B trees vs. Skip List

\[
\begin{align*}
& a, a_2, \ldots, a_k \\
& \leq a_i \\
& a_i - a_r \\
& a_r
\end{align*}
\]

K-ary search tree \((K = 8)\)

\[
\frac{\log n}{\log k} \times k
\]

\[
\log n \cdot \log k - \log n
\]

Most keys will be in secondary memory when the number of keys is very large.

Performance is determined by \(n\) accesses to secondary memory (slower by \(10^2 - 10^4\)).
External memory model

2 levels
  — primary 0 cost
  secondary 1 cost

Goal is to minimize memory cost

Skip lists can thought of as a k-ary search trie tree
where the expected value of k = 2

The probability that search in skip list exceeds \( c \cdot 2 \log n \) is
less than \( \frac{1}{n^2} \)

(Inverse polynomial bound)
\( x_i \in \{ x \mid y_i \in Y \} \)

Report the number of points in the range \([x_i, x_2] \times [y, y_2]\).

- The actual data: reporting version
- Counting: brute force

An obvious method is brute force

\( \Rightarrow O(n) \) time

It may be acceptable when output (answer) is large.

"Output-sensitive" query algorithm

The time to search will depend on the size of the output.

\[ O(f(n) + K) \]

Fixed cost usually included in counting version also.
One dimensional problem

\[ \ldots \left[ \ldots \ldots \ldots \right] \ldots \ldots \ldots \ldots \]

\( X_1 \quad X_2 \)

Variation of binary search:
\( O(\log n + k) \) query time

\( O(n) \) space

\( O(n \log n) \) preprocessing time

(Insertion/Deletion) Update cost

Two dimensions
For \( \binom{n}{2} \) distinct possible \( \mathbb{I} \) of \([x_1, x_2]\), we can build separate data structures in the \( Y \) direction.

Query cost: \( O(\log n + k) \)

Space: \( O(n^2) \cdot n \cdot O(n^3) \times \) unacceptable