Ordered Bag
Dynamic Array Implementation
Ordered Bag Abstraction

• Same operations as Bag ADT
  – Add
  – Contains
  – Remove

• Property: elements in *sorted order*
Applications of Ordered Collections

- Fast merge
- Set operations $\Rightarrow$ union, intersection, etc.
Fast Merge

5 9 10 12 17

1 8 11 20 32

1

5 9 10 12 17

1 8 11 20 32

1 5

5 9 10 12 17

1 8 11 20 32

1 5 8

5 9 10 12 17

1 8 11 20 32

1 5 8 9

5 9 10 12 17

1 8 11 20 32

1 5 8 9 10
Fast Merge (cont.)
Set Operations: Similar to Merge

• You can quickly merge two ordered arrays into a new ordered array
  – What is its complexity? → O(n)

• Set operations (intersection, union, difference, subset) are similar to merge
  – Try these on your own...(See Chapter 9)
Maintaining an Ordered Bag

2 Options

• Add to the bag then sort (we’ll see this later)

• *Add to the bag in the correct position*
Binary Search

• The formal name for this process is **binary search**
• Each step cuts region containing the value in half
• Starting with *n* items, how many times can I cut in half before reaching a set of size one?
Binary Search: $O(\log n)$

- A $O(\log n)$ search is much much faster than an $O(n)$ search (for large $n$)
- $\log_2 1,000,000 \sim 20$
- Log of largest unsigned integer value in C (4294967295) is 32
  - Instead of 4 billion comparisons, you only need 32
What are the requirements for performing a binary search?

- Random access to the elements
- Elements are already in sorted order
Binary Search Ordered Array: Intuition

- Compute the middle index
- Check for the value at that index
- If found the value, done, return the index
- If not found
  - If value is less than value at the index, repeat with left half of array
  - Else repeat with right half of array
int _binarySearch(TYPE * data, int size, TYPE val) {
    int low  = 0;
    int high = size;
    int mid;
    while (low < high) {
        mid = (low + high) / 2;
        //mid less than val looking for
        if (LT(data[mid], val))
            low  = mid + 1;
        else    high = mid;
    }
    return low;
}
Binary Search Ordered Array: Return Value

- If value is found, returns index of that value
- If value is not found, returns position where it can be inserted without violating ordering
- **NOTE:** returned index can be larger than a legal index

```c
int _binarySearch(TYPE * data, int size, TYPE val) {
    int low = 0;
    int high = size;
    int mid;
    while (low < high) {
        mid = (low + high) / 2;
        if (LT(data[mid], val))
            low = mid + 1;
        else    high = mid;
    }
    return low;
}
```
Summary

• Searching DynArr and LinkedLists are $O(N)$ on average
• Binary Search provides $O(\log N)$ search but requires that
  – We have random access to data (ie. data is in an array)
  – The data is ordered
• This means, of course, that we can only do efficient binary search on an array (NOT a linked list)
Other ways of maintaining a sorted bag?

- Use a sorting algorithm?
- Linear search for proper location when inserting?
Option 1: Sort upon add

• Add N items
  – Put it at end:  O(1)
  – sort the items  O(NlogN)

• Overall complexity:  N * (1 + NlogN)
  –  O(N^2logN)
Option 2: Linear search, insert

• Add $N$ items
  – Linear search to proper location: $N$
  – Insert at that location: $N$

• Overall complexity: $N \times (N+N)$
  – $O(N^2)$
Option 3: Binary Search

- Add N items
  - _binary Search to find location: logN
  - insert at that location: N

- Total Complexity: N(logN + N)
  - O(N²)
Which operation is now faster?

Using a dynamic array for an Ordered Bag, which of the following operations is made faster or slower by using a binary search?

- add(element)
- contains(element)
- remove(element)

- Are any made slower?
- Which option (1-3) is best?
  - sort
  - linear search
  - binary search
Bug??

In 2006, a bug was found by Google – http://googleresearch.blogspot.com/2006/06/

Where is it?

How can we fix it?

int _binarySearch(TYPE * data, int size, TYPE val) {
    int low = 0;
    int high = size;
    int mid;
    while (low < high) {
        mid = (low + high) / 2;
        if (LT(data[mid], val))
            low = mid + 1;
        else    high = mid;
    }
    return low;
}
Your Turn

• Now that we have _binarySearch, how do the following change?
  – addBag
  – containsBag
  – removeBag

• Complete Worksheet #26
• Read Binary Search Correctness Argument