Skew Heaps

Problems with leftist heaps
- Extra storage for npl
- Extra complexity/logic to maintain and check npl

Observation:
- Right side of leftist heap is “often” heavy and requires a switch.

Solution: skew heaps
- “Blindly” adjusting version of leftist heaps
- Merge always switches children when fixing right path
- Worst case time: merge, insert, deleteMin = O(n)
- Amortized time: merge, insert, deleteMin = O(log n)

Amortized vs. average complexity

Recall:
- Average-case complexity: \( \text{avg} \) # steps algorithm takes on random inputs of size \( N \)
- Amortized complexity (improved definition):
  \[ \text{max} \] total # steps algorithm takes, in the worst case, for \( M \) consecutive operations on inputs of size \( N \), divided by \( M \) (i.e., divide the max total by \( M \)).

Example: if \( M \) operations take total \( O(M \log N) \) time in the worst case, amortized time per operation is \( O(\log N) \).
Merging Two Skew Heaps

Only one step per iteration, with children always switched

Skew Heap Code

```c
SkewHeap merge(heap1, heap2) {
    case {
        heap1 == NULL: return heap2;
        heap2 == NULL: return heap1;
        heap1.findMin() <= heap2.findMin():
            temp = heap1.right;
            heap1.right = heap1.left;
            heap1.left = merge(heap2, temp);
            return heap1;
        otherwise: return merge(heap2, heap1);
    }
    return merge(heap2, heap1);
}
```

Example

Skew Heap Code

Skew Heap Code

Runtime Analysis:
Worst-case and Amortized

- No worst case guarantee on right path length!
- All operations rely on merge

⇒ worst case complexity of all ops =

- It is known: $M$ merges take time $\Theta(M \log n)$ in the worst case

⇒ amortized complexity of all ops =