

Due: Wednesday, October 21, 2:30 PM EST

This assignment is worth 12.5% of the final grade. Each question is worth a multiple of 5 points. If you have no idea how to answer a question, you will receive 20% of the credit for that question by saying “I do not know how to answer this question”. Leaving the question blank, you will receive 10% of the credit. If your answer makes no sense, you will not receive any credit. Any answer that shows some understanding of the question will receive more 20% credit.

1. Consider a perfectly balanced binary search tree T with $n = 2^4 - 1 = 15$ nodes. We give a name for the location of the nodes in T as follows: Let the root be called node 1. The nodes at the next level are nodes 2,3 going left to right. More generally the nodes in the j^{th} level are $2^{j-1}, 2^{j-1} + 1, \dots, 2^j - 1$ going left to right. In terms of these tree location names, where are the following keys located? **Note:** You should write out a balanced binary search tree to help answer the question. You can, but do not have to, submit the tree you wrote out.

- (5 points) The median key in T .

Solution: The median is at location 1.

- (5 points) The third largest key in T .

Solution: The third largest key in T is at location 3.

2. (10 points) Indicate how you would organize a linked list so that the largest keys could be access in constant time. Indicate how you would possibly rearrange the list when a new key is being inserted.

Solution:

Note: It is best to illustrate this on the board.

As mentioned in class we need a link to the header of the linked list to access the list. We can insure that the link to the header is to the highest key. Suppose the largest key in the current list is k_ℓ (located at the head of the linked list) and the head is pointed to by a link L . We remember L by setting $L' = L$. When a new key k arrives, we compare k with k_ℓ . If k is larger, then we place the new key at some location and point to it by L and then the location with the previous largest key is pointed to by L' . If k is not larger than k_ℓ , we can insert k to be located by a link from k_ℓ .

3. (10 points) Consider the following question related to hashing and hash tables. Let $h(ID) = ID \pmod{83}$. By “mod 83”, I mean divide by 83 and take the remainder. For example, for the ID = 1021518, we would get $1021518 \pmod{83} = 37$. That is, this ID is being hashed or mapped onto a much smaller number (i.e. a number between 0 and 82).

Suppose the above hash function was a perfect hash function (which it is not) in the sense that it is equivalent to a balls and bins experiment where every ball lands in a bin chosen uniformly at random. Suppose there were only 4 students in our class and all student IDs were integers x with $0 \leq x \leq 82$.

Calculate the probability that (at least) two students will be hashed to the same hash value. Hint: First calculate the probability that all 4 students get a unique hash value.

Solution: For a perfect hash function, the probability p_{yes} that there will be a collision when 4 students are being mapped onto 83 locations is equal to $1 - p_{no}$ where $p_{no} = \prod_{i=1}^3 \frac{83-i}{83} \approx .0707$.

- (10 points)

With regard to implementing a dynamic dictionary, compare using a hash table vs using a balanced binary search tree. Consider both “average” time and worst case time to perform an operation (i.e., search for and modify a record, add a record, delete a record). Indicate how you would deal with collisions.

Solution: The average case and worst case for all dictionary operations is $\Theta(\log n)$ where n is the number of keys in the tree. For the average case, all operations using a hash table take constant time. However, for the worst case time, all operations using a hash table can take linear time $\Omega(n)$ since all keys may have been hashed to the same location. Conclusion: for average case time, hash table are faster. For worst case time, balanced binary search trees are faster.

- (10 points) In addition to the dictionary operations, name another operation that one might want to use for a set of records. How would you implement this operation in a hash table and in a binary search tree?

Note: This is partially a “thought” question. You should have an application in mind (e.g., a data structure for employees in a company). You can add one or a few additional links to the data structure to facilitate how you would implement the new operation.

Later, we may explore *data bases* which go beyond the data structures we so far have encountered.

Solution: The median will be at the root (or maybe at one of the children of the root). There is no obvious way to maintain a hash table that would allow us to easily access the median valued key.

Note: It will be interesting if other operations are suggested and if so, what is your reason.. I can imagine someone saying that the linked list is a sorted linked list which would then facilitate some operations but still would take $\Omega(n)$ for the median.