Dynamic connectivity lower bound:

[Patrascu & Demaine - SICOMP 2006]

inserting/deleting edges & connectivity queries require $\Omega(\log n)$ cell probes/op.
even if connected components are paths
even amortized (but here prove for worst case)

Proof:
- consider $\sqrt{n} \times \sqrt{n}$ grid with
  perfect matching between
  columns $i$ & $i+1$ for each $i$,
  forming permutation $\pi_i$
- block operations:
  - $\textbf{update}\ (i, \pi) : \pi_i \leftarrow \pi$
  = $O(\sqrt{n})$ edge deletions & insertions
  - $\textbf{verify-sum}\ (i, \pi) : \sum_{j=1}^{i} \pi_j = \pi$?
  = $O(\sqrt{n})$ connectivity queries
- Claim: $\sqrt{n}$ updates + $\sqrt{n}$ verify sums
  require $\Omega(\sqrt{n} \cdot \sqrt{n} \cdot \log n)$ cell probes
  $\Rightarrow \Omega(\log n)$/op.
Bad access sequence:
- for i in bit-reversal sequence:
  - verify-sum(i, \sum_{j=1}^{i} \pi_j) \Rightarrow \text{answer=yes (but DS must check)}
- update(i, \pi_{\text{random}})
  - uniform random permutation
- build tree over time:

- left & right subtrees of each node interleave

- Claim: for every node v in tree,
  say with l leaves in its subtree,
  during right subtree of v (time interval)
  must do \Omega(l \sqrt{n}) expected cell probes
  reading cells last written during left subtree

- sum lower bound over all nodes:
  - read r of write w only counted at lca(r, w)
  - linearity of expectation
\Rightarrow \Omega(n \lg n) lower bound total
(each leaf in \Theta(lg n) subtrees)
Proof of claim:

- left subtree has \( \frac{1}{2} \) updates with \( \frac{1}{2} \) rand. perms.
- any encoding of these permutations must use \( \Omega(l \sqrt{n \log n}) \) bits [information/Kolmogorov theory]
- if claim fails, find smaller encoding \( \Rightarrow \) contradict.
- setup: know the past (before \( v \)'s subtree)
- goal: encode (verified) sums in right subtree
  \( \Rightarrow \) can recover (updated) perms. in left subtree

\[
\begin{array}{cccccccc}
\emptyset & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\
\downarrow & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array}
\]

\( \pi_i = \pi_{i-1}^{-1} \circ \cdots \circ \pi_1^{-1} \circ \pi_j \circ \pi_{i+1}^{-1} \)

- farther left: known \( \Rightarrow \) not yet updated

Warmup: query is \( \sum(i) = \sum_{j=1}^{i} \pi_j \) \( \text{(partial sums)} \)
- let \( R = \{ \text{cells read during right subtree} \} \)
  \( W = \{ \text{cells written during left subtree} \} \)
- encode \( R \cap W \) (address & contents of each cell)
  \( \Rightarrow |R \cap W| \cdot O(\log n) \) bits [assume poly. space \( \Rightarrow w = O(\log n) \)]
- decoding alg. for sums in right subtree:
  - simulate sum queries in right subtree
  - to read cell written in right subtree: easy
    - in left subtree: \( R \cap W \)
    - in past: known

\( \Rightarrow |R \cap W| \cdot O(\log n) = \Omega(l \sqrt{n \log n}) \)

\( \Rightarrow |R \cap W| = \Omega(l \sqrt{n}) \)
Verify-sum instead of sum:
- permutations \( \pi \) given to verify-sum encode the information we want
  \( \Rightarrow \) no info LB
- setup:
  - know (fixed) past
  - don't know updates in left subtree
  - don't know queries in right subtree
  - but know that queries return YES
- decoding idea:
  - simulate all possible input permutations for each query in right subtree
  - know one returns YES, all others NO
- trouble: incorrect query simulation
  reads cells \( R' \neq R \)
  - if read \( r \in R' \setminus R \), it must be incorrect
  - but can't tell whether \( r \in W \setminus R \) or \( p \setminus (R \cup W) \)
  - can't afford to encode \( R \) or \( W \)
- idea: encode separator \( S \)
  for \( R \cap W \) & \( W \setminus R \)
- when decoding, to read cell written in right subtree: easy
  in \( R \cap W \): encoded explicitly
  in \( S \): must be in past \( \Rightarrow \) known
  not in \( S \): must not be in \( R \) \( \Rightarrow \) incorrect; ABORT
- only one simulation returns YES; rest NO or ABORT
  \( \Rightarrow \) recover desired permutation
  \( \Rightarrow \) encoding length \( \Omega (\sqrt{n} \log n) \)
**Separators:**
- Given universe \( U \) & number \( m \)
- **Separator family** \( \mathcal{S} \) for size-\( m \) sets if
  \[
  \forall A, B \subseteq U \text{ with } |A|, |B| \leq m \& A \cap B = \emptyset: \\
  \exists C \in \mathcal{S} \text{ such that } A \subseteq C \& B \subseteq U \setminus C
  \]
- **Claim:** \( \exists \) separator family \( \mathcal{S} \)
  with \( |S| \leq 2^0(m + \log_2 |U|) \)

**Proof Sketch:**
- Perfect hash family \( \mathcal{H} \) with \( |\mathcal{H}| \leq 2^0(m + \log_2 |U|) \)  
  \[\text{[Hagerup & Thorup - STACS 2001]}\]
- Gives mapping from \( A \& B \) to \( O(n) \)-size table
- Store \( A \text{-or-} B \) bit in each table entry
- \( 2^0(m) \) such vectors
  \( \Rightarrow 2^0(m) \cdot 2^0(m + \log_2 |U|) = 2^0(m + \log_2 |U|) \)

**Encoding:** \( R \cap W + \text{separator of } R \cap W \& W \setminus R \)
- **Size:** \( |R \cap W| \cdot O(\log n) + O(|R| + |W| + \log n) \)
  \[= \Omega(\log n \cdot \log n) \]
  \( \Rightarrow |R \cap W| = \Omega(\log n) \)
  \( \Rightarrow \text{Claim} \)
  or \( |R| + |W| = \Omega(\log n \cdot \log n) \)  \( \Rightarrow \Omega(\log n) \) for op.
Update-query trade-off: (possible by same technique)
\[ t_q \log \frac{t_u}{t_q} = \Omega(\lg n) \quad \& \quad t_u \log \frac{t_q}{t_u} = \Omega(\lg n) \]

- for \( t_u = \Omega(t_q) \), trees can match (small mods. to link-cut trees)
- for \( t_u = \Omega(\lg n (\lg \lg n)^3) \), can match [Thorup-STOC 2000]