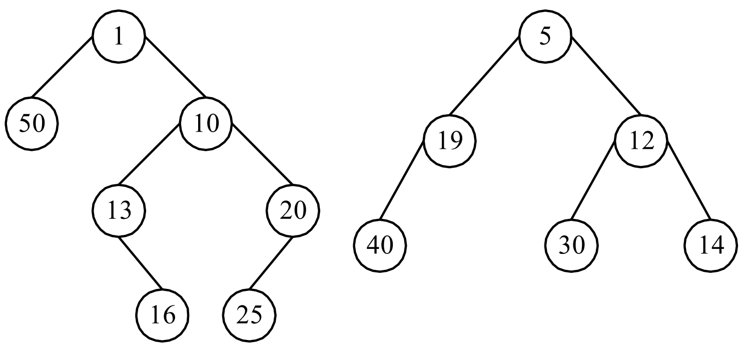
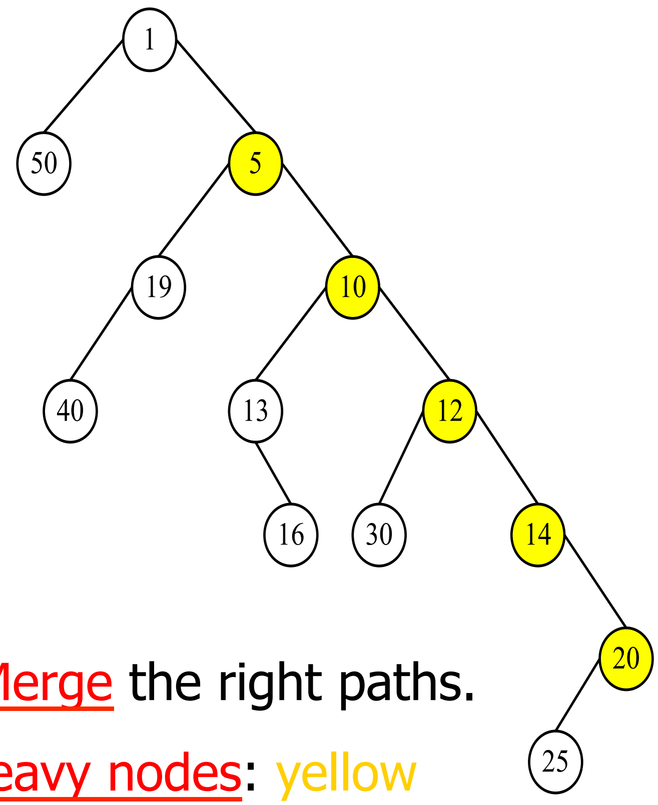
```
(* SML implementation of skew heaps *)

datatype PQ = Leaf | Node of (int * PQ * PQ)

fun meld (A,B) =
    case (A,B) of
        (_,Leaf) => A
      | (Leaf,_) => B
      | (Node(ka,La,Ra), Node(kb,Lb,Rb)) =>
        case Int.compare (ka, kb) of
            LESS => Node(ka, meld(Ra,B), La)
          | _    => Node(kb, meld(A,Rb), Lb)

fun deletemin A =
    case A of
        Leaf => (NONE,A)
      | Node(ka,La,Ra) => (SOME ka, meld(La,Ra))

fun insert (k,A) =
    let val n = Node(k,Leaf,Leaf) in
        meld(n,A)
    end

(* ocaml implementation of skew heaps *)

type 'a tree = Empty | Node of 'a tree * 'a * 'a tree

let rec meld a b =
  match (a,b) with (Empty, _) -> b | (_, Empty) -> a
    | (Node(al, ak, ar), Node(bl, bk, br)) ->
        if (ak <= bk) then Node((meld ar b), ak, al)
        else Node((meld a br), bk, bl)

let insert k a =
  let n = Node(Empty, k, Empty) in
  if (a = Empty) then n else meld a n

let findmin a =
  match a with Empty -> failwith "Findmin_on_empty_heap"
    | Node(al, ak, ar) -> ak

let deletemin a =
  match a with Empty -> failwith "Deletemin_on_empty_heap"
  | Node(al, ak, ar) -> meld al ar

let isempty a = a = Empty
```

