Grid World: Part I

1.
"Constant time" means that the program or operation will execute in a certain amount of time or memory space that's independent of the input size. As illustrated here with the pseudocodes:

```
int n = someValue;    //declaring a variable n of some value
doSomething
doSomethingElse
```

This has two statements that are independent of what the variable n is.

```
min = infinity;
x = 0;

for i=1 to n
        if A[ i ] < min
                then min = A[ i ];
                     x++;
```

This pseudocode tries to find the minimum value in an unordered of an array. It is an example of a non-constant time operation since it would require scanning each element in the array.

2.
GW gw_build(int nrows, int ncols, int pop, int rnd);

This function creates a 2D world by individually creating columns and then storing them into arrays. Based on the return statement, this function also serves as a linked list which we use to make our columns.

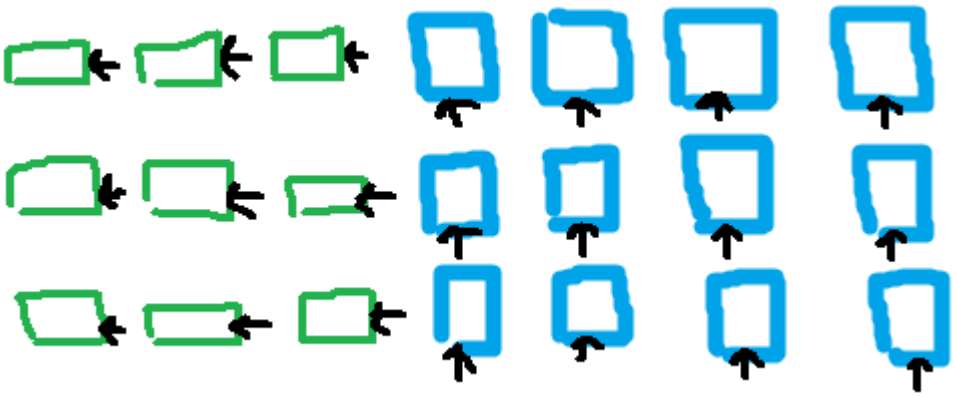
We perform this linked list operation as many times as there are nodes, which is one column at a time. We also do this one nrows at a time.

To do this, we perform the following steps for the linked list:

1. Allocate memory for the new node
2. Determine where we place the new nodes (which is right after the previously created node)
3. Point the new node to the successor
4. Point the predecessor to the new node

The function sets a random flag of the number of people (specified by int pop) that are placed into random districts. These districts are made by the linked list used to produce the nrows and ncols. When the random flag isn't set, then the people are put into node (0,0).

The function runs O(CR+N) times because the columns are created ncols times and then they're stored into an array.



world[0……i]

int * gw_members(GW g, int i, int j, int *n);

For this function, we're creating int arrays which will contain the specific number of the district based on the location (i,j) respectively.
As each district is created, we also take account the members living in each district. We do this by incrementing the values for each district and then saving to int *n.

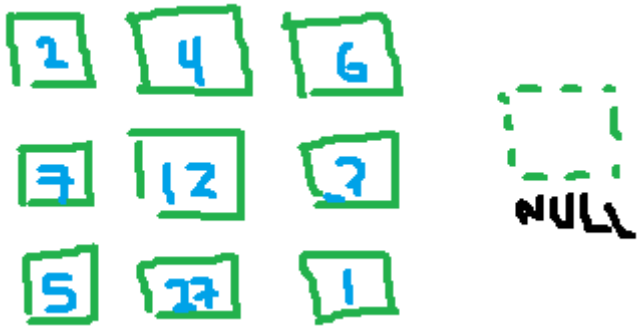
The runtime for this is $O(N_{ij})$ because we are saving values into each int arrays.

Each square is represented as districts –defined as (i,j)- and the colored dots are the different members living in each district.

int gw_district_pop(GW g, int i, int j);

Since this function prints the values of people living within the district, we are getting information from the list [namely the districts (i,j)] and then finding how many members live in each.

