CS261 Data Structures

Ordered Bag

Dynamic Array Implementation

Goals

- Understand the downside of unordered containers
- Binary Search
- Ordered Bag ADT

Downside of unordered collections

- What is the complexity of finding a particular element in the dynamic array implementation of the Bag
- What about the linked list implementation of the Bag
- Many applications will require a significant number of accesses of particular values…so we need a more efficient means for finding values.

Power of Ordered Collections

- Why do you suppose that dictionaries or phonebooks keep their elements in order?
- Suppose I asked you to find the phone number for Chris Smith?
- Suppose I asked you who was the person with phone number 753-6692?

Guess My Number

- We all know the heuristic of cutting a collection in half from the game "guess my number"
- I'm thinking of a number between 1 and 100. What questions will you ask to find my number (efficiently)?

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
- $\text{Log}_2\ 1{,}000{,}000 \sim 20$
- Log of largest unsigned integer value in C (4294967295) is 32

Binary Search…

- What are the requirements for performing a binary search?

Binary Search Requirments

- 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

Binary Search: Ordered Array Algorithm

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

```
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;
}
```

Ordered Bag Abstraction

- Same operations as Bag ADT
  - Add
  - Contains
  - Remove
- Property: elements maintain sorted order
- How can we do this efficiently?

Which operation is now faster?

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

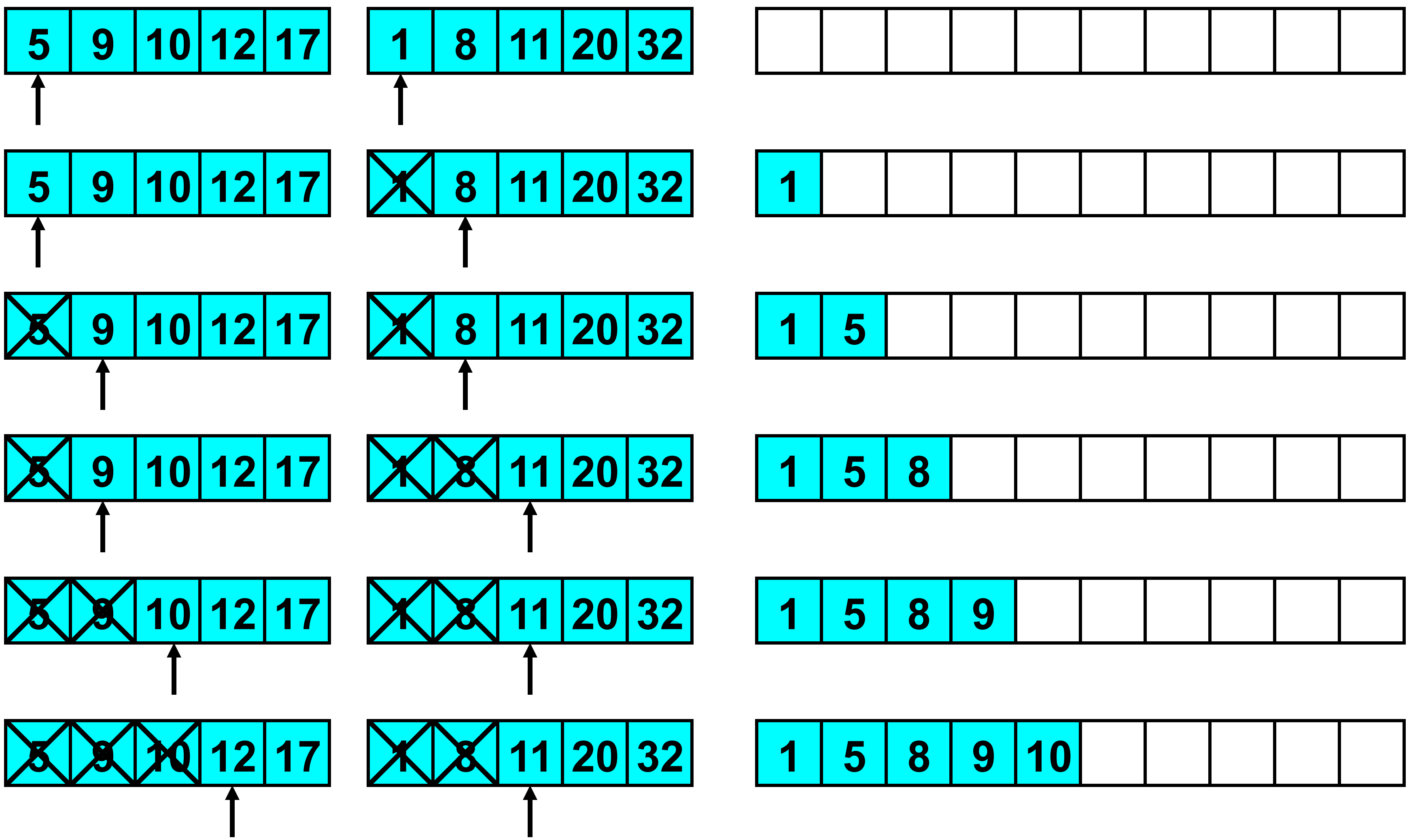
- add(element)
- contains(element)
- remove(element)

- Are any made slower?

