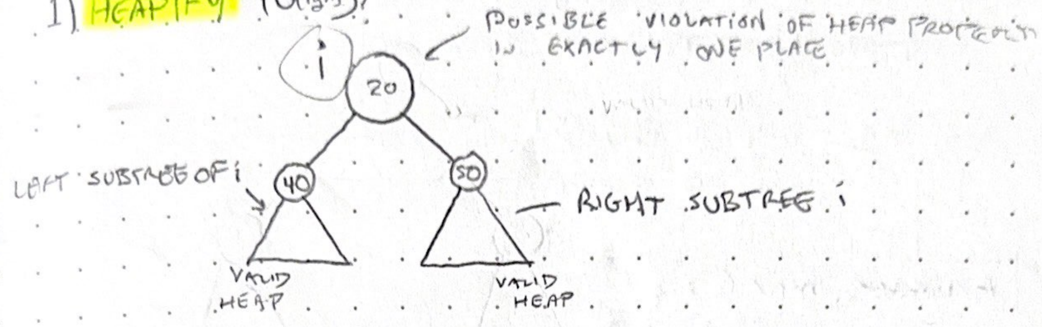
IN OUR PROJECT THE DATA ARE INTS!

VALUE OF NODE IS AT LEAST THE VALUE OF PARENT

$$A[i] \leq A[\text{PARENT}(i)]$$

BASIC OPERATIONS:

1) **HEAPIFY** (O(lg n))?



HEAPIFY WILL MOVE VIOLATION DOWN THE HEAP UNTIL PEACE IS RESTORED!

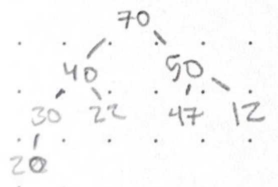
PSUEDO CODE:

CALL ANY TIME YOU THINK THERE IS A VIOLATION

```
heapify(i) {
    n = index_of_max (i, left(i), right(i))
    IF (n != i) {
        SWAP (i, n);     SWAP A[i] & A[n]
        heapify(n)
    }
}
```

CODE RECURSIVELY!

EXAMPLE HEAP:



IN ARRAY

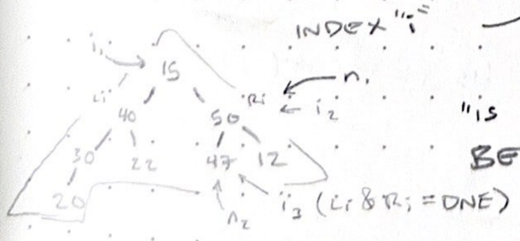
0	1	2	3	4	5	6	7
70	40	50	30	22	47	12	20

HEAPIFY FIXES THIS

LET'S SAY 15 IS A[0] →

15	40	50	30	22	47	12	20

INDEX "i"



"IS THE HEAD/FRONT SUPPOSED TO BE THERE? IF NOT, MOVE!"

COMPARE $\hat{}$ TO SIZE (BOUNDS) OF HEAP
Lᵢ & Rᵢ POSITION

DSA [02/14/24]

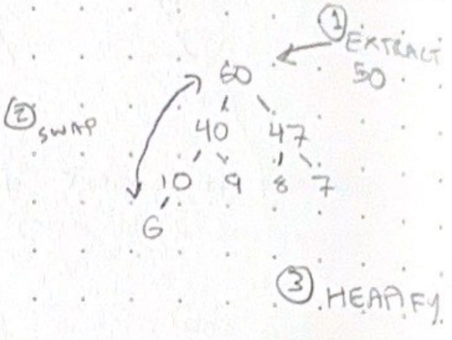
BASIC OPERATIONS! CONTINUED

2) EXTRACT MAX ($O(\lg n)$) - "HOW WE PULL STUFF"

- REMOVE FRONT ELEMENT FROM HEAP!

STEP #1: CHECK BOUNDS! YOU CAN'T EXTRACT IF THERE ARE NO ELEMENTS!

```
EXTRACT MAX () {
O(1) [ int max = A[0];
       A[0] = A[size - 1];
       size--;
O(lg n) - HEAPIFY(0)
O(1) - RETURN MAX;
}
```

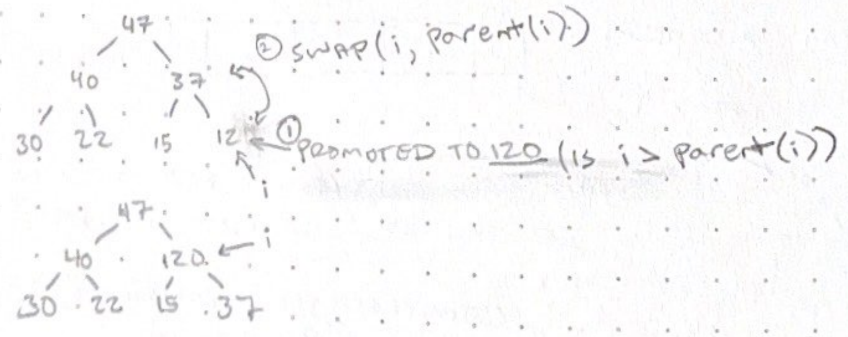


WE MAY WANT TO INCREASE VAL. TO MOVE PRIORITY!

PUBLIC! INDEX KEY

```
3) INCREASE KEY (i, k) { //O(lg n)
      //BOUNDS ETC. CHECKING
      // MAKE SURE IT IS IN BOUNDS AND AN ACTUAL INCREASE
      IF (i < 0 OR i > size - 1) OR (A[i] ≥ k) RETURN!   (IF = RETURN)
PROMOTE - [ A[i] = k; //PROMOTED A[i]
      while (i > 0 && A[parent(i)] < A[i] {
          swap(i, parent(i));
          i = parent(i);
      }
}
```

MOVE UP UNTIL IT IS IN THE RIGHT SPOT!



DSA [02/14/24]

BASIC OPERATIONS: CONTINUED

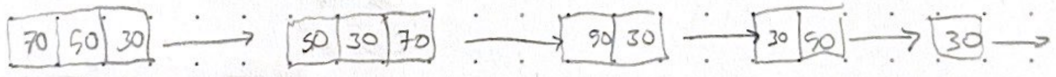
```
  PUBLIC
    v
4) HEAP INSERT (val) {         (O(lgn))
O(1)   [ size++;
       [ A[size -1] = val-1;  ← NOW WE CAN CALL INCREASE KEY
O(lgn) [ INCREASE_KEY (size-1, val);
   }
   //BOUNDS/SIZE CHECK
```

```
      UNLIKELY TO NEED IT, BUT REQUIRED TO CODE
      ↓
5) BUILD HEAP () { // O(n)
     for(int i = size/2 - 1;  i>=0; i--) {
         heapify(i); // O(lgn)
     }
   }
```

ANY VALUES IN AN ARRAY REARRANGED INTO A (MAX) HEAP

```
6) HEAP SORT () {     O(nlgn)
        for (int i=size-1; i>0; i--){
O(n)        SWAP(0,i);
            SIZE--;
            heapify(0); //O(lgn)
        }
   }
```

- REQUIRES HEAP
- RETURNS SORTED ARRAY (NOT HEAP)
- MUST BE EXACTLY THE SIZE OF THE SORTED ELEMENTS
- USE 2 ARRAYS AT MAX, ONE PASS OF COPYING



$$\begin{tikzcd} \text{HEAP1} \arrow[r, "\text{COPY}"] & \text{HEAP2} \arrow[d] \\ & \text{HEAPSORT(HEAP2)} \arrow[r] & \text{SORTED ARRAY} \end{tikzcd}$$