B-Trees
B-trees of order m have the following properties:

- The root has at least two subtrees unless it is a leaf.
- Each non-root internal node n holds \( k-1 \) elements and \( k \) children where \( \lceil m/2 \rceil \leq k \leq m \).
- Each leaf n holds \( k-1 \) elements where \( \lceil m/2 \rceil \leq k \leq m \).
- All leaves are on the same level.
FIGURE 12.14 A B-tree of order 6
• B-trees were created to make the most efficient use possible of the relationship between main memory and secondary storage

• For all of the collections we have studied thus far, our assumption has been that the entire collections exists in memory at once
• Consider the case where the collection is too large to exist in primary memory at one time

• Depending upon the collection, the overhead associated with reading and writing from files and/or swapping large segments of memory in and out could be devastating
• Disk access: 8 ms. (SSD: .08 ms)
• RAM access: 50 nano sec
• CPU speed: i7 3770K .25 nanosec / clock cycle, Decode 4 instructions per clock cycle!
  Execute 128 micro ops simultaneously!
  Retire up to 4 micro ops per clock cycle!
(1 instruction == 1 or more micro ops)
How many instructions executed in 8 msec? 180K
• B-trees were designed to flatten the tree structure and to allow for larger blocks of data that could then be tuned so that the size of a node is the same size as a block on secondary storage

• This reduces the number of nodes and/or blocks that must be accessed, thus improving performance
**Figure 19.83**

A 5-ary tree of 31 nodes has only three levels.
• A variation of B-trees called B*-trees were created to solve the problem that the B-tree could be half empty at any given time
• B*-trees have all of the same properties as B-trees except that, instead of each node having \( k \) children where \( \frac{m}{2} \leq k \leq m \), in a B*-tree, each node has \( k \) children where \( \frac{2m-1}{3} \leq k \leq m \)
• This means that each non-root node is at least two-thirds full
• Another potential problem for B-trees is sequential access
• B+-trees provide a solution to this problem by requiring that each element appear in a leaf regardless of whether it appears in an internal node
• By requiring this and then linking the leaves together, B+-trees provide very efficient sequential access while maintaining many of the benefits of a tree structure
FIGURE 12.15 A B+-tree of order 6
figure 19.84
A B-tree of order 5
**Figure 19.84**

A B-tree of order 5
**Figure 19.85**

The B-tree after insertion of 57 in the tree shown in Figure 19.84.
**figure 19.86**

Insertion of 55 in the B-tree shown in Figure 19.85 causes a split into two leaves.
**Figure 19.87**

Insertion of 40 in the B-tree shown in Figure 19.86 causes a split into two leaves and then a split of the parent node.