Lecture Plan

• Array Based implementation of sorted list
  – Time Complexity Analysis of Operations on Lists
  – Can we do better?

• Linked List implementation of Sorted List
  – Time Complexity of Operations
  – Can we use the optimization technique here?
ADT Sorted List

• **Transformers**
  – MakeEmpty
  – InsertItem
  – DeleteItem

• **Observers**
  – IsFull
  – GetLength
  – RetrieveItem

• **Iterators**
  – ResetList
  – GetNextItem

- change state
- observe state
- process all
Member functions

Which member function specifications and implementations must must change?

- InsertItem
- DeleteItem
InsertItem for SortedList ADT (array)

• Find proper location for the new element in the sorted list.

• Create space for the new element by moving down all the list elements that will follow it.

• Put the new element in the list.

• Increment length.
Array Implementation

- **InsertItem**
  
  *Initialize location to position of first item*
  
  *Set moreToSearch to (have not examined Info(last))*
  
  *while moreToSearch*
    
    *switch (item.ComparedTo(Info(location)))*
      
      *case LESS : Set moreToSearch to false*
      
      *case EQUAL : // Cannot happen*
      
      *case GREATER :*
        
        *Set location to Next(location)*
        
        *Set moreToSearch to (have not examined Info(last))*
      
      *for index going from length DOWNTO location + 1*
        
        *Set Info(index) to Info(index-1)*
        
        *Set Info(location) to item*
        
        *Increment length*

*Why can't EQUAL happen?*
void SortedType :: InsertItem ( ItemType item )
{
    bool moreToSearch;
    int location = 0;

    // find proper location for new element
    moreToSearch = ( location < length );
    while ( moreToSearch )
    {
        switch ( item.ComparedTo( info[location] ) )
        {
            case LESS : moreToSearch = false;
                        break;
            case GREATER : location++;
                          moreToSearch = ( location < length );
                          break;
        }
    }

    // make room for new element in sorted list
    for ( int index = length; index > location; index-- )
    {
        info[ index ] = info[ index - 1 ];
    }
    info[ location ] = item;
    length++;
}
DeleteItem for SortedList ADT (array)

• Find the location of the element to be deleted from the sorted list.

• Eliminate space occupied by the item being deleted by moving up all the list elements that follow it.

• Decrement length.
Array Implementation

• Deleteltem
  
  Initialize location to position of first item
  Set found to false
  while NOT found
  switch (item.ComparedTo(Info(location)))
    case GREATER : Set location to Next(location)
    case LESS : // Cannot happen
    case EQUAL : Set found to true
  for index going from location +1 TO length -1
    Set Info(index - 1) to Info(index)
  Decrement length

Why can't LESS happen?
void SortedType :: DeleteItem ( ItemType item )
{
    int location = 0 ;
    // find location of element to be deleted
    while ( item.ComparedTo ( info[location] ) != EQUAL )
        location++ ;
    // move up elements that follow deleted item in sorted list
    for (int index = location + 1 ; index < length ; index++)
        info [ index - 1 ] = info [ index ] ;
    length-- ;
}
Let's see

Set location to listData
Set moreToSearch to (location != NULL)
while moreToSearch
    switch (item.ComparedTo(location->info))
        case GREATER :
            Set location to location->next
            Set moreToSearch to (location != NULL)
        case LESS :
            Set moreToSearch to false
See the problem?
Pointer-based Implementation

We need a trailing pointer

(a) listData → Alex → Chris → John → Kit

(b) Set predLoc to location

(c) Set location to Next(location)
Inserting ‘S’ into a Sorted List

Private data:
- length: 3
- listData: ?
- currentPos: ?

predLoc  location

moreToSearch
Finding proper position for ‘S’

Private data:
- length: 3
- listData: ?
- currentPos: ?
- predLoc: NULL
- location: ‘C’
- moreToSearch: true

Diagram:
- PredLoc: NULL
- Location: ‘C’
- MoreToSearch: true
Finding proper position for ‘S’

Private data:
- length
- listData
- currentPos
- moreToSearch
- predLoc
- location

- length: 3
- listData: ?
- currentPos: ?
- moreToSearch: true
- predLoc
- location
Finding Proper Position for ‘S’

