# COMP3506 – Assignment 1

Kenton Lam

Due 09/08/2019 6:00 pm

## Question 2

### Memory Complexity

We assume that each grid element takes constant space. Then, in this implementation, the memory used is of the order $O(wh)$ where $w$ and $h$ are the width and height of the grid, respectively. Note that in particular, if $T$ is String, this is not the case as each string can take an arbitrary amount of space. A

This is calculated by considering the private fields of the implemented ArrayGrid class. The *width* and *height* members take constant space, $k$. The *arrayData* member takes space directly proportional to the area of the grid, $w \times h$. Together, it takes $O(wh + k)$ space. However, because the order of $wh$ greater than the constant width and height integers, we discard the constant $k$ to get $O(wh)$.

### Memory Efficiency

The grid allocates space for every element even if the element has not been added yet. This is not a problem if you expect that most cells would be filled. However, if the grid is very large but mostly empty (i.e. it is sparse), most of this space would remain empty (holding null values).

### An Alternative

One possible alternative is a hash map implementation, mapping keys as ordered pairs $(x, y)$ to their values [1]. This would use less memory by only allocating space as elements are added. In the case of a sparse grid, for each element, a hash-table stores its hash and the element itself. As a result, the memory complexity is $O(n)$ where $n$ is the number of elements added.

### Comparison

Memory complexity has been discussed above. For time complexity of each method,

- **add(int x, int y, T element)** – both implementations have average $O(1)$ time inserts, due to the properties of a hash table and array. However, the hash map can take more time if the hash table needs to be recomputed or resized. The array grid never needs to be resized because everything has been preallocated.
- **get(int x, int y)** – both implementations are $O(1)$ time, due to the constant time indexing in both cases.
- **remove(int x, int y)** – again, both constant time.

- **clear()** – array grid would take $O(wh)$ time because it needs to iterate over every grid position. The hash map would take $O(n)$ time because it knows which positions have hashes stored and only needs to delete those.
- **resize(int newWidth, int newHeight)** – resize is actually unnecessary for a hash map; if implemented, this would take $O(1)$ time. However for array grid, this is $O(wh)$ (where $w$ and $h$ are the old width/height) because it must check every cell for elements and copy to the new array.

## References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms.* Mit Press, 3 ed., 2009.