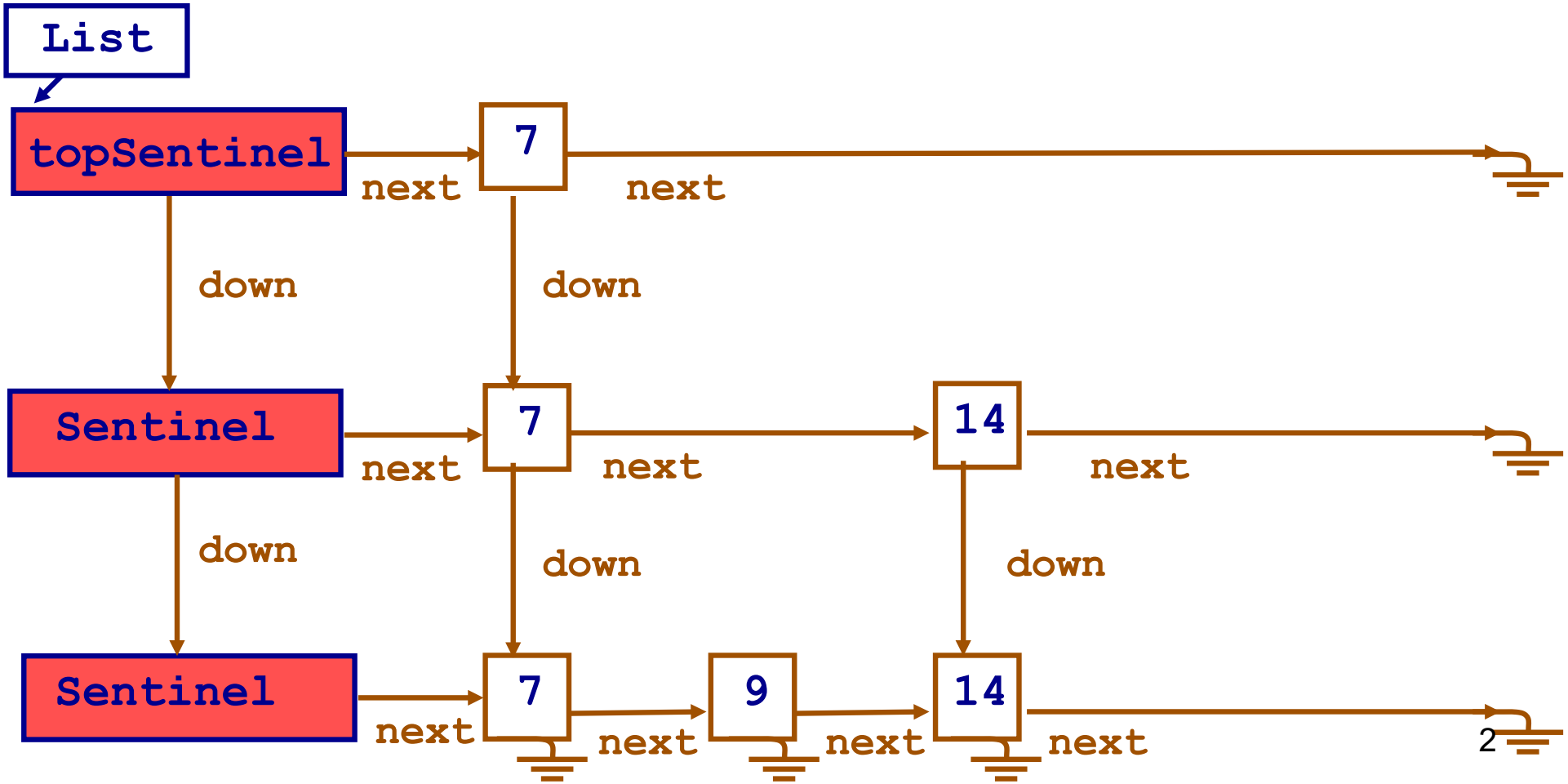
CS 261: Data Structures

Skip Lists Practice

Skip Lists

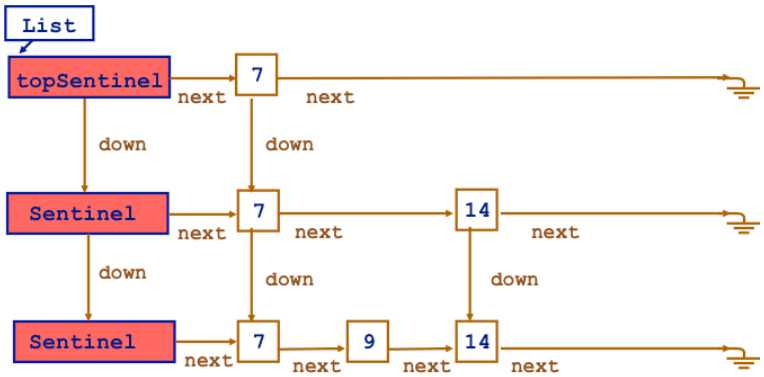


Skip Link

```
struct skipLink{
    TYPE value;
    struct skipLink *next;
    struct skipLink *down;
}

struct skipList{
    struct skipLink *topSentinel;
    int size
}
```

not equal to the number of links



slideRight – returns one link before

```
struct skipLink * slideRight
        (struct skipLink * current, TYPE e)
{
   assert(current);

   while( current->next != 0  &&
