CMPSC 24: Lecture 8
Abstract Data Type: Lists

Divyakant Agrawal
Department of Computer Science
UC Santa Barbara

Recap

- Abstract Data Types
  - Specification (What?)
  - Implementation (How?)
- Unsorted lists
  - Array-based implementation
  - Pointer-based implementation
- Questions
  - Reference vs. value type
  - Comparing enumerated types
- Exceptions (TODAY)

Lecture Plan

- Linked list implementation of unsorted list
- Time complexity of algorithms
  - Big Oh

ADT Unsorted List

- Transformers
  - MakeEmpty
  - InsertItem
  - DeleteItem
- Observers
  - IsFull
  - GetLength
  - RetrieveItem
- Iterators
  - ResetList
  - GetNextItem

change state

observe state

process all

Specification

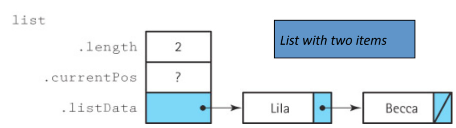
```
// SPECIFICATION FILE      ( unsortedType.h )
#include "ItemType.h"
struct NodeType;

class  UnsortedType     // declares a class data type
{
public :              // 8 public member functions
  UnsortedType();
  void MakeEmpty( );
  bool IsFull( ) const;
  int  GetLength( ) const;  // returns length of list
  void RetrieveItem( ItemType&  item, bool&  found );
  void InsertItem( ItemType  item );
  void DeleteItem( ItemType  item );
  void ResetList( );
  void GetNextItem( ItemType&  item );
```


Linked List Implementation (private part)

```
private:
  NodeType* listData;
  int length;
  NodeType* currentPos;

 struct  NodeType
  {
    ItemType Info;
    NodeType *next;
  }
```



Linked List Implementation

How do you know that a linked list is empty?

`listData` is `NULL`

What should the constructor do?

Set `length` to 0

Set `listData` to `NULL`

What about `currentPos`?

We let `ResetList` take care of initializing `currentPos`

7

Linked List Implementation

What about the observers `IsFull` and `GetLength`?

GetLength just returns length

Can a linked list ever be full?

Yes, if you run out of memory

Ask for a new node within a try/catch

8

Linked List Implementation of IsFull

```
bool Unsortedtype::IsFull() const
{
  NodeType* location;
  try
  {
    location = new NodeType;
    delete location;
    return false;
  }
  catch (std::bad_alloc exception)
  {
    return true;
  }
}
```

What about `MakeEmpty`?

9

Linked List Implementation of MakeEmpty

```
void Unsortedtype::MakeEmpty()
{
  NodeType* tempPtr;
  while (listData != NULL)
  {
    tempPtr = listData;
    listData = listData->next;
    delete tempPtr;
  }
  length = 0;
}
```

Why can't we just set `listData` to NULL?

10

C++ concepts to come

- Exceptions
- Deep vs. shallow copying
- Constructor, destructor, copy constructor
- Dynamically allocated arrays

11

Order of Magnitude of a Function

- The order of magnitude, or Big-O, of a function expresses an upper bound to the growth of a function relative to its parameters.
- Used to analyze the space and time complexity of algorithms/programs.

12

Asymptotic Analysis

- **Ignoring constants in T(*n*)**
- **Analyzing T(*n*) as *n* "gets large"**

Example: $T(n) = 13n^3 + 42n^2 + 2n\log n + 4n$

As n grows larger, n^3 is MUCH larger than n^2, $n\log n$, and n, so it dominates $T(n)$

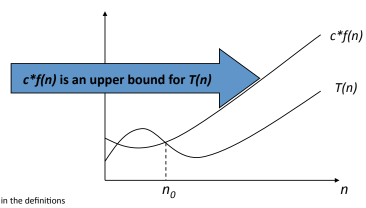
The running time grows "roughly on the order of n^3"

$T(n) = O(n^3)$ ← The big-oh (O) Notation

13

Big-Oh Defined

T(n) = O(f(n)) if there are constants c and n0 such that $T(n) \le c*f(n)$ when $n > n_0$



