CS 261 Spring 2011

Sorted Dynamic Array
Bag and Set
by
Tim Budd
Ron Metoyer
Sinisa Todorovic
Admin Info

• Class on May 6 canceled

• We will have a substitute instructor on:
  – May 4 (20min quiz counts as Worksheet)
  – May 11
Ordered Collections

• How do we organize data in dictionaries or phonebooks?

• Find in a phonebook:
  – the phone number of John Smith
  – the person with phone number 753-6692
Guess My Number

• Integer numbers are ordered

• I’m thinking of a number in [1, 100]

• Ask questions to guess my number
Binary Search

• The formal name -- Binary Search
• Works by iteratively dividing the interval which contains the value
• Dividing, e.g., in half, in each step
• Suppose we have n items, how many iterations before the interval is of size one?
Log \( n \) search

- A \( \log n \) search is much much faster than an \( O(n) \) search.
Binary Search Algorithm

```c
int binarySearch (TYPE * data, int size, TYPE val){
    int low = 0;
    int high = size;
    while (low < high) {
        ???
    }
    return ???;
}
```
Binary Search Algorithm

```c
int binarySearch (TYPE * data, int size, TYPE val) {
    int low = 0;
    int high = size;
    while (low < high) {
        int mid = (low + high) / 2;
    }
    return ??? ;
}
```
Binary Search Algorithm

int binarySearch (TYPE * data, int size, TYPE val) {
    int low = 0;
    int high = size;
    while (low < high) {
        int mid = (low + high) / 2;
        if (data[mid] < val)
            low = mid + 1;
        else
            high = mid;
    }
    return low;
}
int binarySearch (TYPE * data, int size, TYPE val){
    int low = 0;
    int high = size;
    while (low < high) {
        int mid = (low + high) / 2;
        if (data[mid] < val))
        {
            low = mid + 1;
        }
        else
        {
            high = mid;
        }
    }
    return low;
}
What does this Algorithm Return

• If value is found, returns its index
• If value is not found, returns index where it can be inserted without violating ordering
• Careful: returned index can be larger than the size of a collection
Makes which Bag operation faster?

• Suppose we use the dynamic array implementation of a bag

• Which operation is made faster by using a binary search?
  – Add(element)
  – Contains(element)
  – Remove(element)
An example operation

```c
int sortedContains (struct dynArr *da, TYPE val) {
    int idx = binarySearch(da->data, da->size, val);
    if (idx < da->size && da->data[idx] == val)
        return 1;
    return 0;
}
```
Add to a sorted Dynamic Array

```c
int sortedAdd (struct dynArr *da, TYPE val)
{
    int idx = binarySearch(da->data, da->size, val);
    _addAt(da, idx, val);
}
```
Add to a sorted Dynamic Array

```c
int sortedAdd (struct dynArr *da, TYPE val)
{
    int idx = binarySearch(da->data, da->size, val);
    _addAt(da, idx, val);
}
```

What is $O(??)$
int sortedRemove (struct dynArr *da, TYPE val) {
    int idx = binarySearch(da->data, da->size, val);
    if (idx < da->size && da->data[idx] == val)
        _removeAt(da, idx);
}
Remove

int sortedRemove (struct dynArr *da, TYPE val) {
    int idx = binarySearch(da->data, da->size, val);
    if (idx < da->size && da->data[idx] == val)
        _removeAt(da, idx);
}

What is O( ?? )
Why else do We Need an ordered Collection?

• Fast merge operations

• Fast set operations (special case)
  – union
  – intersection
Fast Merge

input 1: 5 9 10 12 17
input 2: 1 8 11 20 32
merge result:  

---

1. The diagram illustrates a fast merge process.
2. The two input lists are 5 9 10 12 17 and 1 8 11 20 32.
3. The merge result is an ordered list combining elements from both inputs.
4. The fast merge algorithm efficiently merges two sorted lists into one.

---

*Note: The merge result is not explicitly shown in the diagram, but it would be the sorted list 1 5 8 9 10 11 12 17 20 32.*
Fast Merge

input 1: 5 9 10 12 17

input 2: 1 8 11 20 32

merge result: 1
Fast Merge

input 1
5 9 10 12 17
8 9 10 12 17

input 2
1 8 11 20 32
8 11 20 32

merge result
1 5
Fast Merge

Input 1: 5 9 10 12 17
Input 2: 1 8 11 20 32
Merge Result: 1 5 8
Fast Merge

input 1  |  input 2  |  merge result
---------|-----------|-----------------|
5 9 10 12 17 | 1 8 11 20 32 | 1 5 8 9 10
Complexity of Merge

• What is $O(??)$
Set Operations

• Union is a special case of Merge

• Intersection is a special case of Merge

• Difference can be derived from Union and Intersection
Example: Intersection

Let \( i, j \) be index of two collections \( d, e \)

while \((i < d->\text{size} \, \&\& \, j < e->\text{size})\) {
    if \((d[i] < e[j])\) \(i++\);
    else if \((d[i] > e[j])\) \(j++\);
    else{ /* they are equal */
        add to intersection
        and advance both
        \(i++; \, j++;\)
    }
}
Example: Union (unique elements)

Let $i$, $j$ be index of two collections $d$, $e$

while $(i < d->size \&\& j < e->size)$ {
    if ($d[i] < e[j]$) {
        add $d[i]$ to union; $i$++;
    }
    else if ($d[i] > e[j]$) {
        add $e[j]$ to union; $j$++;
    }
    else {
        add $d[i]$ to union; $i$++; $j$++;
    }
}

add rest of $d$, and rest of $e$ to union;
Difference (D - E)

Let i, j be index of two collections d, e

while (i < d->size && j < e->size){
    if (d[i] < e[j]) {add d[i] to diff; i++;}
    else if (d[i] > e[j]) j++;
    else {i++; j++;}
}