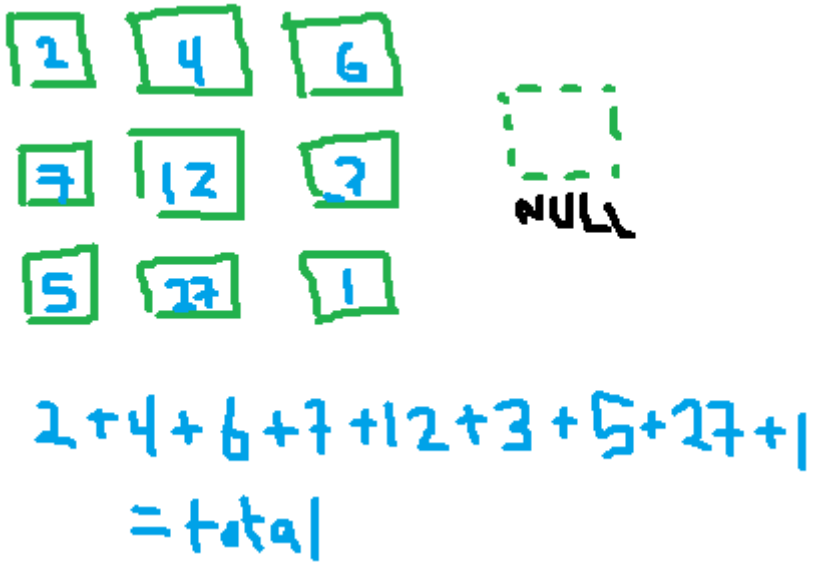
The runtime is O(1) because we're performing statements of independent length. Meaning that we can perform as many statements as long as they don't involve calling int i and j.



int gw_total_pop(GW g);

This function counts the entire population of all combined districts. We take the list and count each district (i,j) by incrementing the value of the total population.

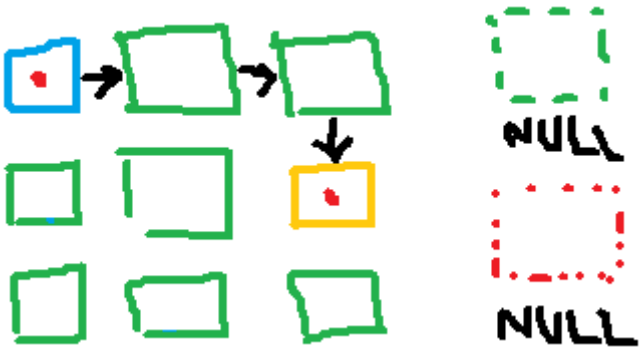
The runtime is O(1) since we're simply incrementing a value.



int gw_move(GW g, int x, int i, int j);

This function takes the person and moves them to a different district which we do by finding the member ID in district (i,j). To determine the district, we either traverse forward or backwards on the list until we find the specific district to place our member in.

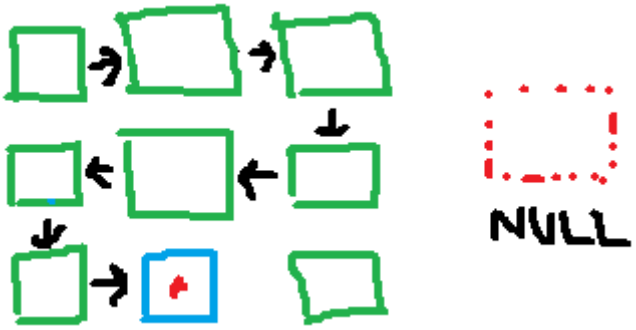
The runtime is O(1) since we're confirming whether or not such a member ID exists and that we traverse through each element until we find the desired district.



int gw_find(GW g, int x, int *r, int *c);

This function traverses through the list and looks at district (i,j) then at a member ID. If the district (i,j) doesn't contain the member ID, then it moves on to the next one until the member is found.

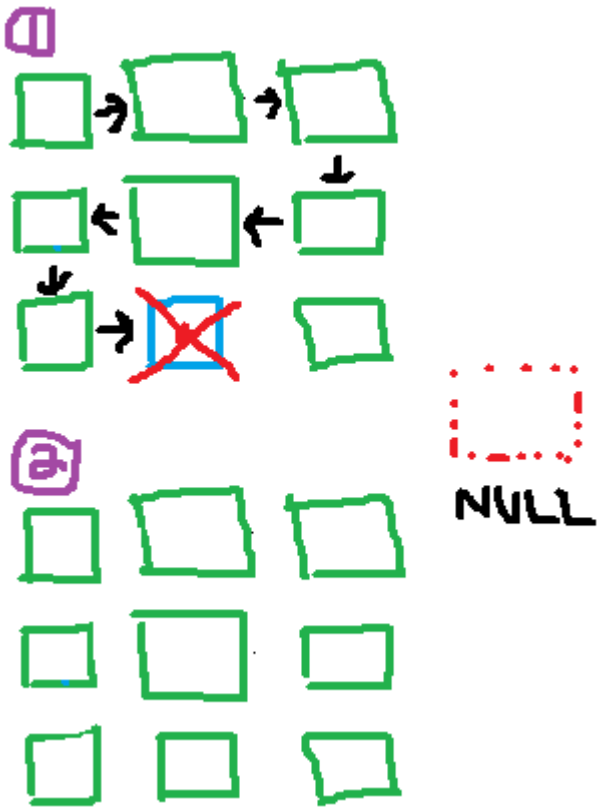
The runtime is O(1) since we're traversing through the list and finding a member ID.