Skew heaps

- meld: merge + swapping



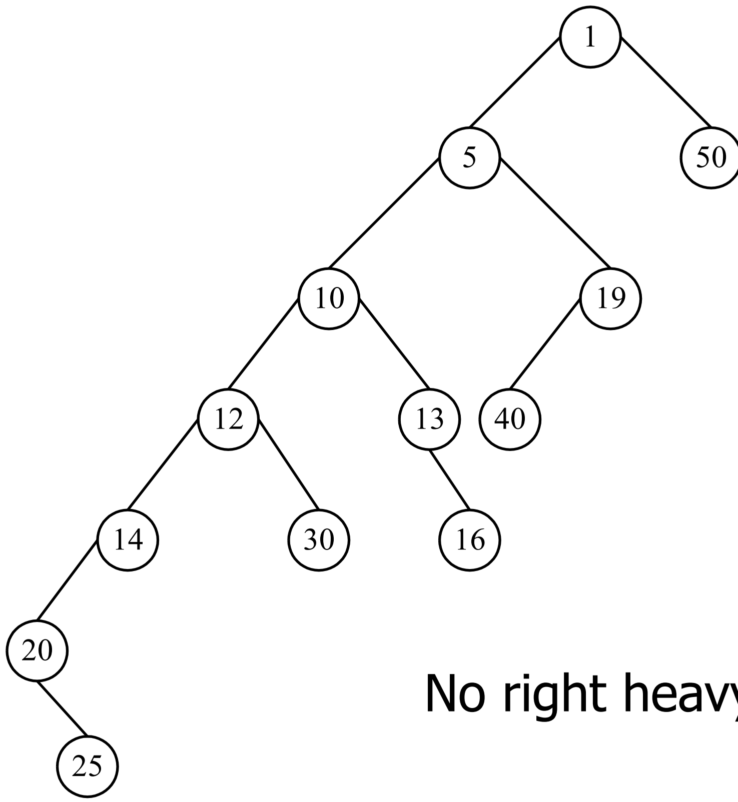
Two skew heaps



Step 1: Merge the right paths.

5 right heavy nodes: yellow

Step 2: Swap the children along the right path.



No right heavy node

Amortized analysis of skew heaps

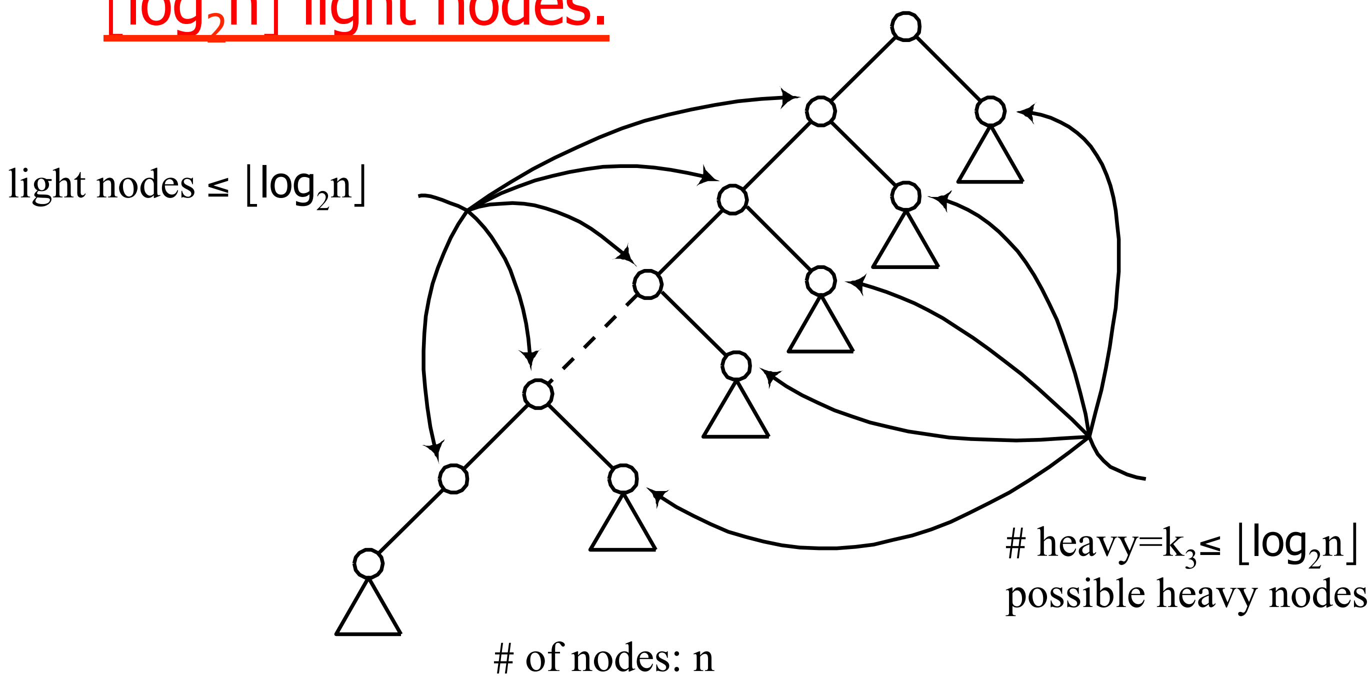
- meld: merge + swapping
- operations on a skew heap:
  - find-min(h): find the min of a skew heap h.
  - insert(x, h): insert x into a skew heap h.
  - delete-min(h): delete the min from a skew heap h.
  - meld(h_1, h_2): meld two skew heaps h_1 and h_2.

