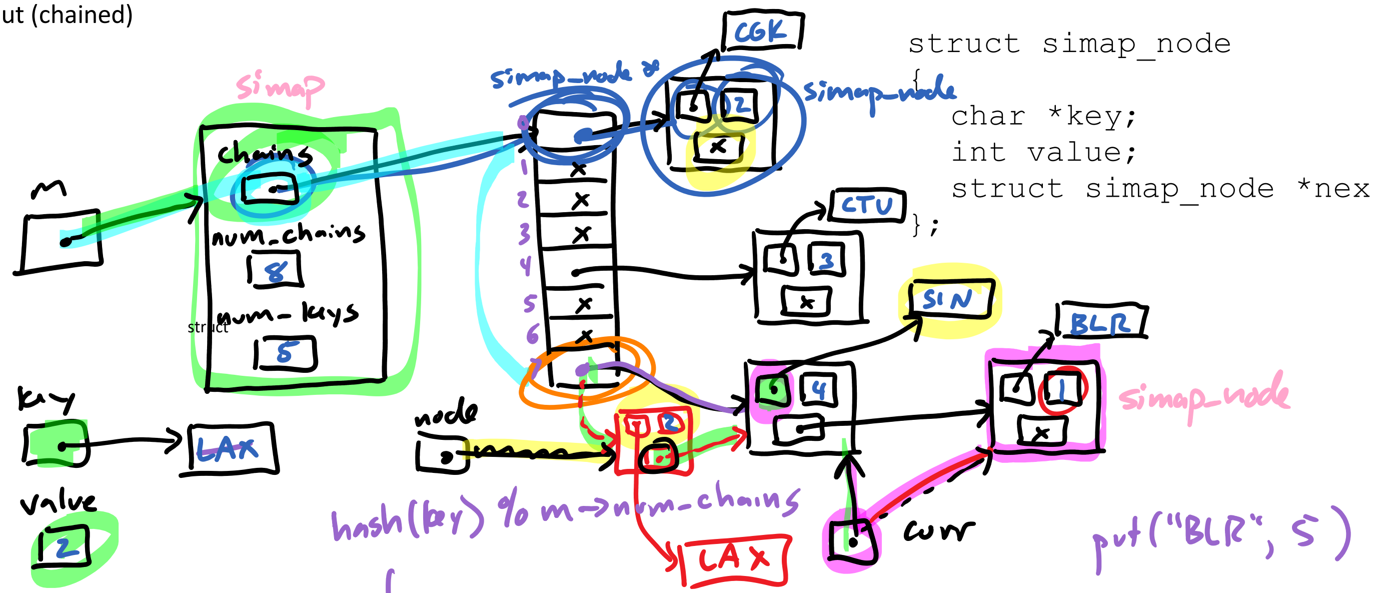
Put (chained)



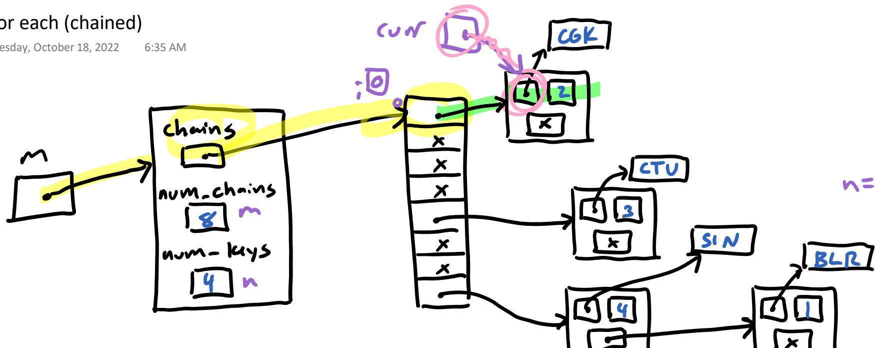
```
struct simap_node
{
   char *key;
   int value;
   struct simap_node *next;
};
```

```
void simap_put(simap *m, const char *key, int value) {
   size_t chain = compute_index(m, key);
   simap_node *curr = m->chains[chain];
   while (curr != NULL && strcmp(curr->key, key) != 0)
      curr = curr->next;
   if (curr == NULL) {
      simap_node *node = malloc(sizeof(*node));
      node->next = m->chains[chain];
      m->chains[chain] = node;
      m->num_keys++;
      node->value = value;
      node->key = malloc(strlen(key) + 1);
      strcpy(node->key, key);
   }
   else {
      curr->value = value;
   }
```

```
struct simap{
   size_t num_chains;
   size_t num_keys;
   simap_node **chains;
};
```

For each (chained)

Tuesday, October 18, 2022 6:35 AM



n = # keys

```
void simap_for_each(simap *m, void (*f)(const char *key, int value, void *arg), arg) {
  for (size_t i = 0; i < m->num_chains; i++) {
     for (simap_node *curr = m->chains[i]; curr != NULL; curr = curr->next)
     {
        f(curr->key, curr->value, arg);
     }
  }
```

