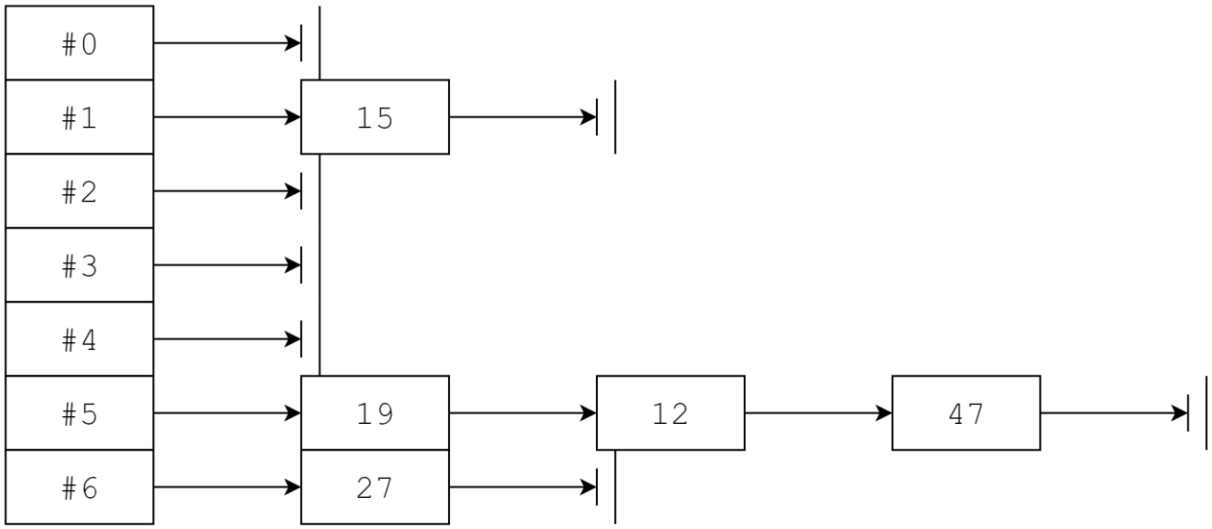
Computer Science 2210A: Assignment 3

1. Denote the hash table where the integers are stored by T[]. Hence every given integer is stored at $T[h(k)]$, where $h(k)$ is the position of integer k on the hash table. Then:

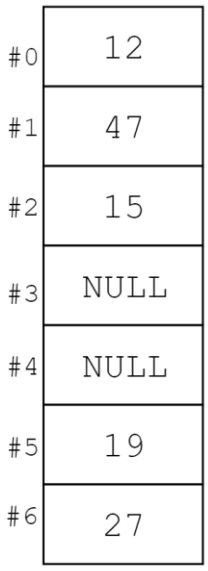
$$h(19) = 19\ mod\ 7\ = 7 \cdot 2 + 5\ (mod\ 7) = 5 \Rightarrow T[h(19)] = T[5]$$
$$h(27) = 27\ mod\ 7 = 7 \cdot 3 + 6\ (mod\ 7) = 6 \Rightarrow T[h(27)] = T[6]$$
$$h(12) =\ 12\ mod\ 7 = 7 \cdot 1 + 5\ (mod\ 7) = 5 \Rightarrow T[h(12)] = T[5]$$
$$h(47) = 47\ mod\ 7 = 7 \cdot 6 + 5\ (mod\ 7) = 6 \Rightarrow T[(47)] = T[5]$$
$$h(15) = 15\ mod\ 7 = 7 \cdot 2 + 1\ (mod\ 7) = 1 \Rightarrow T[(15)] = T[1]$$

Here is a diagram showing the current state of the hash table.



Note the table only contains pointers to linked lists. The indices are merely for the reader's reference.

