external memory model (Aggarwal & Vitter ’88) - single processor with internal memory holding $M$ objects. Also unbounded external memory. All computation
on internal memory.
- can do individual input or output (I/Os) operations reading or writing blocks of B objects between memories.
- Focus purely on minimizing # I/Os.
- data structure size judged by # flocks in external memory
- input of N objects, output of K objects
Assume \( \mathbb{Z} \mathbb{B} \leq M = N \).
\[ m := \lfloor M/B \rfloor \]
\[ n := \lfloor N/B \rfloor \]
\[ k := \lfloor K/B \rfloor \]

(assume input in \( n \) contiguous blocks of ext. memory)
Problem: Is query object $x$ in a collection of $N$ objects?

**EXTERNALSCAN($x$):**
- for $i \leftarrow 1$ to $n$
  - read block $i$ of the file into memory ($\star$)
  - if the block contains $x$
    - return **TRUE**
  - return **FALSE**

Uses $O(n) = O(N/B)$

$\text{I/OS}$
If the input is sorted:

Binary search over whole blocks.

\[ O(\log n) = O(\log (N/B)) \]

I/Os.
B-tree:

Uses $O(ln) = O(\frac{n}{B})$ blocks.

Search, insert, delete in $O(\log_B n) = O(\log_B (\frac{n}{B}))$ I/Os.
\[ \Theta(n) = \Theta(N/b) \text{ leaves} \]

all objects stored in leaves (\(\Theta(B)\))

objects per leaf

leaves are sorted

root has between \(2^B\) children.

other internal nodes have between \(\frac{B}{2} + B\) children
Each node has some search keys and children. Each child has only objects with value between the \((i-1)st\) with keys.
Use keys to help search know which subtree contains & to recurse.

Each node reduces search space by a $O(B)$ factor so $O(\log_B n)$ I/Os to search.
Could build a B-tree with $N$ insertions.

$O(N \log\frac{n}{\log B})$ time $I/O$, total.

Faster if we sort first.
Sorting "bottom-up" internal merge sort.

Data partitioned into sorted runs. Initially each is size 1.

In each round, merge disjoint
consecutive pains of runs into one bigger run in linear time in run length.

$O(n)$ time per round reduce # runs by a factor 2 per round $\Rightarrow O(\log n)$ rounds $\Rightarrow O(n \log n)$ time
External Merge Sort:

1) Individually sort blocks of \( M \) input objects. That's \( m \) blocks per sort.

\( O(n) \) time

Now have \( n/m \) sorted runs of \( m \) blocks.
Each round does $m$-way merges, sorts collections of $m$ runs.

Read first block from each run.

Repeatedly "consume" smallest object from input blocks.
t place it in an output block.

Read next input block when previous block of run is empty.

Write output block when it is full.
\( m = 3 \)

\[ 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \]

\( 0(\text{I/Os per round}) \)

Reduce runs by factor \( m \) after each round.

Initially \( n/m \) runs
To build a B-tree:

1. Sort all elements
2. Build bottom up.
Adds only $O(n)$ I/Os.
Range searching in 1D:

B-tree.

To search in \([q_1, q_2]\),

Search for \(q_1\).

Search for \(q_2\).

Output (portions of) blocks between them.
\[ O(\log n + kB) = O\left(\log_2 \left(\frac{N}{B}\right) + \frac{k}{B}\right) \]
External \(kd\)-tree (2D):

- Root is first \(\log B\) levels of internal \(kd\)-tree.
- \(\Theta(B)\) children \(kd\)-trees for root block.
- Groups of \(\leq B\) points form the leaves.
Query same as internal tree. Only load blocks you need.

\[ T(n) \text{: } 3 \text{ I/Os to search n blocks.} \]

\[ T(n) \leq 3 + 2T(n/2) \]

\[ = O(\sqrt{n}) \]

\[ O(\sqrt{n} + k) = O(\sqrt{\frac{n}{b^2}} + k/b) \]
I/Os total can't do better with $O(n)$ size data structure.