Binomial Heaps

- A binomial tree of order 0 is a single node.
- A binomial tree of order $n+1$ has a root node whose children are binomial trees of orders $n, n-1, n-2, \ldots, 0$.
- binomial tree of order n has 2^n nodes.
  Proof: Clearly, binomial tree of order 0 has $2^0 = 1$ node.
  By induction suppose for $m < n$, binomial trees of order m have 2^m nodes. Then, the binomial tree of order n has $1 + \sum_{m<n} 2^m = 2^n$ nodes.
- Each binomial tree has a heap structure, that is key at a node is no larger than the keys at the children.
- Binomial Heap is collection of binomial trees, each with different order.

- Binomial heap with n nodes has the largest order tree of order at most $\log n$.
- In a binomial heap of max order n, there are at most $2^{n+1} - 1$ nodes.
  Proof: In worst case, there is a binomial tree of orders 0, 1, ..., n. Summing up their sizes gives the total number of nodes as $\sum_{i=0}^{n} 2^i$ which is $2^{n+1} - 1$.

- For order n binomial-tree, the number of nodes at depth d is $\binom{n}{d}$.
  Proof: For $n = 0$ or $d = 0$, this clearly holds.
  Suppose by induction it holds for $m < n$. Then, for order n binomial tree, the number of nodes at depth $d + 1$ is sum of the number of depth d nodes for each of the children of the root, that is:
  $\sum_{i=d}^{n-1} \binom{i}{d} = \binom{n}{d+1}$.

Combining/Merging

- Combining two binomial trees of equal order: Make one of the trees a child of other (smaller key root becomes the parent, and the other a child). Takes $O(1)$ time.
- Merge two binomial heaps: First sort the binomial trees in the collection based on their order.
- Combine equal order binomial heaps as in doing "addition", i.e., from smallest order to largest order, so that no two binomial trees after this operation have the same order.
- Takes $O(m)$ time, where m is the order of the largest tree in the heaps. Here m is $O(\log n)$, when total number of nodes is n.

- Find minimum: $O(\log n)$ time by going through the root of each binary tree. If pointer to minimum is kept track of, then it takes $O(1)$ time.

- Insertion: create a binomial tree of one node, and merge $O(\log n)$ time. Note that along with this operation, minimum pointer can be updated.

- Delete minimum: Find minimum, and make all its children binary-trees and do a merge. Takes $O(\log n)$ time. Pointer to minimum can be correspondingly updated.
- Decrease key: Bubble up until heap property is restored. Takes $O(\log n)$ time.
- Delete arbitrary element: Decrease the value to smallest possible ($-\infty$) and bubble up, and then do as in delete.

Fibonacci Heaps

- Similar to binomial heaps. However, the individual trees may not be exactly binomial trees (since in some cases, we cut parts of the tree to shift them to the root level). Also, we do not do merging for each insertion.
- This allows many of the operations to be done in "amortized" $O(1)$ time except mainly for deletion. This would give something like $O(m + n \log n)$ time when the number of deletions and the largest number of nodes in the system is at most n, and m is the total number of operations.

- Normally, unordered binomial tree U_k has k children which are $U_0, U_1, \ldots, U_{k-1}$, in some order.
- Each tree is like a heap: parent's key value is at most the children's key values.
- In some cases, some of the children might have been 'cut'.
- The roots of the unordered binomial trees are arranged in a circular linked list, with a pointer to the minimum.
- For each node, we keep track of its degree. Some of the nodes are marked.

Amortized Analysis

- Have “Potential” for the nodes/trees. To lend time to the future.
- Each root item has potential one.
- Each marked node as potential two.
- Total potential is the sum of the potentials.
- Intuitively, some of the operations may take lot of time, and others a lot less: Potential allows us to borrow time from cheap operations and pass it to expensive operations.
- This kind of analysis is called amortized analysis: each individual operation may be costly, but overall sum is not much.

- Potential of one denote constant work time (the maximum of the constants in the O notations that we encounter in the analysis).
- Union/Merge of two heaps: Merge the root-linked lists, and update the minimum. Takes $O(1)$ time. Note that the potential is the sum of the potentials of the two heaps.
- Insert: form a unordered tree of one node, and merge with the root list of original heap: Takes $O(1)$ time, and potential gets increased by 1 (constant).
- Finding min is easy and takes $O(1)$ time. So total cost is $O(1)$.

- Delete-Min: This is costly operation. Need to go through the roots to update the minimal pointer. Along with it, we will update the heap as follows:
- Delete the min, and put its children in the root-list.
- Go through each root, merging roots with same degrees.

- The delete operations take time proportional to the number of roots, plus number of children of minimal node.
- Potential is decreased by the number of earlier roots, but increased by the number of new roots.
- The number of new roots is at most $O(\log n)$ (to show: we will later show that any node with degree k has at least F_{k+2} nodes in the subtree rooted at it).
- The number of children of old min is at most $O(\log n)$ (to show: as above).
- Thus, total cost = (net work done) + (new potential) - (old potential) is $O(\log n)$.

- Decrease Key and deleting a node:
- Deleting is decreasing the key to $-\infty$ and then deleting min.
- So just consider decrease key.
- Decrease the value of the key. If not less than parent, then no change. Otherwise, 'cut', and mark the parent if not already marked. If already marked, cut the parent and follow this procedure again with the parent.
- 'cut': remove the node and put it in the root list (updating min if needed).
- Takes time constant for each cut. For cutting marked ancestor nodes, since we change the potential from 2 to 1 (by shifting to root), we are ok.

- Main thing is bounding the maximum degree:
- Suppose $degree(x) = k$ is the degree of a node, and let $z_1, z_2, \ldots, z_k$ be children of x in the order they were attached to x. Let $degree(z_1), degree(z_2), \ldots, degree(z_k)$ be their degrees. Then, $degree(z_1) \geq 0$, $degree(z_i) \geq i - 2$ (it started with degree being $i - 1$, and may have lost one child).

- Recall $F_0 = 0, F_1 = 1, F_{k+2} = F_{k+1} + F_k$.
- $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$.
- $F_{k+2} \geq (\frac{1+\sqrt{5}}{2})^k$.

- s_k is minimum possible size of tree with degree k.
- $s_0 = 1, s_1 = 2, s_2 = 3$
- $s_k \geq 1$ (for root) $+1$ (for z_1) $+ \sum_{i=2}^{k} s_{i-2}$
- note that s_{i-2} is the minimal value of degree of z_i.
- Thus, $s_k \geq 2 + \sum_{i=0}^{k-2} s_i$.
- Now, by induction $s_k \geq F_{k+2}$.
- Holds for $k = 0$. For induction, note that $s_k \geq 2 + \sum_{i=0}^{k-2} F_{i+2} = 1 + \sum_{i=0}^{k} F_i \geq F_{k+2}$ by earlier result.