Chapter 3

- (Abstract Data Types) ADTs
- Lists, Stacks, and Queues
- Skip radix sort (we will come back to this when we talk about sorting).

ADT is an abstraction defined by a set of operations.

Examples:

- Integer ADT: $+$, $*$, -.
- Set ADT: union, intersection, size.

Why are ADTs important?

Questions to ask when analyzing an ADT:

- What are the operations?
- What is the underlying data structure?
- How fast can the operations be carried out?
- What is the frequency of the operations?

The List ADT

A list of size 0 is a null list.

Simple operations: print list, make null, next, and previous.

Key operations:

- FIND(item) = position

- FIND_Kth(position) = item

- INSERT(item, position)

- DELETE(position)

What are some possible data structures?

Linked List vs Array

Note: Typically, array size has to be declared in advance.

	ARRAY	**LIST**	**Sorted LIST**
FIND			
FIND_Kth			
INSERT			
DELETE			

Sparse Tables

The obvious implementation of a table or matrix is as a 2-D array.

r is the number of rows.
c is the number of columns.
The size of the array is $O(r * c)$;

This is a huge waste of space when there are very few elements actually in the table.

Imagine Table of 40,000 students and 2,500 courses.
size of the array is $2,500 \times 40,000 = 10^8$
The average student registers for about 4 classes.
Only 160,000 entries.
What's a better representation for sparse tables?

Linked List Representation

For any kind of sparse table, a better representation is a linked list. See Fig. 3.29.
Size $O(r + c + e)$ where e is the number of elements that are actually filled in.
Determine:
Who's taking 373?
What courses is Joe taking?

The Stack ADT

A LIFO discipline (first in, last out):

Operations: PUSH(item, stack) = new-stack

POP(stack) = top item

	ARRAY	LIST
PUSH		
POP		
FIND_Kth		

List Inversion (Simple example)

A stack is to invert the list E,S,R,U,O,C.
Algorithm:

```
Repeat-until-done:
     S := READ(input);
     push(S, stack)

Repeat-until-done: pop(S, stack); write(S);
```

See section 3.3.3 for a complex example.

The Queue ADT

A FIFO discipline (first in, first out):

- ENQUEUE(x, queue): inserts an element at the end of the queue.

- DEQUEUE(queue): deletes and returns the first item in the queue.

List implementation: keep a front and rear pointer and update each position with the dequeue and enqueue operations.

Array Implementation of Queues

F = index to front of queue.
R = index to rear of queue.

To enqueue, what do we do?

To dequeue, what do we do?

What is the problem with this approach?