Section 06: B-Trees and Heaps

1. B-Trees

(a) Insert the following into an empty B-Tree with M = 3 and L = 3: 12, 24, 36, 17, 18, 5, 22, 20.

(b) Given the following parameters for a B-Tree with $M = 11$ and $L = 8$:

- Key Size = 10 bytes
- Pointer Size = 2 bytes
- Data Size = 16 bytes per record (includes the key)

Assuming that M and L were chosen appropriately, what is the likely page size on the machine where this implementation will be deployed? Give a numeric answer based on two equations using the parameter values above. **Hint**: Some equations you might need to use are:

$$M = \lfloor \frac{p+k}{t+k} \rfloor$$
$$L = \lfloor \frac{p-t}{k+v} \rfloor$$

where p is the page size in bytes, k is key size in bytes, t is pointer size in bytes, and v is value size in bytes. **Hint**: Think about where these values come from.

(c) Give an example of a situation that would be a good job for a B-tree. Furthermore, are there any constraints on the data that B-trees can store?

(d) Find a tight upper bound on the worst case runtime of these operations on a B-tree. Your answers should be in terms of L, M, and n.

(i) Insert a key-value pair

(ii) Look up the value of a key

(iii) Delete a key-value pair

(e) Insert and then delete the following from a B-tree:

(i) Insert the following into an empty B-Tree with $M = 3$ and $L = 3$: 3, 18, 14, 30, 32, 36, 15, 16, 12, 40, 45, 38.

(ii) Delete 45, 14, 15, 36, 32, 18, 38, 40, 12 from the tree in the previous part.

2. Memory

(a) What are the two types of memory locality?

(b) Does this more benefit arrays or linked lists?

(c) What about Java makes it a poor choice for implementing B-trees?

3. Memory and B-Trees

(a) Based on your understanding of how computers access and store memory, why might it be faster to access all the elements of an array-based queue than to access all the elements of a linked-list-based queue?

(b) Why might `f2` be faster than `f1`?

```java
public void f1(String[] strings) {
    for (int i=0; i < strings.length; i++) {
        strings[i] = strings[i].trim();        // omits trailing/leading whitespace
    }
    for (int i=0; i < strings.length; i++) {
        strings[i] = strings[i].toUpperCase();
    }
}

public void f2(String[] strings) {
    for (int i=0; i < strings.length; i++) {
        strings[i] = strings[i].trim(); // omits trailing/leading whitespace
        strings[i] = strings[i].toUppercase();
    }
}
```

(c) Let k be the size of a key, t be the size of a pointer, and v be the size of a value.

Write an expression (using these variables as well as M and L) representing the size of an internal node and the size of a leaf node.

(d) Suppose you are trying to implement a B-tree on a computer where the page size (aka the block size) is $p = 130$ bytes.

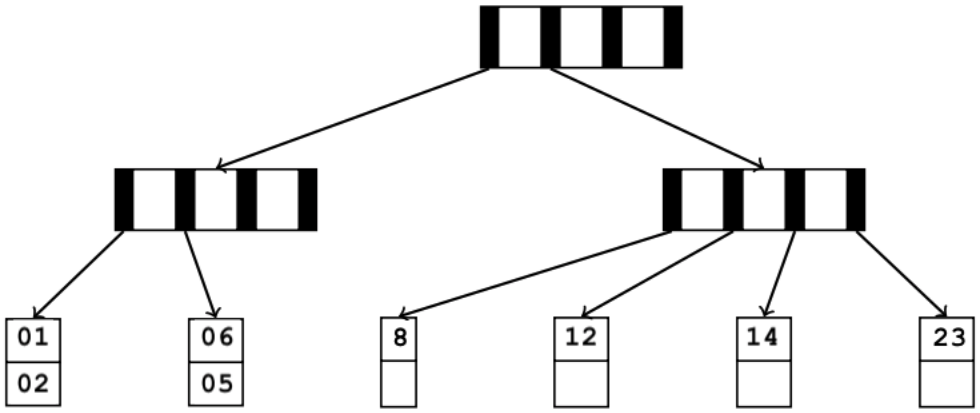
You know the following facts:

- Key size $k = 10$ bytes
- Value size $v = 6$ bytes
- Pointer size $t = 2$ bytes

What values of M and L should you pick to make sure that your internal and external nodes (a) fit within a single page and (b) uses as much of that page as possible.

Be sure to show your work. The equations you derived in the previous part will come in handy here.

(e) Consider the following "B-Tree":



(i) What are M and L?

(ii) Is there anything wrong with the above B-Tree? If so, what is wrong?

(f) Consider the following code:

```
public static int sum(IList<Integer> list) {
    int output = 0;
    for (int i = 0; i < 128; i++) {
        // Reminder: foreach loops in Java use the iterator behind-the-scenes
        for (int item : list) {
            output += item;
        }
    }
    return output;
}
```

You try running this method twice: the first time, you pass in an array list, and the second time you pass in a linked list. Both lists are of the same length and contain the exact same values.

You discover that calling `sum` on the array list is consistently 4 to 5 times faster then calling it on the linked list. Why do you suppose that is?

(g) Suppose you are trying to use a B-Tree somebody else wrote for your system. You know the following facts:

- $M = 10$ and $L = 12$
- The size of each pointer is 16 bytes
- The size of each key is 14 bytes
- The size of each value is 11 bytes

Assuming M and L were chosen wisely, what is most likely the page size on this system?

4. Memory and B-Trees: A Sequel

(a) Suppose you are writing a program that iterates over an `AvlTreeDictionary` – a dictionary based on an AVL tree. Out of curiosity, you try replacing it with a `SortedArrayDictionary`. You expect this to make no difference since iterating over either dictionary using their iterator takes worst-case $\Theta(n)$ time.

To your surprise, iterating over `SortedArrayDictionary` is consistently almost 10 times faster!

Based on your understanding of how computers organize and access memory, why do you suppose that is? Be sure to be descriptive.

(b) Excited by your success, you next try comparing the performance of the `get(...)` method. You expected to see the same speedup, but to your surprise, both dictionaries' `get(...)` methods seem to consistently perform about the same.

Based on your understanding of how computers organize and access memory, why do you suppose that is?

(Note: assume that the `SortedArrayDictionary`'s `get(...)` method is implemented using binary search.)

(c) You want to implement a B-Tree for a computer that has a page or block size of $p = 256$ bytes. Your pointers are $t = 4$ bytes long, your keys are $k = 2$ byte long, and your values are $v = 8$ bytes long. What should you select for M and L in order to maximize the performance of your B-tree? Please show your work.

Reminder: M and L must selected such that the following two inequalities remain true:

$$Mt + (M-1)k \leq p \qquad \text{and} \qquad L(k+v) \leq p$$

5. Heaps

(a) Insert the following sequence of numbers into a *min heap*:

[10, 7, 15, 17, 12, 20, 6, 32]

(b) Now, insert the same values into a *max heap*.

(c) Now, insert the same values into a *min heap*, but use Floyd's `buildHeap` algorithm.

(d) Insert 1, 0, 1, 1, 0 into a *min heap*.

6. Heaps: A Sequel

(a) Insert the following numbers into a max heap in the order provided: 10, 15, 5, 2, 20, 30, 4, 1, 41

DO NOT use `buildHeap`; just insert them one by one. Showing your work might be prudent on a real exam (for partial credit).

(b) Re-order the numbers so that if you were to insert them into a heap again, (1) the result would be an identical heap, and (2) the number of comparisons is minimal.

(c) Draw out the array version of the heap from part (a).