CSC165 Week 8

Larry Zhang, October 28, 2014
an informal, anonymous and very useful survey of your learning experience

http://goo.gl/forms/AyC01bEk7g

Exclusive for Tuesday evening section
some “scary” announcements

→ A2 due next Monday, 10:00pm

→ term test 2 next Tuesday
  ◆ 6:10pm~7:00pm, in BA1130
  ◆ aid sheet: 8.5”x11”, double-sided, handwritten
  ◆ what’s in the test: three proofs

→ extended office hours:
  ◆ Thursday 4-6pm, BA4262
  ◆ Friday & Monday 4-6pm, BA5287

→ deadline for dropping: Nov. 3
some tips for Assignment 2
proof for statement with multiple quantifiers

$$\forall x \in X, \exists y \in Y, \forall z \in Z, \exists w \in W, P \Rightarrow Q$$

assume generic $x \in X$

pick some $y \in Y$

assume generic $z \in Z$

pick some $w \in W$

assume $P$

... 

then $Q$

then $\forall z \in Z, \exists w \in W, P \Rightarrow Q$

then $\exists y \in Y, \forall z \in Z, \exists w \in W, P \Rightarrow Q$

then $\forall x \in X, \exists y \in Y, \forall z \in Z, \exists w \in W, P \Rightarrow Q$
the floor function

"jump"
How to prove $a = b$?

➔ Prove $(a \leq b) \land (a \geq b)$

Assume two integers $a < b$

➔ $a \leq b - 1$  # two integers differ by at least 1

When the statement seems kind-of trivial but proving directly looks messy

➔ try contradiction
today’s outline

➔ formal definition of $O, \Omega$
➔ worst-case analyses of two algorithms
➔ problem solving session
formal definitions of $O$ and $\Omega$
recap \( O(n^2) \)

set of functions that **grow no faster** than \( n^2 \)

\[ \rightarrow \text{ count the number of steps} \]

\[ \rightarrow \text{ constant factors don’t matter} \]

\[ \rightarrow \text{ only highest-order term matter} \]

These functions are in \( O(n^2) \)

\[
\begin{align*}
n^2 & \quad 2n^2 + 3n \\
\frac{n^2}{165} + 1130n + 3.14159
\end{align*}
\]
the formal definition of $O(n^2)$

A function $f(n)$ is in $O(n^2)$ iff

$$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \text{ such that } \forall n \in \mathbb{N}, n \geq B \Rightarrow f(n) \leq cn^2$$

Beyond breakpoint $B$, $f(n)$ is upper-bounded by $cn^2$, where $c$ is some wisely chosen constant multiplier.
"chicken size" is in $O("turkey size")$

A chicken grows slower than a turkey in the sense that, after a certain breakpoint, a chicken will always be smaller than a turkey.
the formal definition of $O(n^2)$

A function $f(n)$ is in $O(n^2)$ iff

$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N},$ such that $\forall n \in \mathbb{N}, n \geq B \Rightarrow f(n) \leq cn^2$

**Simple example: prove** $700n^2 \in O(n^2)$

Pick $c = 711$, or any real number $\geq 700$

Pick $B = 0$, or any natural number $\geq 0$

then $\forall n \in \mathbb{N}, n \geq 0 \Rightarrow 700n^2 \leq 711n^2$

then $\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow 700n^2 \leq cn^2$

then $700n^2 \in O(n^2)$
the formal definition of $\Omega(n^2)$

a function $f(n)$ is in $O(n^2)$ iff

$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N},$ such that $\forall n \in \mathbb{N}, n \geq B \Rightarrow f(n) \leq cn^2$

a function $f(n)$ is in $\Omega(n^2)$ iff

$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N},$ such that $\forall n \in \mathbb{N}, n \geq B \Rightarrow f(n) \geq cn^2$

$O(n^2)$: set of functions that grow no faster than $n^2$

$\Omega(n^2)$: set of functions that grow no slower than $n^2$

$\Theta(n^2)$: set of functions that are in both $O(n^2)$ and $\Omega(n^2)$ (functions growing as fast as $n^2$)
examples

\[ 7n \in \mathcal{O}(n^2) \quad 7n \notin \Omega(n^2) \]

\[ 7n^3 \notin \mathcal{O}(n^2) \quad 7n^3 \in \Omega(n^2) \]

\[ 7n^2 \in \mathcal{O}(n^2) \quad 7n^2 \in \Omega(n^2) \]