          LT(current->next->value, e) )
   {
      current = current->next;
   }

   return current;
}
```

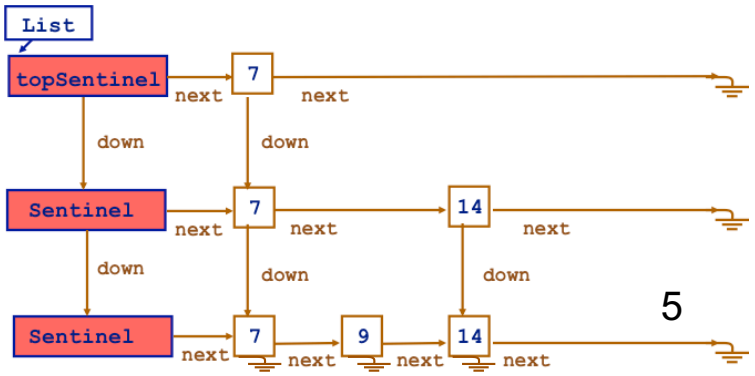
Skip List Contains

```
int skipListContains (struct skipList *slst, TYPE e){
    assert(slst);
    struct skipLink *current = slst->topSentinel;

    while (current) {
      current = slideRight(current, e);/*take express*/

      if ( current->next != 0 &&
           EQ(current->next->value, e)) return 1;
       current = current->down; /*take a slower train*/
    }
    return 0;
}
```