* Ignoring absolute values in the definitions

14

Big-O Notation

- *T(n) = O(f(n)) if there are constants c and n_0 such that $T(n) \le c*f(n)$ when $n > n_0$*
- If $f(n)=1000n$ and $g(n)=n^2$, $n_0 = 999$ and $c = 1$, then $f(n) \le 1*g(n)$ where $n > n_0$ and we say that $f(n) = O(g(n))$
- The O notation indicates '*bounded above by a constant multiple of.*'

15

Big-Oh Properties

- Fastest growing function dominates a sum
  - $O(f(n)+g(n))$ is $O(\max\{f(n), g(n)\})$
- Product of upper bounds is upper bound for the product
  - If f is $O(g)$ and h is $O(r)$ then fh is $O(gr)$
- f is $O(g)$ is transitive
  - If f is $O(g)$ and g is $O(h)$ then f is $O(h)$
- Hierarchy of functions
  - $O(1)$, $O(\log n)$, $O(n^{1/2})$, $O(n\log n)$, $O(n^2)$, $O(2^n)$, $O(n!)$

16

Some Big-Oh's are not reasonable

- Polynomial Time algorithms
  - An algorithm is said to be polynomial if it is $O(n^c)$, $c > 1$
  - Polynomial algorithms are said to be <u>reasonable</u>
    - They solve problems in reasonable times!
    - Coefficients, constants or low-order terms are ignored
      e.g. if $f(n) = 2n^2$ then $f(n) = O(n^2)$
- Exponential Time algorithms
  - An algorithm is said to be exponential if it is $O(r^n)$, $r > 1$
  - Exponential algorithms are said to be <u>unreasonable</u>

17

Can we justify Big O notation?

- Big O notation is a *huge* simplification; can we justify it?
  - It only makes sense for *large* problem sizes
  - **For sufficiently large problem sizes, the highest-order term swamps all the rest!**

18

Classifying Algorithms based on Big-Oh

- A function $f(n)$ is said to be of at most logarithmic growth if $f(n) = O(\log n)$
- A function $f(n)$ is said to be of at most quadratic growth if $f(n) = O(n^2)$
- A function $f(n)$ is said to be of at most polynomial growth if $f(n) = O(n^k)$, for some natural number $k > 1$
- A function $f(n)$ is said to be of at most exponential growth if there is a constant c, such that $f(n) = O(c^n)$, and $c > 1$
- A function $f(n)$ is said to be of at most factorial growth if $f(n) = O(n!)$.
- A function $f(n)$ is said to have constant running time if the size of the input n has no effect on the running time of the algorithm (e.g., assignment of a value to a variable). The equation for this algorithm is $f(n) = c$
- Other logarithmic classifications: $f(n) = O(n \log n)$
  $f(n) = O(\log \log n)$

19

Names of Orders of Magnitude

$O(1)$ **bounded (by a constant) time**

$O(\log_2 N)$ **logarithmic time**

$O(N)$ **linear time**

$O(N*\log_2 N)$ **$N*\log_2 N$ time**

$O(N^2)$ **quadratic time**

$O(2^N)$ **exponential time**

20

How do various functions grow?

N	$\log_2 N$	$N*\log_2 N$	N^2	2^N
1	0	0	1	2
2	1	2	4	4
4	2	8	16	16
8	3	24	64	256
16	4	64	256	65,536
32	5	160	1024	4,294,967,296
64	6	384	4096	
128	7	896	16,384	

21

Big-O Comparison of List Operations

	Array-based	Pointer-based
Constructor	O(1)	O(1)
IsFull	O(1)	O(1)
GetLength	O(1)	O(1)
RetrieveItem	O(N)	O(N)
MakeEmpty	O(1)	O(N)
InsertItem	O(1)	O(1)
DeleteItem	O(N)	O(N)
ResetList	O(1)	O(1)
GetNextItem	O(1)	O(1)

22