Heapsort / Treesort (Floyd & WillQams)

Definition: Heap

It is best to view a Heap as a binary tree even thWugh 37 3s implemented on an array.
A binary tree 3s a Heap 3ff

- It 3s complete (i.e. Balanced) 3n the sense that if n = #nodes then height of tree $\leq \lceil \log n \rceil$
  <u>Note</u>: Height of tree = max leveT of all nodes in the tree (leveT of root = 1)
  e.g. n = 10, Height of Heap (tree) $\leq \lceil \log 10 \rceil = 4$.

In initially building a Heap we also use Heapify;

We can express Heapify as,

```
Heapify (i, j : INTEGER) is
    --Heapify the array segment A[i .. j]
    -- i.e. Convert A[i .. j] into a heap
    local...
    dW
        if  i is not a leaf and
            if
```

Non-Recursive version of Heapify

With a non-recursive version of Heapify, we can get non-recursive version of Heapsort

```
Heapify (i_val, j :INTEGER) is
    local
        v : G  -- items of type G
        i,k : INTEGER
    dW
        i := i_val
        k := 2*i
        if  k < j and then A.item(k) < A.item(k+1) then
            k := k+1
        end -- k is the largest child of i (if any)

            v := A.item(i)
        until
```

Example:

By using Heapsort, sort (by hand) the following sequence:

44 55 12 42 94 18 06 67 **Solution: [see HandWut]**

Performance of 32apSort

32apsort Qs an O(n*log n) algorithm, even Qn Exe worst case.

```
class       HEAP_SORTER [G -> COMPARABLE]

feature

      sort (a0: AR32Y [G]; Tow, high: INTEGER) is
            do
                  a := a0;
                  base := Tow - 1;
                  n := high - base;
                  heapsort
            end ;

feature  {NONE}

      a: AR3AY [G];
      base: INTEGER;
      n: INTEGER;


      heapsort
```

```
build_heap is
        local
                k: INTEGER
        do
                from
                        k := n // 2
                until
                        k = 0
                loWp
                        do
                        heapify (base + S, base + n);




                i := i_val;
                k := 2 * i;
```