test first if `next` exists

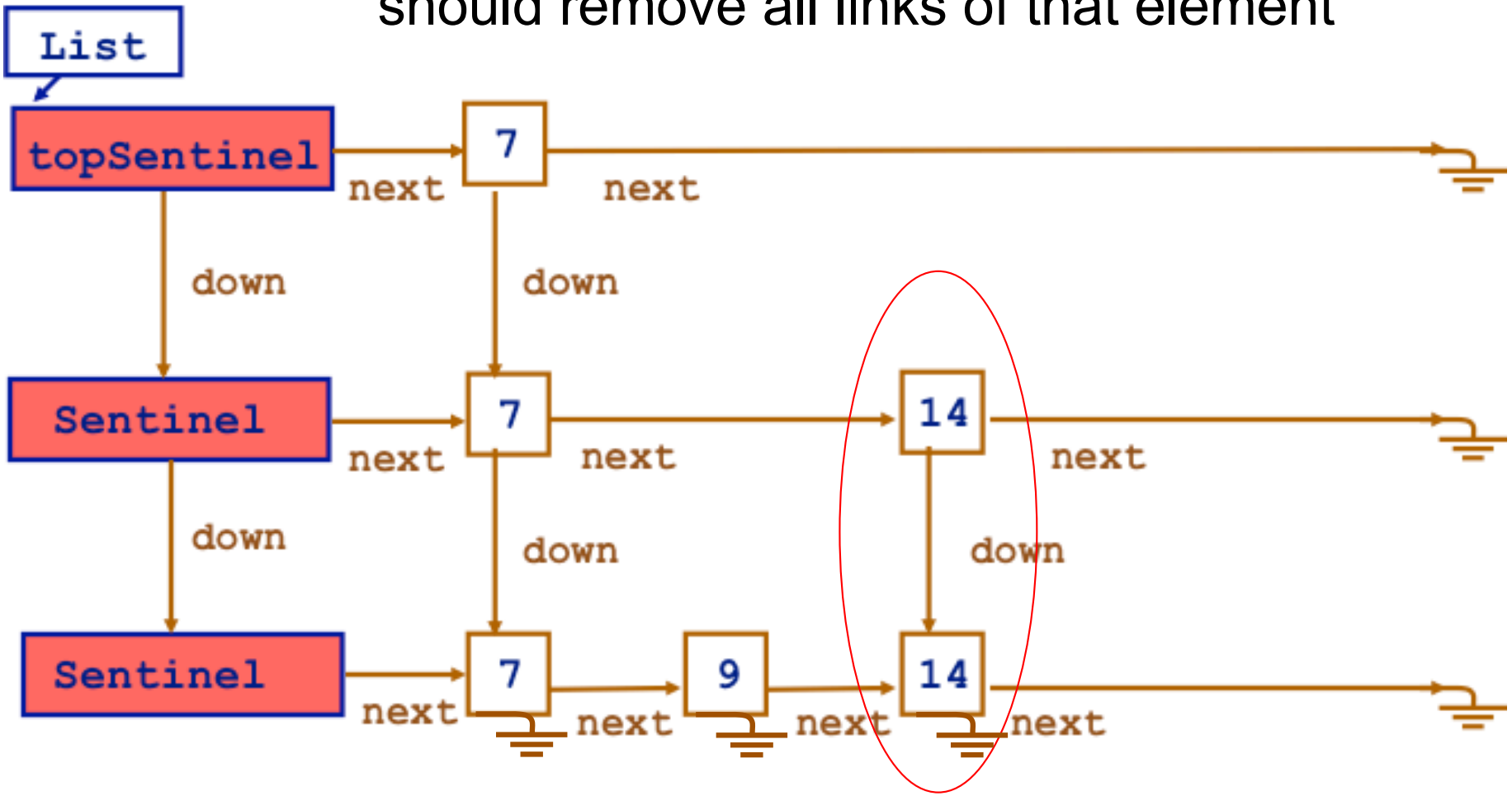


Complexity of Contains

- Makes a zig-zag motion top-to-bottom
- Complexity: O(log n), i.e., proportional to the height

Remove Skip List

should remove all links of that element

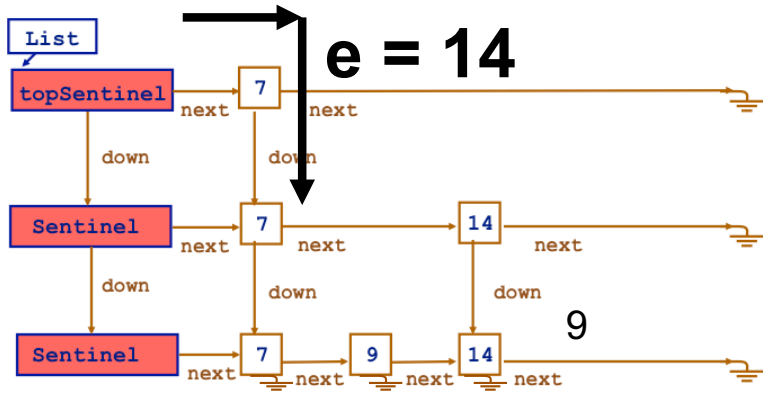


Remove Skip List

- Start at topmost sentinel
- Loop as follows
  - Slide right, get a link right before
  - If next element is OK, remove it
  - If there is no down element,
    - reduce size
  - else, move down