Private data:
- length: 3
- listData
- currentPos: ?

moreToSearch: false

Diagram:
- predLoc
- location
- ‘C’ -> ‘L’ -> ‘*’
Inserting ‘S’ into Proper Position

Private data:
- length: 4
- listData
- currentPos
- predLoc
- location
- moreToSearch: false

Diagram:
- Node 'C' connected to 'L'
- Node 'X' connected to 'S'
- 'S' connected to moreToSearch (false)
Deleteltem for SortedList ADT (pointers)

• Any changes?
How to improve searching in sorted lists?

• Binary search
Binary Search in a Sorted List

- Examine the element in the middle of the array. Is it the sought item? If so, stop searching. Is the middle element too small? Then start looking in second half of array. Is the middle element too large? Then begin looking in first half of the array.

- Repeat the process in the half of the list that should be examined next.

- Stop when item is found, or when there is nowhere else to look.
```cpp
void SortedType::RetrieveItem ( ItemType& item, bool& found )
{
    int midPoint ;
    int first  =  0;
    int last   = length - 1 ;

    found = false ;
    while (( first <= last ) && !found )
    {
        midPoint = ( first + last ) / 2 ;  // INDEX OF MIDDLE ELEMENT
        switch ( item.ComparedTo( info [ midPoint ] ) )
        {
        case LESS : . . . // LOOK IN FIRST HALF NEXT
        case GREATER : . . . // LOOK IN SECOND HALF NEXT
        case EQUAL  : . . . // ITEM HAS BEEN FOUND
        }
    }
}
```
Trace of Binary Search

item = 45

<table>
<thead>
<tr>
<th>15</th>
<th>26</th>
<th>38</th>
<th>57</th>
<th>62</th>
<th>78</th>
<th>84</th>
<th>91</th>
<th>108</th>
<th>119</th>
</tr>
</thead>
</table>

info[0]   [1]   [2]   [3]   [4]   [5]   [6]   [7]   [8]   [9]
first    midPoint    last

**LESS**

last = midPoint - 1

**GREATER**

midPoint + 1
item = 45

15 | 26 | 38 | 57 | 62 | 78 | 84 | 91 | 108 | 119

info[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]

first, midPoint

LESS

first = midPoint - 1

midPoint

last = midPoint - 1

GREATER

last = midPoint + 1
item = 45

first > last

loop terminates
```cpp
void SortedType::RetrieveItem ( ItemType& item, bool& found )
// ASSUMES info ARRAY SORTED IN ASCENDING ORDER
{
    int midPoint
    int first = 0;
    int last = length - 1;
    found = false;
    //1
    while (( first <= last ) && !found )
    {
        midPoint = ( first + last ) / 2; //2
        switch ( item.ComparedTo( info[ midPoint ] ) )
        {
            case LESS        :
                last = midPoint - 1;
                break;
            case GREATER     :
                first = midPoint + 1;
                break;
            case EQUAL       :
                found = true;
                item = info[ midPoint ];
                break;
        }
    }
}
```
Loop Invariant

- \( 0 \leq \text{first} \leq \text{last} + 1 \leq \text{length} \)
- \( \text{found} \rightarrow (\text{length} > 0 \land \text{item} = \text{info}[(\text{first} + \text{last})/2]) \)
- \( \neg \text{found} \rightarrow (\text{item not in info}[0..\text{first}-1] \land \text{item not in info}[\text{last} + 1..\text{length}-1]) \)

- At 1:: \( 0 \leq \text{first} \leq \text{last} \leq \text{length}-1 \land J \land K \)
- At 2:: \( 0 \leq \text{first} \leq \text{midpoint} \leq \text{last} \leq \text{length}-1 \land J \land K \)
- LESS:: \( 0 \leq \text{first} \leq (\text{midpoint}-1) + 1 \leq \text{length} \land \text{item not in info}[\text{midpoint}..\text{length}-1] \)
- GREATER:: \( 0 \leq \text{midpoint} + 1 \leq \text{last} + 1 \leq \text{length} \land \text{item not in info}[0..\text{midpoint}] \)