int gw_kill(GW g, int x);

This function traverses through the list and searches for a specific member ID. When it finds the member, it deletes the node from the district (i,j).

The runtime is O(1) because we are traversing through the link and searching for an element.



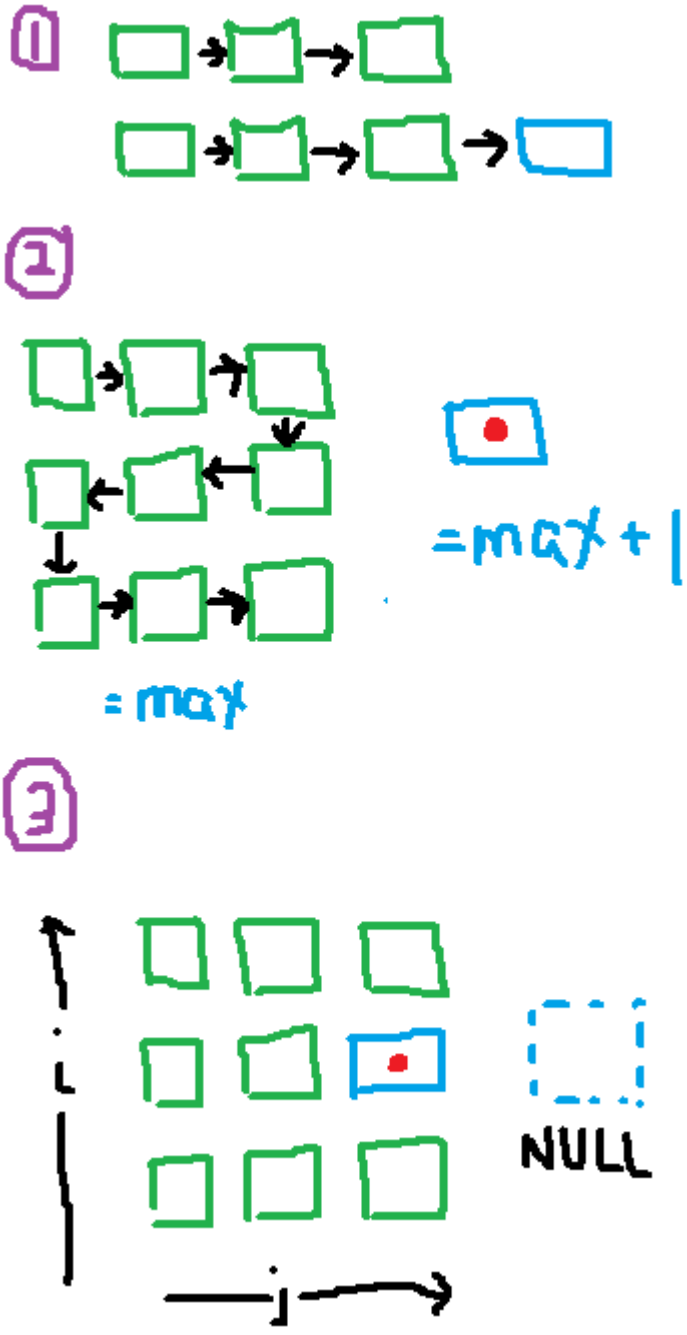
int gw_create(GW g, int i, int j);

This function creates a new node which is then popped into a stack.

When the function assign the user an ID, it finds if a member has previously died and assigns the value to the new member. Otherwise, it gives the member an incremented ID (ie if the member with the 'highest' ID is 9, then the new member is given the unique ID of 10).

When the user gives a value for i and j, the function will find out if the values are valid in the list. If they are, then the new member is assigned to the district, otherwise it returns false.

The runtime is O(1) because while some elements trigger memory reallocation, the time to insert the element is still constant on average.



void gw_free(GW g);

This function frees up the entire stack. This is achieved by going traversing through the list and freeing each node.