

Quiz #2 Answers

1. Compare and contrast a hash table that uses linking to one with open addressing.

On the comparison side, obviously both of these are hash tables that use some hash function to map a key into a smaller value that can be used to direct access a bin in a vector/array. From there on out things are different. With linking, the bins are the heads of linked lists and new elements are added to the lists while searching requires walking the lists. The ratio of number of objects over number of hash cells can be greater than one, but this has a tendency to degrade performance.

With open addressing the hash bins store the data itself. If you try to store something in a place that is already taken you have to pick a different spot. We discussed doing this with linear probing, quadratic probing, and double hashing. The ratio of number of items to number of bins can never be greater than one and because performance drops when it gets close to one most hashes that use open addressing will force a value below that. When this value is exceeded the hash is grown and all the elements must be placed in it again. The greatest limitation of this method is that removing can lead to very poor performance if we don't do complete rehashing regularly.

2. Given the following code, what must you put into MyList and MyObject to get this code to compile and run properly?

```
template<class DataType>
void swap(DataType &d1,DataType &d2) {
    DataType tmp=d1;
    d1=d2;
    d2=tmp;
}
template<class Container>
void sort(Container &cont) {
    for(int j=0; j<cont.size(); ++j) {
        for(int k=0; k<cont.size()-j-1; k++) {
            if(cont[k]>cont[k+1]) swap(cont[k],cont[k+1]);
        }
    }
}
void foo() {
    MyList<MyObject> list;
    // fill the list
    sort(list);
}
```

Starting with the sort we see that we are calling a size method on the container and we are using operator[] to get individual elements. For the objects in the list we are doing comparisons with operator> and in the swap we are doing assignments so we have to make sure that if there are pointers we have overloaded operator=.

Extra Credit: Describe how you find the successor of a node in a binary search tree. (Write on back for more space.)

If there is a right child it is the smallest element on the right subtree. If there isn't a right child it is the nearest ancestor that the current node was to the left of.