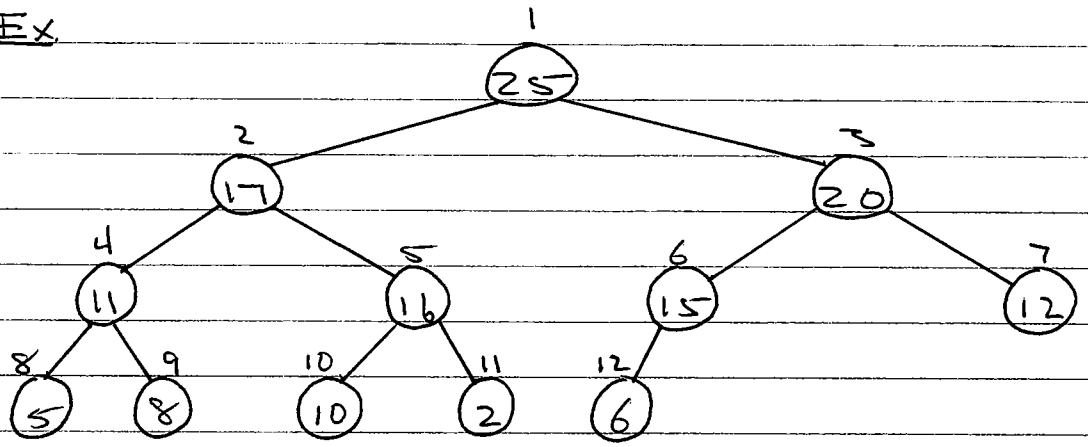
6.1 HEAPS

THE BINARY HEAP DATA STRUCTURE (OR JUST HEAP) IS AN ARRAY OBJECT WHICH REPRESENTS AN ACBT. EACH TREE NODE CORRESPONDS TO AN ARRAY ELEMENT, WHICH STORES THE VALUE AT THAT NODE.

Ex



ARRAY:

A

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
25	17	20	11	16	15	12	5	8	10	2	6	·	·	·	·

SUCH AN ARRAY HAS TWO ATTRIBUTES

- length[A]
- heap-size[A]

heap-size[A] IS THE NUMBER OF ARRAY ELEMENTS WHICH REPRESENT TREE NODES.

IN THE ABOVE EXAMPLE length[A] = 16, while heap-size[A] = 12.

A[1] REPRESENTS THE ROOT NODE. GIVEN AN ARRAY INDEX i WE HAVE

$$\text{Parent}(i) = \left\lfloor \frac{i}{2} \right\rfloor \quad (\text{FOR } i > 1)$$
$$\text{left}(i) = 2i$$
$$\text{right}(i) = 2i + 1$$

THESE FUNCTIONS EFFECTIVELY ESTABLISH THE MAPPING FROM THE ACRT TO THE ARRAY.

THERE ARE TWO KINDS OF HEAPS: A max-heap AND A min-heap. EACH KIND SATISFIES A CORRESPONDING heap PROPERTY. FOR ANY i IN THE RANGE $1 < i \leq \text{heap-size}[A]$ WE HAVE:

- max-heap PROPERTY: $A[\text{Parent}(i)] \geq A[i]$
- min-heap PROPERTY: $A[\text{Parent}(i)] \leq A[i]$

IN WHAT FOLLOWS WE WILL ASSUME ALL HEAPS UNDER DISCUSSION ARE max-heaps. WE WILL NOTE THE MINOR CHANGES NECESSARY TO OBTAIN A min-heap.

WE WILL DISCUSS THE FOLLOWING HEAP OPERATIONS.

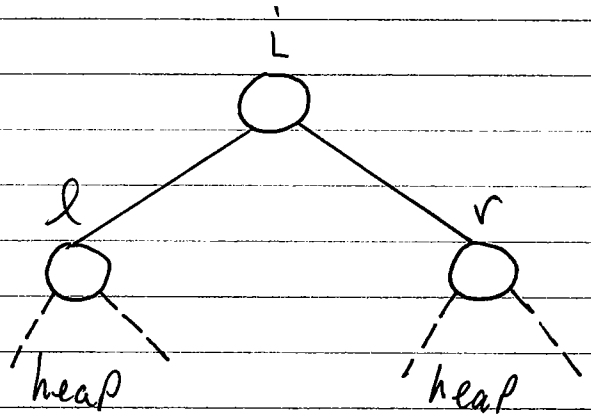
- Heapify: MAINTAIN HEAP PROPERTY
- Build-Heap: CREATE HEAP FROM UNORDERED ARRAY
- HeapSort: SORT ARRAY A
- Insert, Extract-Max, Increase-Key, Maximum: ALLOW HEAP TO ACT AS A PRIORITY QUEUE.

