CS3000
5/8 - thurs :)

Admin

- HW1 out, due 5/15 4pm
- Funalgo recitation today! 1:30pm
- lecture Question

Agenda

1. correctness of insertion sort
2. Best case vs. worst case
3. Recursive sequences

0. Summation Reminders

$$\sum_{i=1}^{n} i = 1+2+3+4+\dots+n \quad \text{(by def)}$$

$$= \frac{(n)(n+1)}{2} \quad \text{(formula)}$$

$$= \Theta(n^2) \quad \text{(bound)}$$

$$\sum_{i=1}^{\infty} \left(\frac{1}{2}\right)^i = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} \dots$$

$$= 1$$

$$= \Theta(1)$$

1. correctness of insertion sort

↳ loop invariant: up to "next" element is sorted

↳ APP1, Q3: overwriting? swapping?

```
INSERTIONSORT(A, n)
for i = 2 to n
        key = A[i]
        j = i - 1
        while j > 0 and A[j] > key
                A[j + 1] = A[j]
                j = j - 1
        A[j + 1] = key
```

ex) 13, | 11, 4, 8, 7

key = 11
j = 1

$A[2] = A[1]$

13, 13, 4, 8, 7

j = 0
$A[1] = 11$

11, 13, | 4, 8, 7

key = 4
j = 2
$A[3] = A[j]$

11, 13, 13, 8, 7

loop invariant: true

- prior to first iteration
- if true before an iteration, then true before next iteration
- at termination

Assume: while loop works
prove for loop

loop invariant: A[1..i-1] is sorted
↳ show A is sorted at the end

A[j+1] = A[j]

11, 11, 13, 8, 7

4, 11, 13, 8, 7

$\boxed{\text{Proof}}$

① initialization
- before first iteration begins, i=2
- A[1.. i-1] = A[1] is trivially sorted

② maintenance
- assume A[1...i-1] sorted before iteration begins
- move elements A[i-1], A[i-2], ... by one position to the right
- until we find correct sorted position for A[i]
- at line 7, A[i] is moved to its sorted place, so A[1..i] is sorted
- incrementing i preserves the loop invariant

③ termination (useful property that shows overall correctness)
- when loop terminates, i=n+1
- therefore A[1..n] is sorted, so our algo works! :)

2. Best-case vs. worst case

Best-case array
- already sorted!
- $\Theta(n)$

Worst-case array
- reverse sorted :(
- ?

T(n) ⟶ complexity class

	INSERTIONSORT(A, n)	cost	# times
1	**for** $i = 2$ **to** n	c_1	n
2	$key = A[i]$	c_2	$n-1$
3	$j = i - 1$	c_3	$n-1$
4	**while** $j > 0$ **and** $A[j] > key$	c_4	
5	$A[j+1] = A[j]$	c_5	☆
6	$j = j - 1$	c_6	
7	$A[j+1] = key$	c_7	$n-1$

$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_7(n-1)$

$= c_1 \cdot n + c_2 \cdot n - c_2 + c_3 \cdot n - c_3 + c_7 \cdot n - c_7$

$= (c_1 + c_2 + c_3 + c_7) \cdot n - (c_2 + c_3 + c_7) + \boxed{\cancel{\;}}$

worst case example: 9, 7, 6, 4

How many times does $\boxed{\text{line 4}}$ run? $\longrightarrow$ i times for $i = 2, 3, 4, \ldots, n$

- $A[j] \geq key$ always true
- $j > 0$ determines the answer

ex

$i=2$
$j=1$ $\quad 1>0?$
$0>0?$
$\boxed{2 \text{ times}}$

$i=3$
$j=2$ $\quad 2>0?$
$1>0?$
$0>0?$
$\boxed{3 \text{ times}}$

$i=4$
$j=3$ $\quad 3>0?$
$2>0?$
$1>0?$
$0>0?$
$\boxed{4 \text{ times}}$

$\hookrightarrow$ How many times altogether?

$$\sum_{i=2}^{n} i = \frac{(n)(n+1)}{2} - 1 \quad \text{(line 4)} \quad \rightsquigarrow \frac{n^2}{2} + \frac{n}{2} - 1$$

$$\sum_{i=2}^{n} (i-1) = \frac{(n)(n-1)}{2} \quad \text{(lines 5, 6)} \quad \rightsquigarrow \frac{n^2}{2} - \frac{n}{2}$$

$T(n)$ = what we had so far, plus...

$$c_4\left(\frac{n^2}{2} + \frac{n}{2} - 1\right) + c_5\left(\frac{n^2}{2} - \frac{n}{2}\right) + c_6\left(\frac{n^2}{2} - \frac{n}{2}\right)$$

$$= \frac{c_4}{2} n^2 + \frac{c_4}{2} n - c_4 + \frac{c_5}{2} n^2 - \frac{c_5}{2} n + \frac{c_6}{2} n^2 - \frac{c_6}{2} \cdot n$$

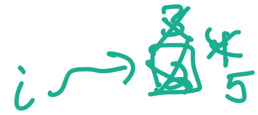
$$= \left(\frac{c_4 + c_5 + c_6}{2}\right) \cdot n^2 + \left(\frac{c_4 - c_5 - c_6}{2}\right) n - c_4$$

$\hookrightarrow \Theta(n^2)$

12:43

2.5 Insertion sort

Best case run time $\Theta(n)$
Worst case run time $\Theta(n^2)$ } what about space?




```
INSERTIONSORT(A, n)
for i = 2 to n
    key = A[i]
    j = i - 1
    while j > 0 and A[j] > key
        A[j + 1] = A[j]
        j = j - 1
    A[j + 1] = key
```

Focus:

- variables created during the algorithm
- single variable takes up constant space
- variables are destroyed at end of block

var	space	# at once
i	c_1	1
key	c_2	1
j	c_3	1

Space complexity $\Theta(1$

3. Recursive Sequences

Sequence:

- ordered list (possibly infinite) of numbers
- a function on the positive integers (positions)

$\overset{2}{a_1}$ $\overset{4}{a_2}$ $\overset{6}{a_3}$ . . .

$a_n = 2 \cdot n$!! :)

$a_{500} = 1000$

recursively defined:

- a_n is defined by previous terms
- to compute a_n, we need a_{n-1}
  to compute a_{n-1} we need a_{n-2} . . . !! :(

often: recursive $\longrightarrow$ closed form !! :)

iteration method

- define first few terms by the base case

• establish a pattern for a_n for arbitrary n

ex recursively defined

$$\boxed{\begin{array}{l} a_n = 2 \cdot a_{n-1} + 1 \\ \text{base case: } a_1 = 1 \end{array}}$$

$a_1 = 1$

$a_2 = 2 \cdot a_1 + 1$

$a_3 = 2 \cdot a_2 + 1$

$= 2 \cdot (2 \cdot a_1 + 1) + 1$

$= 4a_1 + 3$

$a_4 = 2 \cdot a_3 + 1$

$= 2(4a_1 + 3) + 1$

$= 8a_1 + 7$

$a_5 = 2 \cdot a_4 + 1$

$= 2(8a_1 + 7) + 1$

$= 16a_1 + 15$

pattern? in terms of n

$a_n = 2^{n-1} \cdot a_1 + 2^{n-1} - 1$

$= 2^{n-1} + 2^{n-1} - 1$

$= 2 \cdot 2^{n-1} - 1$

$= 2^n - 1$