Applications of Ordered Collections

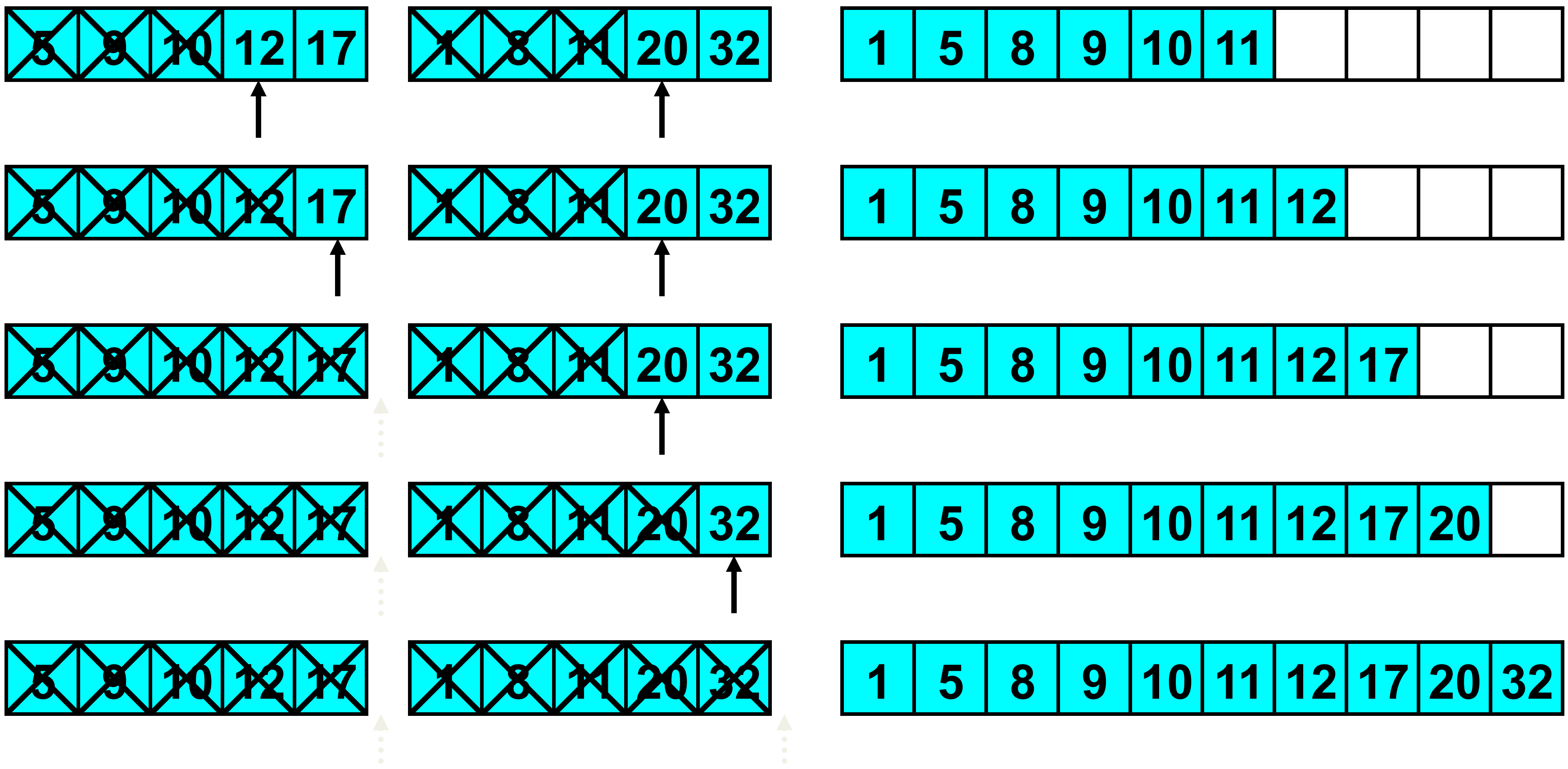
- You will get your chance to write an implementation of a ordered bag
- But first, other reasons for keeping a collection of elements in order:
  - Fast merge
  - Set operations → union, intersection, etc.

Fast Merge



5 9 10 12 17
1 8 11 20 32
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)

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)

Question?

- Why not just sort the array every time you add an element to the collection? Is it more/ less efficient…or the same?

Your Turn

- Now that we have _binarySearch, how do the following change?
  - addBag
  - containsBag
  - removeBag
- Complete Worksheet#26
- Read Binary Search Correctness Argument