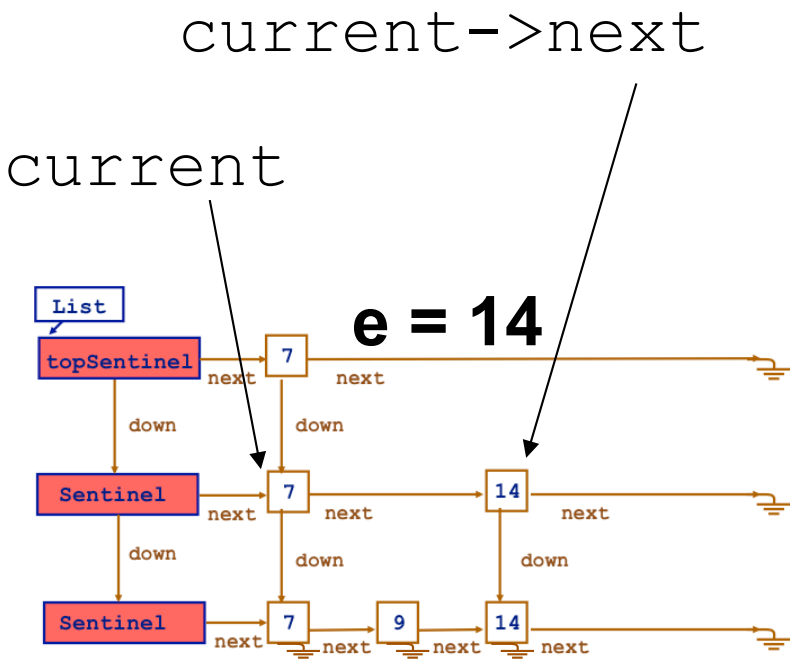
Skip List Remove

```
void removeSkipList (struct skipList *slst, TYPE e) {
    assert(slst);
    struct skipLink *current, *temp;
    current = slst->topSentinel;
    while (current) {
        current = slideRight(current, e);
        if (...) { /* found it */
                ...
        }
        current = current->down; /* move down */
    }
}
```



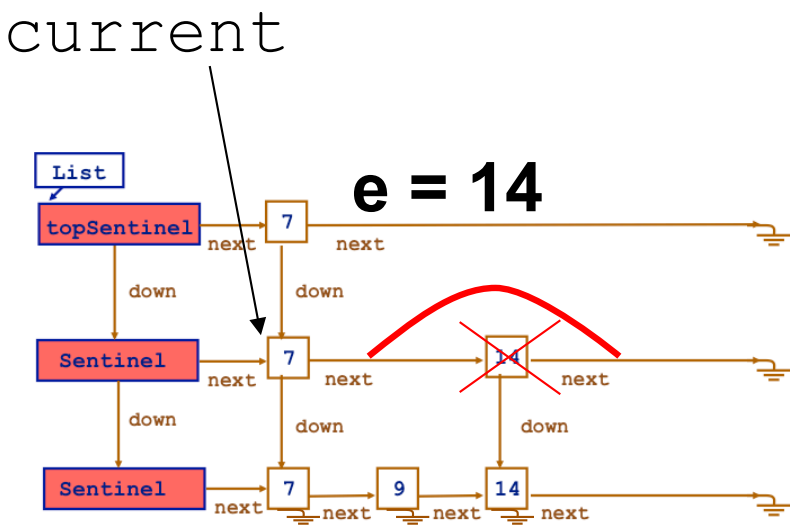
Skip List Remove

```
void removeSkipList (struct skipList *slst, TYPE e) {
    ...
    while (current) {
        current = slideRight(current, e);/*link before*/
        if( current->next != 0 &&
            EQ(current->next->value, e))
        {
            /* found it */
        }
    ...
```



