# CS-245 Database System Principles – Winter 2002
# Assignment 3

*Due at the beginning of class on Tuesday, February 5*

- State all assumptions and show all work.
- Subscribe to cs245@lists.stanford.edu to receive clarifications and changes.
- You can email questions to cs245-staff@cs.stanford.edu

---

**Problem 1. (50 points)**

Show the results of inserting the following keys, in the following order, into an initally empty B+-tree of order **n=5** (e.g., no more than 5 keys and 6 pointers per node):

- grape
- peach
- lemon
- banana
- apple
- apricot
- lime
- kiwi
- grapefruit
- pear
- cucumber
- cabbage
- carrot
- peas
- strawberry
- blueberry
- boysenberry
- cranberry
- watermelon
- canteloupe
- corn
- blackberry

**Do not draw the B+-tree after every key is inserted; instead, just draw the tree after each new B+-tree node is created.** Assume that when a block overflows it is split; keys are not re-distributed.

Next, show the results of deleting the following keys, in the following order, from the B+-tree you have created.

- lime
- cranberry
- lemon
- apple

Show the B+-tree that results from each deletion. Assume that when a block is underfull (according to B+-tree rules), it is coalesced with a block to the left or to the right; keys are not re-distributed.

**Problem 2. (15 points)**

Consider an index organized as a B+-tree. The leaf nodes together contain pointers to a total of $K$ records, and each block that makes up the index has $n$ pointers. We wish to choose the value of $n$ that will minimize search times on a particular disk device with the following characteristics:

- For the disk that will hold the index, the time to read a given block into memory can be approximated by $(70 + 0.05 * n)$ milliseconds. The 70 milliseconds represent the seek and latency components of the read, and the $(0.05 * n)$ milliseconds is the transfer time. That is, as $n$ becomes larger, the larger the block will be and the more time it will take to read it into memory.
- Once the block is in memory a binary search is used to find the correct pointer. The time to process a block in main memory is $(a + b * \log_2 n)$ milliseconds, for some constants $a$, $b$.
- The main memory time constant $a$ is much smaller than the disk seek and latency time of 70 milliseconds. (HINT: This means you can treat 70 + a as 70.)
- The index is full, so that the number of blocks that must be examined per search is $\log_n K$.

What value of $n$ minimizes the time to search in the B+-tree for a given record? An approximate answer is OK. The value you obtain should be independent of $b$. (HINT: If you come up with an equation which is hard to solve analytically, try solving it numerically or graphically.)

**Problem 3. (15 points)**

Imagine that we had a B+-tree with $n = 400$, that is, each node can have up to 400 keys and 401 pointers.

a. What is the maximum number of keys that a three level B+-tree with $n = 400$ could index? What is the minimum number of keys that a three level B+-tree with $n = 400$ could index, assuming no nodes were under-filled (according to the B+-tree rules)? By three level, we mean one leaf B+-tree level and two non-leaf levels.

b. After many insertions, a B+-tree tends to reach a steady state where each node is about 70% full; in other words, each node has $0.7n$ keys in it, on average. Under the assumption that every node (including the root) is about 70% full, how many layers would an $n$=400 B+-tree be if it indexed 6 billion ($6 \times 10^9$) keys?

c. For the B+-tree of part (b), with $n$=400, 70% full nodes and indexing 6 billion keys, what percentage of the total space consumed by the index is used by leaf-level blocks?

**Problem 4. (20 points)**

Consider the following key values:

- 106
- 115
- 916
- 5
- 98
- 126
- 23
- 15
- 31

These keys are to be inserted in the above order into an (initially empty) hash table. The hash function *h(n)* for key *n* is *h(n) = n mod 256*; that is, the hash function is the remainder after the key value is divided by 256 ($2^8$). Thus, the hash value is an 8-bit value. Each block can hold 3 data items.

a. Draw the hash table, **after all data items are inserted**, if it were an *extensible* hash table. Show the keys themselves in the buckets, not the hash value. The bucket numbers are drawn from the bits at the high order end of the hash value. Be sure to indicate *i*, the number of bits in the hash value that are used. Also indicate, for each bucket, the number of hash function bits that are used for that bucket.

b. Draw the hash table, **after all data items are inserted**, if it were a *linear* hash table. Show the keys themselves in the buckets, not the hash value. The bucket numbers are drawn from the bits at the low order end of the data value. Be sure to indicate *i*, the number of bits of the hash value that are being used, *n*, the number of buckets, and *r*, the number of records in the hash table. Also indicate the bucket numbers. Enforce the constraint that $r \leq 2.25n$.