\[ 7n^2 \in \Theta(n^2) \]
growth rate ranking of typical functions

\[ f(n) = n^n \]
\[ f(n) = 2^n \]
\[ f(n) = n^3 \]
\[ f(n) = n^2 \]
\[ f(n) = n \log n \]
\[ f(n) = n \]
\[ f(n) = \sqrt{n} \]
\[ f(n) = \log n \]
\[ f(n) = 1 \]

grow fast

grow slowly
analyse a sorting algorithm
insertion sort

➔ grow a sorted list inside an unsorted list
➔ in each iteration
  ◆ remove an element from the unsorted part
  ◆ insert it into the correct position in the sorted part
insertion sort

see animation at: http://en.wikipedia.org/wiki/Insertion_sort
insertion sort

def IS(A):
    '''sort the elements in A in non-decreasing order'''

1. i = 1
2. while i < len(A):
3.     t = A[i]  # take red square out
4.     j = i
5.     while j > 0 and A[j-1] > t:
7.         j = j - 1
8.     A[j] = t  # put red square in
9.     i = i + 1  # next element to be red-squared

j=i, ..., 1, in worst case that’s i iterations, +1 final loop guard, total lines to run: 3i + 1

each iteration has (3i + 1) + 5 lines to execute
insertion sort

```
def IS(A):
    '''sort the elements in A in non-decreasing order'''
    
    1. i = 1
    2. while i < len(A):
        3. t = A[i]  # take red square out
        4. j = i
        5. while j > 0 and A[j-1] > t:
            7. j = j - 1
        8. A[j] = t  # put red square in
        9. i = i + 1  # next element to be red-squared

    n: size of A
```

Each iteration has \((3i + 1) + 5\) lines to execute.
insertion sort worst-case running time

\[ W_{IS}(n) = 1 + 1 + \sum_{i=1}^{n-1} [(3i + 1) + 5] \]

\[ = 2 + \sum_{i=1}^{n-1} (3i + 6) = 2 + 6(n - 1) + 3 \sum_{i=1}^{n-1} i \]

\[ = 6n - 4 + 3 \cdot \frac{n(n - 1)}{2} \]

\[ = \frac{3}{2} n^2 + \frac{9}{2} n - 4 \]
Prove the worst case complexity of insertion sort is $O(n^2)$

$$W_{IS}(n) = \frac{3}{2}n^2 + \frac{9}{2}n - 4 \in O(n^2)$$

$$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow W_{IS}(n) \leq cn^2$$

**Proof:**

Pick $c = 11$ # pick a big c, could also be 2, 3, 4, 12, 35, 600, ...

Pick $B = 1$ # make sure $11n^2$ dominates for $n \geq B$

Assume $n \in \mathbb{N}$

Assume $n \geq 1$

then $\frac{3}{2}n^2 + \frac{9}{2}n - 4 \leq 11n^2$

then $n \geq B \Rightarrow W_{IS}(n) \leq cn^2$

then $\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow W_{IS}(n) \leq cn^2$
Prove \( W_{IS}(n) = \frac{3}{2}n^2 + \frac{9}{2}n - 4 \in \Omega(n^2) \)

\[ \exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow W_{IS}(n) \geq cn^2 \]

**Proof:**

Pick \( c = \frac{1}{2} \) \quad \# pick a small \( c \)

Pick \( B = 1 \) \quad \# make sure \((\frac{1}{2})n^2\) is smaller for \( n \geq B \)

Assume \( n \in \mathbb{N} \)

Assume \( n \geq 1 \)

then \( \frac{3}{2}n^2 + \frac{9}{2} - 4 \geq \frac{3}{2}n^2 + \frac{9}{2} \times 1 - 4 = \frac{3}{2}n^2 + \frac{1}{2} \)

\[ = \frac{1}{2}n^2 + n^2 + \frac{1}{2} \geq \frac{1}{2}n^2 \]

then \( n \geq B \Rightarrow W_{IS}(n) \geq cn^2 \)

then \( \exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow W_{IS}(n) \geq cn^2 \)
The worst case time complexity of insertion sort is in $O(n^2)$ and in $\Omega(n^2)$, i.e., it’s in $\Theta(n^2)$.
summary

➔ we first derived the exact form of $W_{is}(n)$, then determined it’s upper and lower bounds

➔ don’t alway have to derive the exact form, as we will see soon
analyse another algorithm
maximum slice