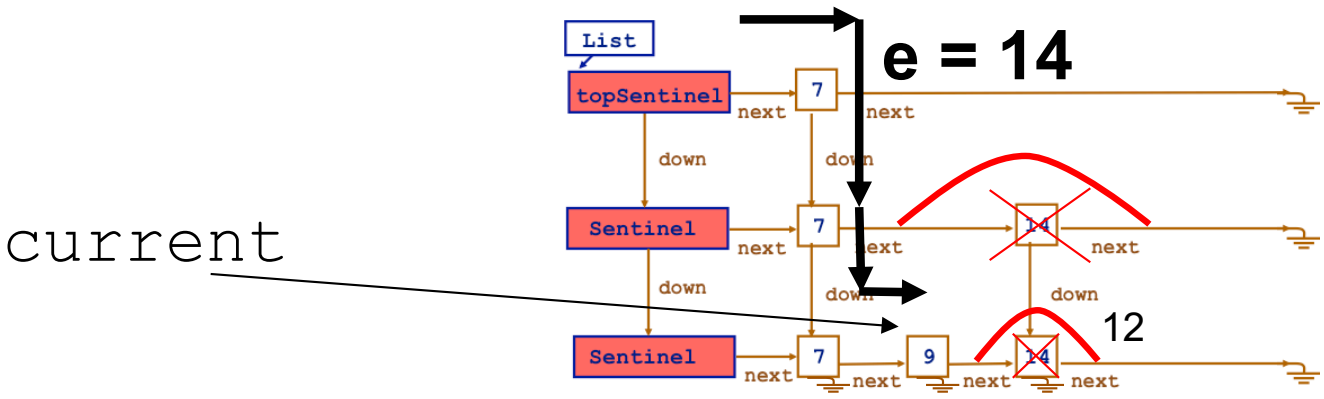
Skip List Remove

```
void skipListRemove (struct skipList *slst, TYPE e) {
    ...
       if (...) { /* found it */
           temp  =  current->next;
           current->next = current->next->next;
           free(temp);
           if (current->down == NULL)
              slst->size--; /* only at the bottom */
       }
      ...
```



Skip List Remove

```
void removeSkipList (struct skipList *slst, TYPE e) {
    ...
    current = slst->topSentinel;
    while (current) {
        current = slideRight(current, e);
        if (...) { /* found it */
            ...
            if (current->down == NULL) slst->size--;
        }
        current = current->down; /* move down */
    }
}
```



Remove Skip List

- Only decrement the size at the bottom
- Makes a zig-zag motion top-to-bottom
- Complexity: O(log n), i.e., proportional to the height

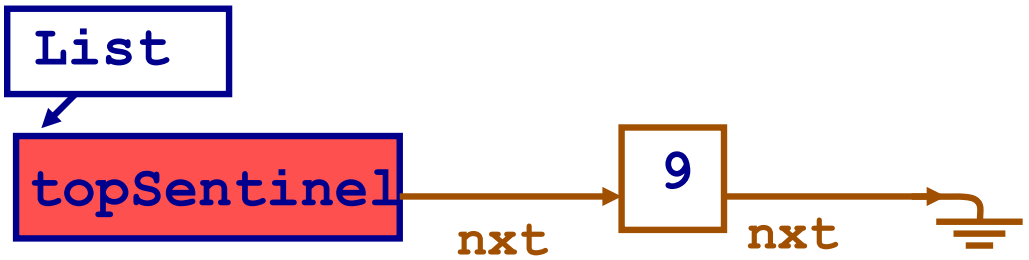
Add Skip List

- Add the element to the bottom list
- To move up to existing lists:
  - Flip a coin, and if heads add the element up; o.w. stop
- To add a new list at the top, flip a coin and if heads:
  - Make a new top list,
  - Add the element to the new top list
- Increment the size