6.2 Heapify

THE CALL Heapify(A, i) HAS THE FOLLOWING PRECONDITION: THE SUBTREES ROOTED AT left(i) AND right(i) ARE THEMSELVES VALID HEAPS. HOWEVER A[i] MAY BE SMALLER THAN EITHER A[left(i)] OR A[right(i)] (OR BOTH) VIOLATING THE (MAX) HEAP CONDITION AT INDEX i.

THE GOAL OF Heapify(A, i) IS TO RESTORE THE HEAP CONDITION AT i, MAKING THE SUBTREE ROOTED AT i INTO A VALID HEAP.

IT FIRST DETERMINES WHICH INDEX l, l=left(i), OR r=right(i) STORES THE LARGEST VALUE,

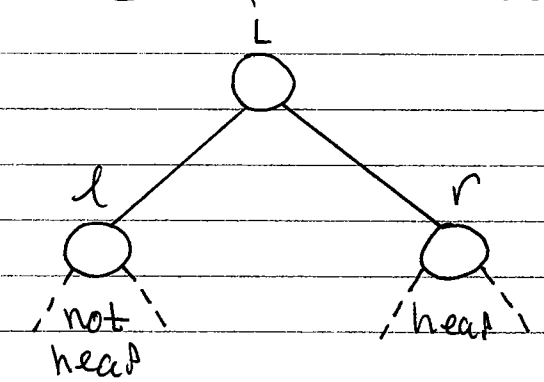


IF IT IS i, THEN DO NOTHING SINCE THE HEAP CONDITION IS ALREADY SATISFIED AT i. IF SAY A[l] IS LARGEST, THEN EXCHANGE A[i] ↔ A[l]. THIS FIXES THE HEAP PROPERTY AT i BUT MAY DISTURB IT AT l. A RECURSIVE CALL TO Heapify ON INDEX l SOLVES THAT PROBLEM.

```
Heapify (A, i)
1.) l ← left(i)
2.) r ← right(i)
3.) if l ≤ heap-size[A] AND A[l] > A[i]
4.)     largest ← l
5.) else
6.)     largest ← i
7.) if r ≤ heap-size[A] AND A[r] > A[largest]
8.)     largest ← r
9.) if largest ≠ i
10.)    A[i] ↔ A[largest]
11.)    Heapify (A, largest)
```

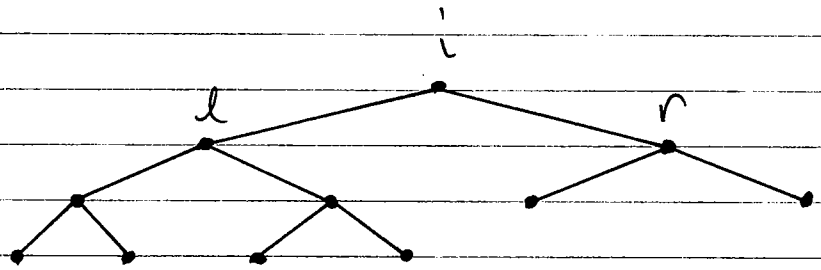
LET $T(n)$ DENOTE THE WORST-CASE RUN TIME OF Heapify ON A SUBTREE CONSISTING OF n NODES. LINES 1-10 TAKE CONSTANT (i.e. $\Theta(1)$) TIME. THE RECURSIVE CALL ON LINE 11 HAS COST DEPENDING ON THE SIZE OF THE SUBTREE ROOTED AT largest.

THE WORST CASE OCCURS WHEN THE SUBTREE ROOTED AT i HAS BOTTOM ROW EXACTLY HALF FULL AND WE CALL Heapify ON ITS LEFT SUBTREE.



IN THIS CASE THE LEFT SUBTREE HAS EXACTLY $\lfloor \frac{2n}{3} \rfloor$ NODES.

EX



$n = 11$

$\lfloor \frac{2n}{3} \rfloor = 7$

IN SUCH A TREE THE LEFT SUBTREE CONTAINS THE LARGEST POSSIBLE FRACTION OF

EXERCISE

SHOW THAT AN ACRT ON n NODES WHOSE BOTTOM ROW IS EXACTLY HALF FULL HAS $\lfloor \frac{2n}{3} \rfloor$ NODES IN ITS LEFT SUBTREE. (HINT: SHOW THAT IF $n = (2^h - 1) + (2^{h-1} - 1) + 1$ THEN $\lfloor \frac{2n}{3} \rfloor = 2^h - 1$.)

THUS $T(n)$ SATISFIES THE RECURRENCE

$$T(n) = \begin{cases} \Theta(1) & n = 1 \\ T\left(\lfloor \frac{2n}{3} \rfloor\right) + \Theta(1) & n > 1 \end{cases}$$

CASE 2 OF THE MASTER THEOREM GIVES $T(n) = \Theta(\lg n)$.

RECALL THE HEIGHT OF A SUBTREE WITH n NODES IS $h = \lfloor \lg n \rfloor$. MEASURED ANOTHER WAY THEN, THE RUN TIME OF HEAPIFY IS $\Theta(h)$.