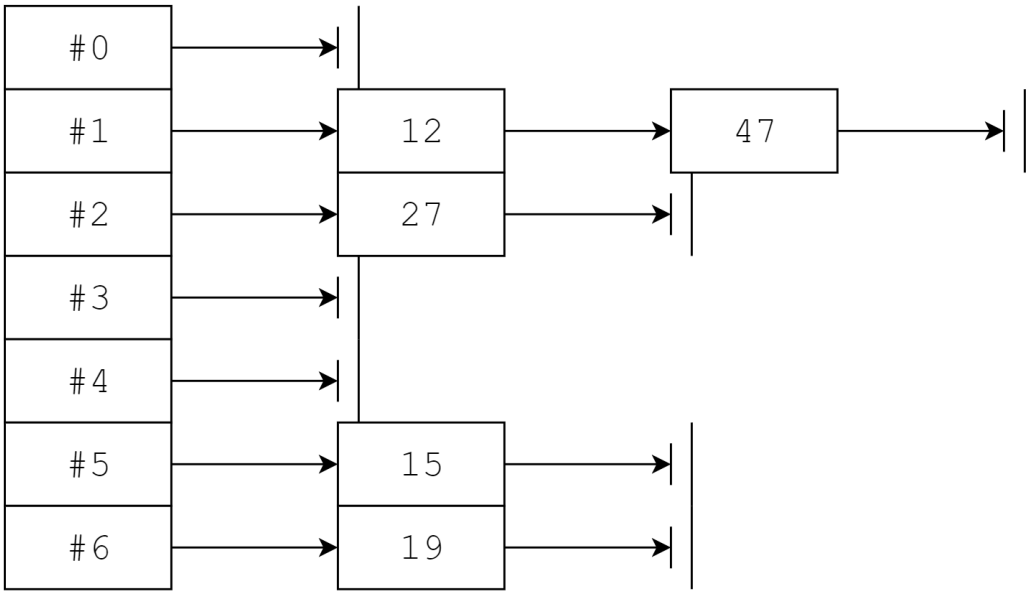
2. Using the same calculations above, but pushing the result to the next available slot:



3. Using the result and notation of the first question:

$$h'(19) = 5 - (19\ mod\ 5) = 5 - 4 = 1 \Rightarrow T[(h + h')(19)\ mod\ 7] = T[6]$$
$$h'(27) = 5 - (27\ mod\ 5) = 5 - 2 = 3 \Rightarrow T[(h + h')(27)\ mod\ 7] = T[2]$$
$$h'(12) = 5 - (12\ mod\ 5) = 5 - 2 = 3 \Rightarrow T[(h + h')(12)\ mod\ 7] = T[1]$$
$$h'(47) = 5 - (47\ mod\ 5) = 5 - 2 = 3 \Rightarrow T[(h + h')(47)\ mod\ 7] = T[1]$$
$$h'(15) = 5 - (15\ mod\ 5) = 5 - 0 = 5 \Rightarrow T[(h + h')(15)\ mod\ 7] = T[5]$$

As noted earlier, the indices in the table are merely a reference for the reader.



4. Evaluate the function $f(n)$ r times. The table below is used to organize the results:

n	$f(n) = f(n-1) + 2n + 1$
1	3
2	$f(1) + 2(2) + 1 = 8$
3	$f(2) + 2(3) + 1 = 15$
4	$f(3) + 2(4) + 1 = 24$
...	...
$r-1$	$f(r-2) + 2(r-1) + 1$
r	$f(r) = f(r-1) + 2r + 1$

$$f(r) = f(r-1) + 2r + 1 = f(r-2) + 2(r-1) + 2r + 2$$
$$= f(r-3) + 2(r-2) + 2(r-1) + 2r + 3$$
$$= f(r-4) + 2(r-3) + 2(r-2) + 2(r-1) + 2r + 4$$