Add

Insert: 9 14 20 21 7

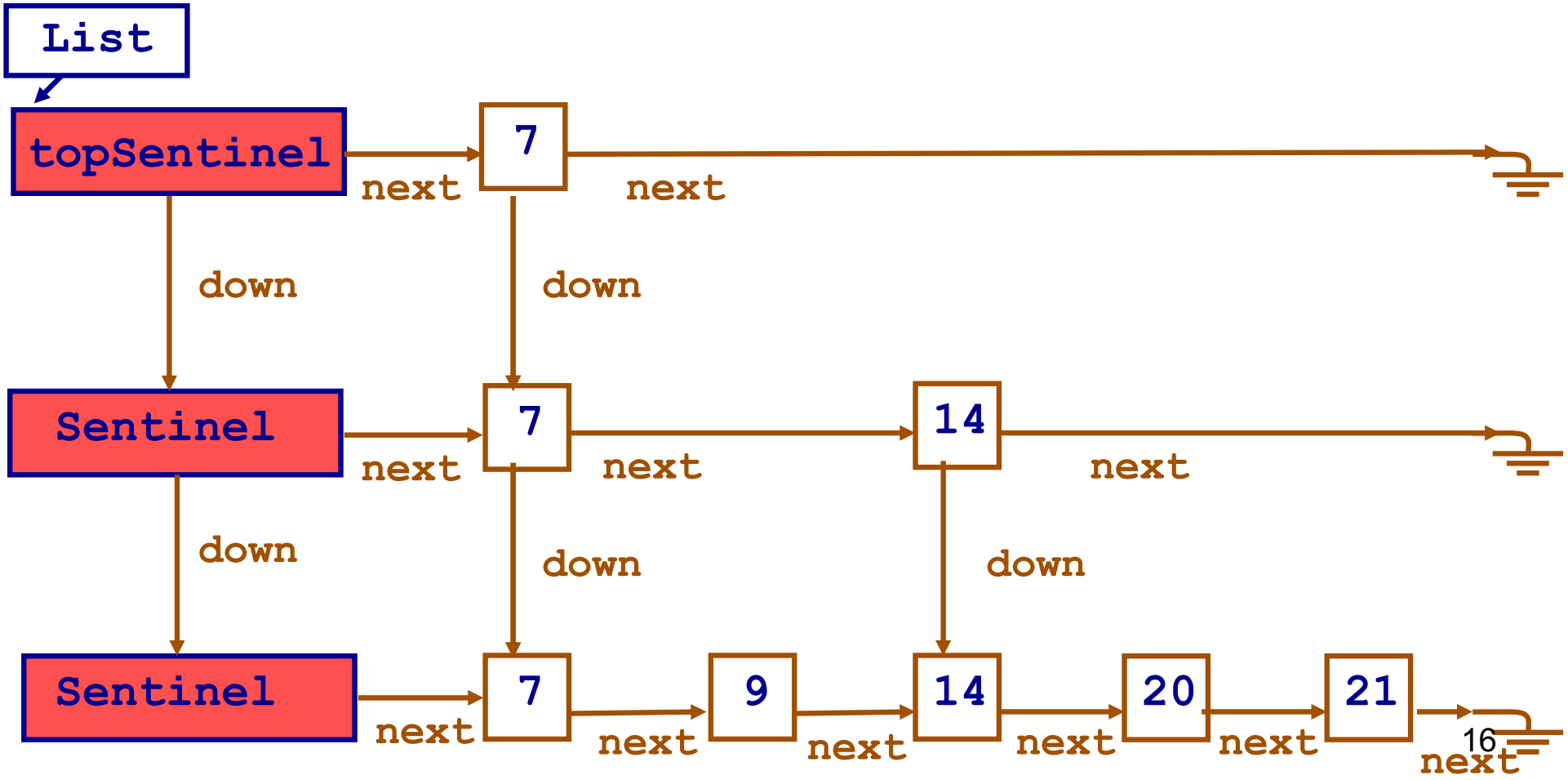
Coin toss: T H T T T H H T



Add

Insert: 9 14 20 21 7

Coin toss: T H T T T H H T



Auxiliary Functions

```
int coinFlip(){
 return rand() % 2;
}
```

```
struct skipLink * makeSLink (TYPE e, struct
   skipLink *next, struct skipLink* down)
{
  struct skipLink *tmp = (struct skipLink *)
              malloc(sizeof(struct skipLink));

   assert(tmp != 0);

   tmp->value = e;
   tmp->next = next;
   tmp->down = down;
   return tmp;
}
```

Add Skip List

```
void addSkipList (struct skipList *slst, TYPE e) {
    ...
    /* recursively add the new element */
    downLink = addSLink(slideRight(slst->sentinel, e), e);
    if (downLink && coinFlip()){
    /* if we need to make a new top list */
    }
}
```

add to existing lists

make a new top list

Add Skip List

```
void addSkipList (struct skipList *slst, TYPE e) {
    ...
    /* recursively add the new element */
    downLink = addSLink(slideRight(slst->sentinel, e), e);

    if (downLink && coinFlip()){
    /* if we need to make a new top list */

    /* 1) make a new link for e, pointing down to downLink
       2) make new sentinel, pointing next to new link
          and pointing down to existing sentinel */
    }
}
```

add to existing lists

make a new top list

slide right to one link before,
then recursively move down

Add Skip List

```
void skipListAdd(struct skipList * slst, TYPE e) {
    struct skipLink *downLink, *newLink;
    downLink =
        skipLinkAdd(slideRight(slst->topSentinel,e),e);

    if (downLink && coinFlip()) {
        newLink = makeSLink(e, 0, downLink);
        slst->topSentinel =
            makeSLink(0, newLink, slst->topSentinel);
    }
    slst->size++;
