CS3000
5/12 - mon.

Admin

- HW1 due thurs 9pm
- APP2 today, due 11:30 5/13
- Recitation 2 5/13

Agenda

1. Recursive algorithms
2. Recursive run-times T(n)
3. Solve the recurrence!
4. APP2

0. logarithm reminders

$\lg 2^{125} = 125$

$\lg 2^3 = \lg 8 = 3$

$\lg 2^k = k$

$n/2^k = 1$ — solve for k

$n = 2^k$

$\lg n = \lg 2^k$ — log both sides

$\lg n = k$

1. Recursive Algorithms

"Good" algorithm – correct
– efficient

correct ☑
can we do better?

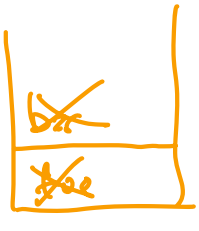
strategy #1 — divide and conquer (uses recursion)

Functions when they run (in general)

```
foo(...)
  bar()
  return

bar()
  return
```

lifespan



variables take up space, then are destroyed when function is over

Recursion

- recursive sequence: $a_n = a_{n-1} + 2n$, base case $a_1 = 2$
- recursive function: function that calls itself

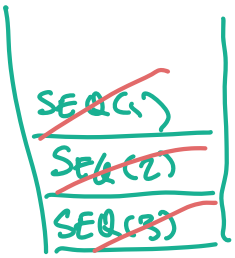
```
SEQ(n)
1  if n == 1          > base case
2      return 2
3  return SEQ(n-1) + 2·n   > recursive call
```

??

12 ↳ n gets smaller!

$z_3 = z_2 + 6 = 12$
$z_2 = z_1 + 4 = 6$
$z_1 = 2$

(ex) SEQ(3)
↳ return SEQ(2) + 6 → 6
↳ return SEQ(1) + 4 → 2
↳ 2



Variables take up space all at once, are destroyed after returning

Strategy #1: Divide + Conquer
- uses recursion
- break the problem into smaller subproblems
- solve the subproblems recursively
- combine smaller solutions into one big solution

Last week: Linear Search
- given A, n, key
- return position i where key is found
- or Nil if not found

$\Theta(n)$ w.c.
Can we do better?
↳ if the array is already sorted

Binary Search (Array is sorted)

(ex) 1 5 8 9 10 11 12 13
① ⑧

key = 8

↳ midpoint

(throw away right)

① ③
↳ 1 5 8

(throw away left)

↳ midpoint

↳ 8 found it!
③

Is it worth it to sort first?

Binary search

- input: A, low, high, key)
- return position if found
- return nil if not found
- subproblem: search for key in left/right half of the array } recursive case

first call: A, 1, n, key

- don't make new arrays!
- A doesn't change

Pseudocode

- what is base case?
  low vs. high
- calculate the midpoint?
- what if we find key?
- what if key > midpoint or key < midpoint?

recursion →

low == high subarray length 1
low > high subarray length 0

mid = ⌊(low + high)/2⌋

return position where found (mid)

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

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

12:43

2. Rewsive Run Times

&& and
|| or
not

```
BINARYSEARCH(A, low, high, key)
1  if low > high                          } base case
2      return NIL                         }
3  mid = ⌊(low + high)/2⌋
4  if key == A[mid]                       } found it!
5      return mid                         }
6  elseif key > A[mid]                    → right half
7      return BINARYSEARCH(A, mid + 1, high, key)
8  return BINARYSEARCH(A, low, mid-1, key)
                                          ↳ left half
```

T(n) = # steps algo requires on input of length n

recurrence

- T(n) is expressed in terms of run-times on subproblems
- base case

```
BINARYSEARCH(A, low, high, key)                                  cost
1  if low > high                                                  c1
2      return NIL                                                 c2
3  mid = ⌊(low + high)/2⌋                                         c3
4  if key == A[mid]                                               c4
5      return mid                                                 c5
6  elseif key > A[mid]                                            c6
7      return BINARYSEARCH(A, mid + 1, high, key)                 c7
8  return BINARYSEARCH(A, low, mid-1, key)                        c8
```

base case: $T(0) = c_1 + c_2$

recurrence:

$T(n) = c_1 + c_3 + c_4 + c_6 + T(n/2)$

↳ no way to bound this!! :)

so, we need to solve the recurrence

3. Solving Recurrences

- $T(n)$ run-time that's a recurrence
- solve to get it in closed form (no $T(n)$ in the $T(n)$)
- bound it!

iteration method

- express $T(n), T(n/2), T(n/4) \ldots$ until we see a pattern
- define pattern for k^{th} iteration
- choose value for k that leads to base case

(ex) binary search run time

$$T(n) = T(n/2) + c$$

$$T(1) = c' \quad \text{(base case)}$$

(1) $T(n) = T(n/2) + c$

↳ $T(n/2) = T(n/4) + c$

(2) $T(n) = T(n/4) + c + c$

$= T(n/4) + 2c$

↳ $T(n/4) = T(n/8) + c$

k^{th} iteration

$T(n) = T(n/[?]) + [?]k$

$= T(n/2^k) + kc$

③ $T(n) = T(n/8) + c + 2c$

$= T(n/8) + 3c$

...

choose a value of k that gets us to base case!

$T(1)$ is base case

choose k such that

$n/2^k = 1$

$n = 2^k$

$\lg n = k$!!:)

$T(n) = T(n/2^k) + k \cdot c$

let $k = \lg n$, plug in

$T(n) = T(n/2^{\lg n}) + k \cdot c$

$= T(n/n) + k \cdot c$

$= T(1) + k \cdot c$

$= c' + \lg n \cdot c$

$\Theta(\lg n)$!!:)

$T(n) = 2 \cdot T(n/2) + n + c$ ①

$T(n/2) = 2 \cdot T(n/4) + n/2 + c$

$T(n) = 2 \cdot [2 \cdot T(n/4) + n/2 + c] + n + c$ ②

$= 4 \cdot T(n/4) + n + 2c + n + c$

$= 4 \cdot T(n/4) + 2n + 3c$