Note for every round of substitution, we have three distinct terms: $f(r-i) + 2(r-i-1) + 1$, where i is an integer less than r. Then, the result is 2 distinct summations and one recurrence call:

$$f(r) = f\big(r - (r-1)\big) + 2\sum_{i=0}^{r-2}(r-i) + \sum_{i=1}^{r-1} 1$$

$$\therefore f(r) = f(1) + 2\sum_{i=0}^{r-2}(r-i) + r - 1$$

$$= 3 + 2\left(\sum_{i=0}^{r-2} r - \sum_{i=0}^{r-2} i\right) + r - 1 \text{ (Breaking the sum into two)}$$

$$= 2\left(\sum_{i=1}^{r-1} r - \sum_{i=1}^{r-1} i - 1\right) + r + 2 \text{ (Shifting index of summation)}$$

$$= 2\left(\sum_{i=1}^{r-1} r - \left(\sum_{i=1}^{r-1} i - \sum_{i=1}^{r-1} 1\right)\right) + r + 2 \text{ (Breaking the sum further)}$$

$$= 2\left(r(r-1) - \left(\frac{r(r-1)}{2} - (r-1)\right)\right) + r + 2$$

$$= 2(r^2 - r) - (r^2 - r) + 2(r-1) + r + 2$$
$$= r^2 - r + 2r - 2 + r + 2$$
$$= r^2 + 2r$$

But r is any integer, so let $r = n$, then we have the final formula given as:

$$f(n) = n^2 + 2n$$

$$\therefore f(n) \in O(n^2)$$

We verify the result above using induction with the base case as $f(1) = 3$. Assuming the results hold for $r \in \mathbb{N}$. That is, we know that $f(r) = r^2 + 2r$. By the recurrence relation:

$$f(r+1) = f(r) + 2(r+1) + 1$$

Then, substituting the new equation for $f(r)$, we obtain:

$$f(r+1) = r^2 + 2r + 2r + 2 + 1$$
$$= (r^2 + 2r + 1) + (2r + 2)$$
$$= (r+1)^2 + 2(r+1)$$

Which is exactly what the new equation returns for $f(r+1)$. ■

5.

```
Algorithm: isSymmetric(r)
Input: r, the root of a tree.
Output: true if the tree with root r is symmetric, and false otherwise.

01  if (isLeaf(r)) then return true
02  else {
03      symmetric ← true
04      q ← children(r).getNext()
05      for each child c of r do {
06          symmetric ← (symmetric and q.value = c.value)
07          if (symmetric = false) then return false }
08      for each child c of r do {
09          symmetric ← (symmetric and isSymmetric(c))
10          if (symmetric = false) then return false }
11      return symmetric }
```

Remark: Line 4 assumes `children(r)` returns an iterator that contains pointers to child nodes of r. In addition `getNext()` is assumed to be the accessor method of that iterator with $O(1)$ complexity.

In the analysis below, we use the following notation:

$$i = \text{number of internal nodes}$$
$$e = \text{number of external nodes}$$

Note that the worst case for this algorithm is if the tree is symmetric. This is because if it were, all the nodes must be visited to verify symmetry. On the other hand, the verification process is terminated early as soon as the algorithm runs into an asymmetric subtree. Assume the base case has a constant complexity c_0. Ignoring the recursion in the recursive case, we have two loops with variable complexity and a few lines with constant complexity c_1. Assume the body of each loop has constant complexities c_2 and c_3 respectively. Note that both loops iterate over the same number of children. Hence, let us denote that number by n_k, which has the property that:

$$\sum_i n_k = n - 1$$

This property is due to the fact that at every different internal node, n_k varies. The subtracted one is the root because it has no parent. As such, the recursive part has a total complexity of $n_k(c_2 + c_3) + c_1$. Focusing on the recursion now, we call the algorithm one time per node. Thus, we repeat the complexities of the base case and recursive case for as many leaves and internal nodes respectively to compute the total running time of the algorithm:

$$T(n) = \left(\sum_i (n_k(c_2 + c_3) + c_1)\right) + \sum_e c_0$$
$$T(n) = (c_2 + c_3)(n - 1) + c_0 e + \sum_i c_1$$
$$T(n) = (c_2 + c_3)(n - 1) + c_0 e + c_1 i$$
$$\therefore O\big(T(n)\big) = O\big((c_2 + c_3)(n - 1) + c_0 e + c_1 i\big)$$
$$O\big(T(n)\big) = O\big((n - 1) + e + i\big) = O(2n - 1) = O(n)$$

Note the identity $n = e + i$ has been used above. ■