# MCMD L4 : I/O-Efficient Searching with B-Trees

```
Disk <---I/O---> RAM <--> CPU
N = size of problem
B = block size
M = size of memory
T = size of output
I/O = block move between disk + memory
```

Sorting N items:
 Theta((N/B) log_{M/B} (N/B))  <<  N log_2 N

-------------------------

Internal Memory Searching

Binary Tree:

```
                  * (root)
         *                   *
    *         *         *         *
 *    *    *    *    *    *    *    *
□□ □□ □□ □□ □□ □□ □□ □□
```

 - all elements at leafs, height log_2 N.
 - search traces a (root)-(leaf) path
 -> Search : O(log_2 N) I/Os
 -> Range query : O(log_2 N + T) I/Os

External Trees:
 BFS blocking:

```
----------------------------------------
|                  * (root)            |
|         *                   *        |
|    *         *         *         *   |
----------------------------------------
| *  | *  | *  | *  | *  | *  | *  | *  |
|□□|□□|□□|□□|□□|□□|□□|□□|
-----------------------------------------
```

 - each block has height O(log_2 B),
                  width theta(B)
 - block height = O(log_2 N)/O(log_2 B) = O(log_B N)
 - output also blocked in sorted order
 - range query : O(log_B N + T/B) I/Os

Optimal: O(N/B) space O(log_B N + T/B) query

```
------------------------------------------
```
What about updates?  Stay balanced?  rotation?

Difficult to maintain block structure on rotation:

```
:                     :        ----------
|         (y)         |        |    (x) |
-----------------------  ->    |  [L]   | (y) --------
|   (x)    |    (z)   |        |       [R]|--| (z)   |
| [L]  [R] |  []  []  |        -----------|   [] []  |
-----------------------                   |__________|
```

- tough to make leaves blocked

```
-------------------------------------------
```

B Trees

Theta(B) - fan out

```
----------------------------------------
|                  * (root)           |
----------------------------------------
| *  | *  | *  | *  | *  | *  | *  | *  |
|[][]|[][]|[][]|[][]|[][]|[][]|[][]|[][]|
-----------------------------------------
```

- allow variable degree fan-out.  Split and merge nodes.

```
------------------------------------------
```
(a,b) Tree

- each node has between a and b fan-out (except root)
- all leaves on same level (balanced)
- root has degree in [2, b].

- O(N) space.  Height O(log_a N)
- Let a,b = Theta(B) ->  each leaf and node in one block
- O(N/B) blocks, O(log_B N + T/B) query

INSERT(x):
Search tree, insert x at leaf v
If v has b+1 elements/children
  Split v:
   - make nodes v' + v'' with (a,b) elements  {a <= b/2}

```
   - remove v from parent(v)
   - insert v' and v'' in parent(v)
  Check if parent(v) needs to be split (recursively up the tree)
Touches O(log_a N) nodes.


DELETE(x):
Search tree for x, delete x from leaf v
If v has a-1 elements/children
  Fuse v to sibling v'
   - move children of v' to v
   - delete v' from parent(v)
     (if parent(v) root with 1 child v, delete root)
   - If (v has >b) Split(v)
  Check if parent(v) needs to be fused with sibling, and recursively...
Touches O(log_a N) nodes.

Rebalancing:
 Let b > 2a  -->  update causes O(1/a) rebalancing ops (amortized)
  (hard to show)
 Let b = 4a
  Split:  leaf contains 4a/2 = 2a  (a far from a or b=4a)
  Fuse:  leaf contains (2a - 5a).  Split if >3a to 3/2 a  - 5/2 a
                (both at least a/2 far from a or b=4a)

----------------------
Summary:
 (a,b) tree w/ a,b = Theta(B)   (i.e. b = B-10, a = B/2 - 21)
 - O(N/B) blocks
 - O(log_B N + T/B) range query I/Os
 - O(log_B N) insert/delete

B-Tree with elements in leaves := B^+-Tree
Weight Balanced B-Tree has more spread out "rebalancing".


Construction in O((N/B) log_{M/B} N/B) I/Os
 - sort elements.  chunk to blocks as leaves.
 - build tree level-by-level bottom up


Does an (a,b) ever become unbalanced
 - all inserts to right?
 - all deletes from left?
(nope, only changes level at root)

Note uses sorting to build.  But cannot sort efficiently by inserting into a
```

tree, element-by-element or even block-by-block.