CS1003: Intro to CS, Summer 2008

Lecture #05
Algorithms, Introduction to Data structures

Instructor: Arezu Moghadam
arezu@cs.columbia.edu

Agenda

- Another class of algorithms
  - Sort; insertion sort, bubble sort
  - Their time complexity
- Starting data structures
Sorting

- **Sorting takes an unordered collection and makes it an ordered one.**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>77</td>
<td>42</td>
<td>35</td>
<td>12</td>
<td>101</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

1 | 2 | 3 | 4 | 5 | 6 |
---|---|---|---|---|---|
5 | 12| 35| 42| 77| 101|

"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping
"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping

```
1  2  3  4  5  6
42 77 35 12 101 5
```

"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping

```
1  2  3  4  5  6
42 35 77 12 101 5
```
"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping

1 | 2 | 3 | 4 | 5 | 6
---|---|---|---|---|---
42 | 35 | 12 | 77 | 101 | 5

No need to swap
"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping

```
1          2          3          4            5            6
42 35 12 77 5 101
```

Largest value correctly placed
Items of Interest

- Notice that only the largest value is correctly placed
- All other values are still out of order
- So we need to repeat this process

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>35</td>
<td>12</td>
<td>77</td>
<td>5</td>
<td>101</td>
</tr>
</tbody>
</table>

Largest value correctly placed

Repeat “Bubble Up” How Many Times?

- If we have \( N \) elements...
- And if each time we bubble an element, we place it in its correct location...
- Then we repeat the “bubble up” process \( N - 1 \) times.
- This guarantees we’ll correctly place all \( N \) elements.
"Bubbling" All the Elements

N - 1

Bubble sort algorithm

for(i=length - 1; i > 0; i--) {
    for(j = 0; j < i; j++) {
        if(a[j] > a[j+1]) {
            int temp = a[j];
            a[j] = a[j+1];
            a[j+1] = temp;
        }
    }
}

\[ \Theta(n^2) \]
Data structures

- We’ve been referring to this informally, but now let’s be precise
- A computer’s memory is a large open space, and we can organize information in it
- A *data structure* is an organized entity in this memory space
- The most primitive data structures: *primitive types*

---

Primitive types

- int, char, double, etc.
- Occupy a well-known amount of memory
  - For 32-bit machines, an char takes 1 byte, an int takes 4 bytes, a double takes 8 bytes
  - *Not always the case*, but enough for this class
  - `sizeof(float);`
- The variable refers to that block of memory in its entirety
  - Can’t typically store decimal places inside an int; “won’t fit”
- But what if we want something more complicated?
Arrays

- Arbitrarily defined as a block of memory divided into cells
- To be more precise, an array is a static structure in memory
  - Memory is organized “contiguously” when you define an array
  - 10 integers => 10 * 4 => 40 bytes on a 32-bit machine
  - The variable referring to the array actually just points to the beginning of the appropriate memory location

Arrays (2)

- The programming language then does some math when you use [ ] to access an index in that array...
  - An array of integers, length 10 is at memory location “4000”.
  - How many bytes is this array in total?
  - What’s the position of the 5th integer?
  - Rationale for 0-based makes a little more sense
More generally…

- For primitive datatypes (int, char, etc.), the variable refers to that entity in its entirety.
- But whenever we work with a more complex data structure than just a primitive datatype, our variable will “point” to the beginning of the structure.
  - Known as a pointer in C.
- The programming language then decides what part of the memory starting at the variable you’re working with.

Strings

- Strings are an interesting case.
- In C, Strings are just arrays, and we treat them as blocks of memory of predefined size.
Heterogeneous Arrays

- Blocks of data items containing different types
- Items within blocks are called components
- Example; structures in C

```c
typedef struct {
    char name[64];
    char course[128];
    int age;
    int year;
} student;
```

Next time…

- More data structures