Part I. Data Structures

- because every algorithm must store and access/search data

Assume you know:

- priority queue (heap)
- dictionary (hashing, balanced binary search trees)

In this course

- more advanced structures/analysis
  in particular amortized analysis

Priority Queue

store n elements, each with integer key

Operations:

- Insert, DeleteMin, DecreaseKey, Build
  (and Delete)
Heap
- binary tree of elements
- structure

```
  a
 /|
/  |
min heap b  c
  a ≤ b
  a ≤ c
```

Thus min key is at root

- shape

```
     a
    /
   /
  add & delete here
```

Almost perfect
So can store in an array in level order and use indexing instead of pointers
Height is $\Theta(\log n)$

Implementing priority queue operations on a heap

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>$\Theta(\log n)$</td>
</tr>
<tr>
<td>Delete Min</td>
<td>$\Theta(\log n)$</td>
</tr>
<tr>
<td>Decrease Key</td>
<td>$\Theta(\log n)$</td>
</tr>
<tr>
<td>Build Heap</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>Merge</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>Destroy 2 heaps</td>
<td></td>
</tr>
</tbody>
</table>

(Useful in some applications)

Build new one
Application

Prim's algorithm for **Minimum Spanning Tree** :
(recall that we used this last day for TSP)
Given an (undirected) graph with weights on edges, find
min. weight spanning tree

Prim's Algorithm (from CS 341)
Grow MST greedily starting from some node s
General step

```
          U
          ↓
          3
          ↓
          1
          ↓
          2
          ↓
          4
```

n=#vertices  m=#edges

**Using heap of edges from U to V-U**
alg. uses:  n Deletemin + m (Insert & Delete)
heap has = m elements
-time:  n O(log m) + m O(log m) = O(n+m) log m)

**Using heap of vertices in V-U** - key of vertex v = min weight of edge from U to v:
heap has = n elements
n Deletemin + m DecreaseKey
n O(log n) + m O(log n) = O((n+m) log n)
This is only better by constant since m is O(n^2) so log m is O(log n)
But we will see a way to improve DecreaseKey.
Binomial Heaps
- improve merge (not yet improvement for Prim)
- use pointers to implement trees, not an array
  This allows
  - keep heap order but go beyond binary trees
  Also relax shape and allow multiple trees.

Binomial Tree Definition

\[ B_0 = \quad B_1 = \quad B_2 = \]

\[ B_k = B_{k-1} + B_{k-1} + \ldots + B_2 + B_1 + B_0 \]

\[ k = \text{rank of } B_k = \text{degree of root} \]

**Claims**
- \( B_k \) has \( 2^k \) nodes \( (2^{k-1} + 2^{k-1} = 2^k) \)
- height \( (B_k) = k \) \( (k-1 + 1) \)
- there are \( \binom{k}{i} \) nodes at depth \( i \) in \( B_k \)
  \[ \binom{k-1}{i} + \binom{k-1}{i-1} = \binom{k}{i} \]

\[ \{ \text{all proved by induction} \} \]

hence the name
Binomial Heap Definition - a collection of binary trees, at most one of each rank, each one heap ordered.

E.g. for \( n = 13 \) use \( B_3 \quad B_2 \quad B_0 \)

\[
\begin{align*}
2^3 &= 8 & 2^2 &= 4 & 2^0 &= 1 \\
13 &= 1 \ 1 \ \_ \ 1 \text{ base 2}
\end{align*}
\]

Expressing \( n \) in binary tells us which trees to use:

\[
\# \text{trees} \leq \lfloor \log n \rfloor + 1 \quad \text{since this is \# binary bits in } n
\]

Height of each tree is \( \leq \lfloor \log n \rfloor \)

Merging two binomial heaps (all other operations will be implemented in terms of this):

Like binary addition:

for \( i = 0, 1, \ldots \) if there are two \( B_i \)'s, link to form a \( B_{i+1} \)
Example of Merging Binomial Heaps

\[ n_1 = 6 \]

\[ + n_2 = 3 \]

\[ \begin{array}{c}
\text{final binomial heap} \\
2 \\
\text{5}
\end{array} \]

Note: in case 1 0 1 1, two B_0's merge to a B_1. Then we have 3 B_1's. We merge two of them and leave one alone.
Analysis of Binomial Heap Operations (worst-case)

Merge $\Theta(\log n)$ time (joining $B_i$ and $B_j$ takes $\Theta(1)$, so the cost is exactly the same as the bit cost of binary addition).

Insert $\Theta(\log n)$ — merge binomial heap with singleton (so it’s like adding 1 to binary counter).

DeleteMin $\Theta(\log n)$
- Look through roots (at most $\Theta(\log n)$) to find the min.
- Yank it out of its tree, leaving $\Theta(\log n)$ binomial trees representing $2^i - 1$, i.e., $B_{i-1}, B_{i-2}, \ldots, B_0$
- Merge this with the rest of the binomial heap. $\Theta(\log n)$

Decrease Key $\Theta(\log n)$ — bubble up as in standard heap.

Creating a binomial heap of $n$ items
- Repeated insertion $O(n \log n)$ worst case
  But we’ll see that in fact it’s $\Theta(n)$.
  Idea: the worst case cost of insertion, $\Theta(\log n)$, does not happen too often.

Cost of repeated insertion = cost of incrementing binary counter