CSE 332: Practice Problems

1 Short Answer

a) See bigO handout on Piazza

b) Suppose a heap is full and has n elements. What is the bigO asymptotic runtime of an insert? Explain your answer

```
If the heap is full, then there must be a resize.  Because it takes n time
to copy the array, the resize operation is in O(n)
```

c) Write pseudocode, or explain briefly in paragraph form, an algorithm that will find the largest element in a BST. Include the worst-case runtime of this algorithm.

```
To find the largest element in a binary search tree, we instantialize a cursor
at the root.  We move the cursor down the right branch of each subtree until
we reach a node that has no right child.  The worst case for this situation
is the degenerate tree that has no left children.  It is then a linked list
with height equal to n, so this is an O(n) operation.
```

d) What is the best-case and worst-case bigO complexities for `find(key k)` in a half-full hash table implemented using linear probing. Briefly explain these best and worst case scenarios.

```
Best case is when every other element in the array is occupied, and then
the linear probe will have to try at most 2 locations, giving us a O(1) runtime.
The worst case is when half of the table is a contiguous cluster.  Then,
attempting to insert at the beginning of this cluster results in n/2 probes,
which is O(n) runtime.
```

e) Provide and explain the two types of locality relevant to caching and memory accesses.

Spacial locality: objects near each other in memory are likely to be accessed at the same time. This is accounted for by the memory page. If a piece of memory is accessed, the whole page is brought into cache, making nearby accesses faster.
Temporal locality: objects recently accessed are likely to be accessed again. This is accounted for by the cache. If we access a piece of memory, we bring it closer to the processor (in faster cache) so that we can access the memory faster.

f) When does the worst-case for B-tree insert occur?

The worst case for the B-tree insert is when the insertion forces the leaf to split, which then forces all of the signposts to split back up to the root. This situation can occur when the B-tree is close to the maximum number of nodes for its height.

g) Given an object consisting of three integers, decide whether summing the three together makes a good hash function. Explain your answer.

If the data is bounded (that is, it has a maximum and a minimum value), summing three numbers together does not evenly distribute the data. The summation will more likely result in a value that is nearer to the middle of the min and max than it does values at the edges

h) Explain the difference between an ADT and a data structure. Use examples.

An ADT describes functions and expected behaviors. For example this may be a priority queue, we support insert() and deleteMin() and the behavior that we expect is that deleteMin() will return the value with smallest priority. A data structure is an approach for actually structuring the data in memory in a way that can provide the desired behavior. In our priority queue example, the heap is a data structure that we can use. Data structures can be analyzed for their runtime and memory usage because they actually describe an approach to the problem. ADTs cannot be analyzed because they only describe the behavior

itself. Many data structures exist to implement an ADT.

i) Explain the worst case for sorted array insert? Is this different than the amortized insert?

Worst case for sorted array insert is when we need to add to the front of the array and all n elements need to be shifted. Because this can occur at every call of insert, it does not change given an amortized analysis.

j) Given the FixedSizeFIFOWorklist from P1, which parts of this interface specify an ADT, and which specify a data structure

Adding peek(i) to the WorkList interface is describing a new expected behavior of our object. However, if we insist that this means it is stored in an array, then it is a data structure decision. Behavior is the condition for an ADT and data structures deal with how the data is actually stored.

k) A client wishes to build and maintain a library of customer information. Discuss between B-tree, Trie, Hash table and AVL tree about what might be best. On what other variables does this choice depend?

If the "customer information" has values that cannot be represented in serial characters, then it cannot be represented as a trie. Choosing a B-tree would depend on the size of the data. If the data set is large enough that disk accesses are likely, then a B-tree might be indicated. If disk accesses are unlikely, our choice is between HashTable and AVL. Tradeoffs here are then between memory efficiency and runtime. Hash tables tend to use more memory (and they need to perform costly resizes) whereas AVL trees have slightly slower finds

l) For B-trees, what are M and L and what factors impact how we select them?

M is the maximum number of subtrees for a signpost node and L is the maximum number of key, value pairs in a leaf node. We select M and L such that signposts

and leaves are as close to one memory page in size without going over. This depends on the size of a page, the size of our keys and values and the size of a pointer.

2 Big O notation

For the following functions, determine the tightest bigO upper bound in terms of n. Write your answers on the line provided.

a)
```
int f1(int n){
      for(int i = n; i>0; i--){
           for(int j = 0; j<i; j++){
                System.out.println("!");
           }
      }
}
```

O(n^2)

b)
```
void f2(int n) {
      for(int i=0; i < n; i++) {
                for(int k=0; k < n; k++) {
                     for(int m=0; m < 10; m++) {
                          System.out.println("!");
      }    }   }    }
}
```

O(n^2)

c)
```
int f3(int n){
      if (n < 10) return n;
      else if(n < 1000) return f3(n-2);
      else return f3(n/4)*f3(n/4);
}
```

O(n)

3 Design Decision

A client is trying to provide a data structure which logs the locations of process jobs on a small server farm. There are 64 servers at the farm and there is no limit to the number of jobs that should be assigned to each server. For each process job, there are four pieces of information which need to be tracked:

- An int ID number which is unique to each job
- An int which indicates to which server the job has been assigned
- A long which is the time that the process was assigned to the server
- A String which identifies the owner of the job

Because this is a log, the client will only delete records when they were created in error. Because of this, fast delete times are not important. Additionally, the client will be calling find much more frequently than insert, so any speed benefits should prioritize speeding up find, if possible.

Provide a data structure and implementation that would meet the clients demands. Explain what data will be stored where and how it will be accessed. Then, justify any decisions you made using material from the course. This includes, but is not limited to: asymptotic runtimes, memory usage, experimental results and data structure properties.

This problem is solved with the Dictionary ADT where the unique job ID is the key and a combination of the other three pieces of data is our value. Once we have identified this, we have a choice for our data structure.

B-tree: If disk accesses are common (as usual). But, it's difficult to implement and doesn't provide much benefit if there isn't much data. We would get log(n) runtimes for all of our functions which is good and good memory utilization because so much of our data is stored in arrays.

AVL: Good overall data structure choice if all jobs can fit in memory. Log(n) finds and inserts. Rare deletions also make AVL desirable. If the data set is large enough that it cannot be stored easily in cache then it might be preferable over a hashtable. Additionally, a hashtable may take more memory than an AVL tree if it is well-maintained.

Hashtable:
A hashtable would be best if the total amount of data is small. We can get quick access times if all of the data is in memory or in the cache, provided the hashtable is well maintained. Memory constraints may make this difficult. Under those constraints, a chaining hashtable is a strong choice, but it could be undesireable if we are frequently resizing.

Other:
A linked list has no advantages over an AVL tree, so it there is no reason to store the data this way. An array might be desirable if the size of the data is very small, but if this is the case, any data structure will be suffficient.