Data Structures

Giri Narasimhan
Office: ECS 254A
Phone: x-3748
giri@cs.fiu.edu

Course Website Correction

- Course Website: https://users.cs.fiu.edu/~giri/teach/**3530Fall16**.html

Time Complexities

- Sequence of Statements

  statement 1;

  statement 2;

  ...

  statement k;

- *total time* = sum of times for all statements:

T(n) = time(statement 1) + time(statement 2) + ... + time(statement k)

- If each statement is "simple" (only involves *basic* operations) then the time for each statement is constant and the total time is also *constant: O(1).*

Time Complexities … 2

- Loops
  - The running time of a loop is, at most, the running time of the statements inside the loop x the # of iterations

```
//executes n times
For i = 1 to n do
  m = m + 2;  // constant time
```

Total time T(n) = constant c x n = cn = O(n)

RECAP

Time Complexities … 3

- **Nested Loops**
  - Analyze from the *inside out*. Total running time is the product of the size of the loops

```
//outer loop executes n times
For i = 1 to n do
    //inner loop executes n times
    For i = 1 to n do
            k = j + 1;  // constant time
```

Total time $T(n) = c \times n \times n = cn^2 = O(n^2)$

Challenging Cases

MaxSubseqSum(*A*)

Initialize maxSum to 0

N := size(A)

For i = 1 to N do

  For j = i to N do

    Initialize thisSum to 0

    for k = i to j do

      add A[k] to thisSum

    if (thisSum > maxSum) then

      update maxSum

$$\sum_{k=i}^{j} 1 = j - i + 1$$

$$\sum_{j=i}^{N} (j - i + 1) = \frac{(N-i+1)(N-i+2)}{2}$$

$$\sum_{i=1}^{N} \frac{(N-i+1)(N-i+2)}{2}$$

$$= \sum_{i=1}^{N} \frac{i^2}{2} - (N + \frac{3}{2}) \sum_{i=1}^{N} i + \frac{1}{2}(N^2 + 3N + 2) \sum_{i=1}^{N} 1$$

$$= \frac{N^3 + 3N^2 + 2N}{6} = O(N^3)$$

RECAP

Challenging Case … 2

BINARYSEARCH(A, key, low, high)

If (low > high) return not found

mid = (low + high)/2

If A[mid] = key then return mid

If A[mid] > key then

    BinarySearch(A, key, low, mid-1)

Else

    BinarySearch(A, key, mid+1, high)

- On each recursive call, high-low+1 is halved
- How many times do you have to halve N before it becomes smaller than 1?
- Answer ≈ $\log_2 N$ Why?

Time Complexity

- Need: To provide information about time taken by an algorithm (or program)
- Obvious that time depends on size of input
- Idea: Write down T(n) = time taken by an algorithm as a function of n, size of input
- But time may vary for different inputs of same length
- Idea: Let T(n) = maximum time taken by an algorithm on any input of size n
    - Worst-case Time Complexity

Time Complexity

- Worst-case Time Complexity
  - T(n) = max time for an algorithm on any input of size n

Our main focus

- Best-case Time Complexity
  - B(n) = min time for an algorithm on any input of size n
- Average-case Time Complexity
  - A(n) = average time for an algorithm on inputs of size n

"Abstract" Data Structure

- **Abstract Data Type** (ADT); described by
  - Kind of data it stores
  - Operations performed on it (no implementations)
- **Data Structure**; consists of
  - data it stores
  - Operations performed on it with implementations
- **Example**: Priority queue is a "abstract" queue of entities each associated with a priority value.
  - Operations:
    - Insert entity with given priority
    - Delete item with highest priority
  - Java interface is an example of an ADT
  - Java class = ADT + Implementation is a data structure
  - List vs LinkedList or ArrayList

Standard ADTs

- List
- Stack
- Queue
- Tree
- Graph
- Set

- Basic operations in (most) ADTs
  - Insert
  - Delete
  - Search/Find/Member

Linear Data Structures

List

- A List deals with a "linear" list of entities of the form
  - $x_0, x_1, \ldots, x_{n-1}$
  - Each entity has a position: x_i has position i
  - Elements are all of same "type"
  - Many, many operations are possible :
    - Insert, insert at position, delete, delete from position, prev, next, find, printList, makeEmpty, isEmpty, size, sort, …
  - Lists can be implemented in one of 2 ways
    - Arrays or Linked lists
  - Arbitrary complex types are easily handled in practice using "generic" java class

Java's List interface

- Get(idx)
- Set(idx, value)
- Add(idx, value)
- Remove(idx)
- listIterator(pos)

Java's ArrayList

- Simple, "resizable" array implementation of a List
- Each item can be of a generic type
- Built on top of AbstractList, Collection, and Object
- Assumes list is Serializable, RandomAccess, Cloneable
- Large collection of operations available, including
  - Add(x) and Add(index, x)
  - Contains(x)
  - Remove(x), Remove(index)

(My)ArrayList Implementation

- See Figures 3.15 and 3.16 from Weiss text
- Maintains
  - list of items in an array called theItems[]
  - Array capacity (length of above array)
  - Current size called theSize
- Allows
  - Change in capacity (capacity doubled if array fills up)
    - No change upon removal
  - Implementation of get(idx) and set(idx,x)
  - Implementation of size(), isEmpty(), clear()
  - Implementation of Iterator interface
    - Index called current
    - next(), hasNext(), remove()

add and remove in ArrayList

- Both involve moving items
- Operation add(idx,x) involves moving all items from position idx onward to move in order to make space for x
- Operation remove(idx) involves moving all items from position idx+1 to close the gap created by the removal
- Study carefully how ArrayIterator is implemented

Java's LinkedList

- Simple, extendible, doubly-linked List implementation using pointers
- Each item can be of a generic type
- Assumes list is Serializable, ~~RandomAccess,~~ Cloneable
- Large collection of operations available, including
  - Add(x) and Add(index, x)
  - Contains(x)
  - Remove(x), Remove(index)

(My)LinkedList Implementation

- See Figures 3.24 -- 3.16 from Weiss text
- Maintains
  - Doubly linked list of Nodes of unlimited capacity
  - Pointer to extra 1st item (header node) called beginMarker and extra last item called endMarker
  - Node holds data and pointers to prev and next items
  - Current size called theSize
  - Extra entry called modCount used to help Iterator detect changes in collection
- Allows
  - Implementation of get(idx) and set(idx,x)
  - Implementation of size(), isEmpty(), clear()
  - Implementation of Iterator interface
    - Index called current
    - next(), hasNext(), remove()