$O(1)$ before loop: $C_=$

assignment $C_=$

compare $C_<$

increment C_{++}

$\rightarrow$ $C_{\rightarrow}$

[ ] $C_{[]}$

( ) $C_{()}$

n_0 might be 0, so don't know $n_0 > 1$ ↓ keep both

	i	work for outer loop
$O(n_0+1)$	0	$C_< + C_{++} + 2C_{\rightarrow} + 2C_= + C_{[]}$ $+ (C_< + 3C_{\rightarrow} + 4C_= + C_{()})n_0$
$O(n_1+1)$	1	$C_< + C_{++} + 2C_{\rightarrow} + 2C_= + C_{[]}$ $+ (C_< + 3C_{\rightarrow} + 4C_= + C_{()})n_1$
⋮	⋮	⋮
$O(n_{m-1}+1)$	m − 1	$C_< + C_{++} + 2C_{\rightarrow} + 2C_= + C_{[]}$ $+ (C_< + 3C_{\rightarrow} + 4C_= + C_{()})n_{m-1}$

$O(1)$ after loop

$$O(1 + (n_0+1) + \cdots + (n_{m-1}+1) + 1)$$
$$= O(n + m + 2)$$
$$= O(n + m)$$

and same answer if we only count fxn calls and initializations of curr

$$(C_< + C_{++} + 2C_{\rightarrow} + 2C_= + C_{()})m$$
$$+ (C_< + 3C_{\rightarrow} + 4C_= + C_{()})(n_0 + n_1 + \cdots + n_{m-1})$$

$$= (C_< + C_{++} + 2C_{\rightarrow} + 2C_= + C_{()})m$$
$$+ (C_< + 3C_{\rightarrow} + 4C_= + C_{()})n + C_= + C_< + C_{\rightarrow}$$

$O(n+m)$

some constant times n
+
some constant times m
+
some low-order terms (insignificant as $n, m \rightarrow \infty$)

$$=O(n+4n)$$
$$=O(5n)$$
$$=O(n)$$

$\alpha = \frac{n}{m} < 1$

$m > n$

$m < 4n$

$\frac{1}{4} < \frac{n}{m} = \alpha$

making sure $\alpha > \frac{1}{4}$ for large n ensures O(n) time for for_each

$O(n)$ vs. $O(n^2)$

$10^9 n$ vs. $\frac{1}{1000} n^2$

there is some n_0 s.t. for any $n > n_0$

$10^9 n < \frac{1}{1000} n^2$

load factor $\frac{n}{m}$

smap-put runs in expected $O(\alpha)$ time (treating key operations as $O(1)$)

ensure $\alpha < 1$

so expected $O(1)$ time

Map Implementation Summary

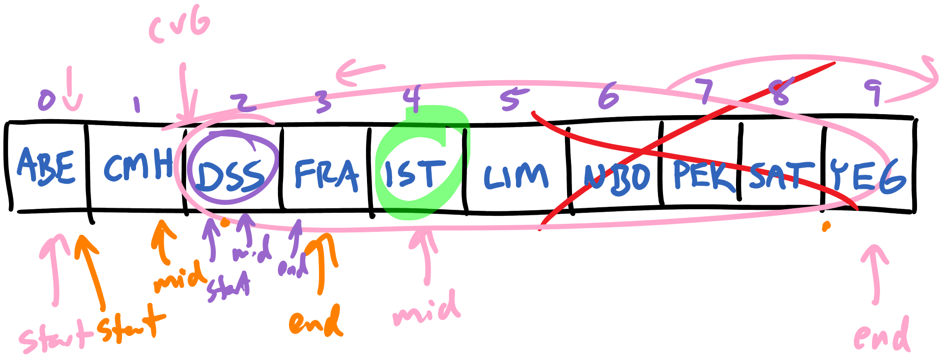
and Set

	unsorted list (array/linked)	sorted list (array)	sorted list (linked list)	hash table
contains/get	O(n)	O(log n) binary search	O(n)	O(1) expected O(n) worst-case
put add	O(n)	O(n) insert	O(n)	O(1) expected O(n) worst-case
remove	O(n)	O(n) fill hole	O(n)	O(1) expected O(n) worst-case
for_each	O(n)	O(n)	O(n)	O(n)
keys_sorted	O(n logn)	O(n)	O(n)	O(n log n)

under assumptions about α and hash

Tuesday, October 18, 2022 7:27 AM

```
bool binary_search_s(const char * const a[], const char *key, int *index, int item_count) {
  int start = 0, end = item_count - 1;
  while (start <= end) {
      int mid = (start + end) / 2;
      int result = strcmp(key, a[mid]);
      if (result == 0) {
            *index = mid;
            return true;
      }
      else if (result < 0)
            end = mid - 1;
      else
            start = mid + 1;
  }
  *index = start;
  return false;
}
```



key DSS

O(log n) iterations

O(1) key compares per iteration

(plus O(1) total for everything else in loop)

O(log n) key comparisons total

size of range = start - end + 1 $\leq \frac{n}{2^{\# \text{ of iterations}}}$

loop terminates at least when size of range < 1

$\frac{n}{2^{\# \text{iterations}}} < 1 \rightarrow n < 2^{\# \text{iterations}}$

$\log_2 n < \# \text{ iterations}$

so # iterations is $\leq \log_2 n + 1$