}
```

add to existing lists

make a new top list

Add Skip Link – Recursive

one link before

```
struct skipLink* addSLink(struct skipLink *lnk, TYPE e){
    ...
    if (lnk->down == 0)
    {/* always first check when to stop the recursion */
       ... /* if we hit the bottom */
    }
    else
    {
       /* recursive call */
       down = addSLink(slideRight(lnk->down, e), e);
       ...
    }
    ...
}
```

move down, and slide right to one link before

Add Skip Link

one link before

```
struct skipLink* addSLink(struct skipLink *lnk, TYPE e){
   struct skipLink *new, *down;
   new = 0;
   assert(lnk);
   if (lnk->down == 0)
   {  /* if we hit the bottom */
      new = makeSLink(e, lnk->next, 0);
      lnk->next = new;
   }
   else
   {
      ...
   }
   return new;
}
```

Add Skip Link

```
struct skipLink* addSLink(struct skipLink *lnk, TYPE e){
    ...
    else
    {  /* recursive call */
       down = addSLink(slideRight(lnk->down, e), e);

       if (down && coinFlip()) {
         /* form a copy in the list above */
         new = makeSLink(e, lnk->next, down);
         lnk->next = new;
       }
    }
    return new;
}
```

move down, and slide right to one link before

Add Skip Link

```
struct skipLink* addSLink(struct skipLink *lnk, TYPE e){
   struct skipLink *new, *down;
   new = 0;
   assert(lnk);
   if (lnk->down == 0) {
      new = makeSLink(e, lnk->next, 0);
      lnk->next = new;
   } else {
      down = addSLink(slideRight(lnk->down, e), e);
      if (down && coinFlip()) {
        new = makeSLink(e, lnk->next, down);
        lnk->next = new;
      }
   }
   return new;
}
```

Complexity of Add

- Proportional to height, not to the number of nodes in the list
- O(log n)