→ input: \( \mathbf{L} \), a list of numbers

→ output: the maximum sum over slices of \( \mathbf{L} \)

\[
\mathbf{L} = [-2, -3, 4, -1, 6, -3]
\]

\[
\text{max} = 4 + (-1) + 6 = 9
\]

useful algorithm for stockbrokers
def max_sum(L):
    ''' maximum sum over slices of L'''
    max = 0
    i = 0
    while i < len(L):
        j = i + 1
        while j <= len(L):
            sum = 0
            k = i
            while k < j:
                sum = sum + L[k]
                k = k + 1
            if sum > max:
                max = sum
            j = j + 1
        i = i + 1
    return max

How it works:
Enumerate all possible slices, compute the sum for each slice, and keep the max.
def max_sum(L):
    '''maximum sum over slices of L'''
    max = 0
    i = 0
    while i < len(L):
        j = i + 1
        while j <= len(L):
            sum = 0
            k = i
            while k < j:
                sum = sum + L[k]
                k = k + 1
            if sum > max:
                max = sum
            j = j + 1
        i = i + 1
    return max

\[ W_{MS}(n) \text{ is in } O(n^3) \]

\[
\begin{align*}
1. & \quad \text{max} = 0 \\
2. & \quad i = 0 \\
3. & \quad \textbf{while} i < \text{len}(L): \\
4. & \quad \quad j = i + 1 \\
5. & \quad \quad \textbf{while} j \leq \text{len}(L): \\
6. & \quad \quad \quad \text{sum} = 0 \\
7. & \quad \quad \quad k = i \\
8. & \quad \quad \quad \textbf{while} k < j: \\
9. & \quad \quad \quad \quad \text{sum} = \text{sum} + L[k] \\
10. & \quad \quad \quad \quad k = k + 1 \\
11. & \quad \quad \quad \textbf{if} \ \text{sum} > \text{max}: \\
12. & \quad \quad \quad \quad \text{max} = \text{sum} \\
13. & \quad \quad \quad j = j + 1 \\
14. & \quad \quad i = i + 1 \\
15. & \quad \quad \text{return max}
\end{align*}
\]

upper bound $W_{MS}(n)$

\[
\begin{align*}
\# \text{line 8-10, at most } n \text{ iters} \\
&\leq 3n + 1
\end{align*}
\]

\[
\begin{align*}
\# \text{line 5-13, at most } n \text{ iters} \\
&\leq n[(3n + 1) + 6] + 1 \\
&= 3n^2 + 7n + 1
\end{align*}
\]

\[
\begin{align*}
\# \text{line 3-14, at most } n \text{ iters} \\
&\leq n [(3n^2 + 7n + 1) + 3] + 1 \\
&= 3n^3 + 7n^2 + 4n + 1
\end{align*}
\]

\[
\begin{align*}
\# \text{line 1-15, add 3 more lines} \\
&\leq (3n^3+7n^2+4n+1)+3 = 3n^3+7n^2+4n+4
\end{align*}
\]
def max_sum(L):
    ''' maximum sum over slices of L'''

    1. max = 0
    2. i = 0
    3. while i < len(L):
        j = i + 1
        4. while j <= len(L):
            sum = 0
            k = i
            5. while k < j:
                sum = sum + L[k]
                k = k + 1
            if sum > max:
                6. max = sum
            j = j + 1
        i = i + 1
    return max

This analysis can be made more precise by considering \[ \left\lfloor \frac{n}{3} \right\rfloor \]

\( W_{MS}(n) \) is in \( \Omega(n^3) \)

lower bound \( W_{MS}(n) \)

Overall, it’s at least \( \frac{n}{3} \cdot \frac{n}{3} \cdot \frac{n}{3} = \frac{n^3}{27} \)
summary

→ we didn’t derive the exact form of \( W_{MS}(n) \), we did different calculations for upper and lower bounds

→ when finding upper-bound, it is OK to over-estimate the number of steps

→ when finding lower-bound, it is OK to under-estimate the number of steps
def max_sum(L):
    ''' maximum sum over slices of L'''
    max = 0
    i = 0
    while i < len(L):
        j = i + 1
        while j <= len(L):
            sum = 0
            k = i
            while k < j:
                sum = sum + L[k]
                k = k + 1
            if sum > max:
                max = sum
            j = j + 1
        i = i + 1
    return max

This \( O(n^3) \) algorithm is pretty stupid...

→ exercise for home: design a max_sum that runs in \( O(n^2) \)

→ challenge for home: design a max_sum that runs in \( O(n) \)

◆ think about computing the sum of [0:4] right after getting the sum of [0:3]
problem solving: penny piles