  The first three operations can be implemented by melding.

Potential function of skew heaps

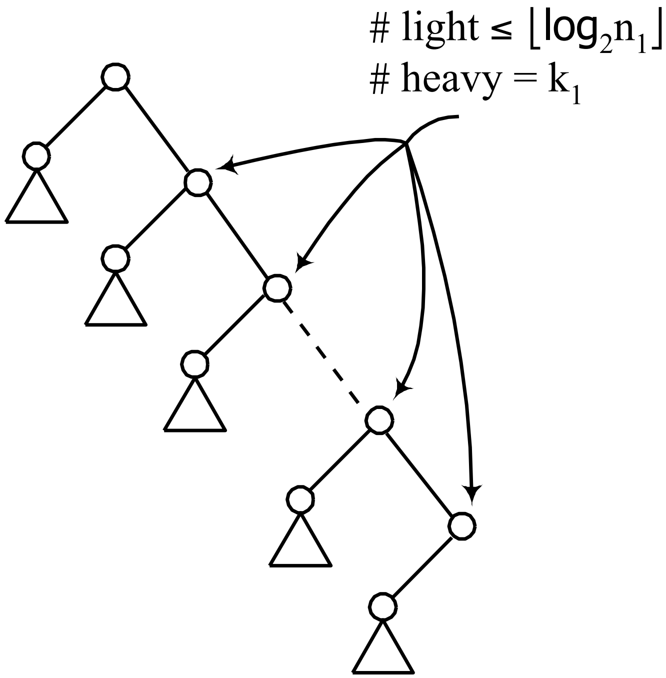
- wt(x): # of descendants of node x, including x.
- heavy node x: wt(x) > wt(p(x))/2, where p(x) is the parent node of x.
- light node : not a heavy node
- potential function Φ_i: # of right heavy nodes of the skew heap.

- Any path in an n-node tree contains at most $\lfloor \log_2 n \rfloor$ light nodes.



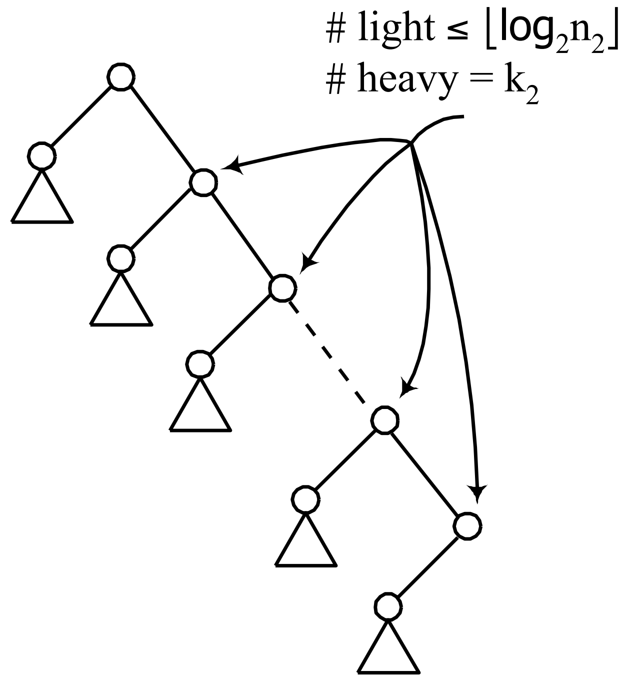
- The number of right heavy nodes attached to the left path is at most $\lfloor \log_2 n \rfloor$.

Amortized time



heap: h_1

# of nodes: n_1



heap: h_2

# of nodes: n_2

$a_i = t_i + \Phi_i - \Phi_{i-1}$

t_i : time spent by OP_i

$t_i \le 2+\lfloor \log_2 n_1 \rfloor + k_1 + \lfloor \log_2 n_2 \rfloor + k_2$

("2" counts the roots of h_1 and h_2)

$\le 2+2\lfloor \log_2 n \rfloor + k_1 + k_2$

where $n=n_1+n_2$

$\Phi_i - \Phi_{i-1} = k_3-(k_1+k_2) \le \lfloor \log_2 n \rfloor - k_1 - k_2$

$a_i = t_i + \Phi_i - \Phi_{i-1}$

$\le 2+2\lfloor \log_2 n \rfloor + k_1 + k_2 + \lfloor \log_2 n \rfloor - k_1 - k_2$

$= 2+3\lfloor \log_2 n \rfloor$

$\Rightarrow a_i = O(\log_2 n)$