CS4102 Algorithms

Nate Brunelle

Spring 2018

Warm up

Show that finding the minimum of an unordered list requires $\Omega(n)$ comparisons

Find Min, Lower Bound Proof

Show that finding the minimum of an unordered list requires $\Omega(n)$ comparisons

Suppose (toward contradiction) that there is an algorithm for Find Min that does fewer than $\frac{n}{2} = \Omega(n)$ comparisons.

This means there is at least one “uncompared” element
We can’t know that this element wasn’t the min!

2	8	19	20	[redacted]	3	9	-4
0	1	2	3	4	5	6	7

Today’s Keywords

- Sorting
- Linear time Sorting
- Counting Sort
- Radix Sort

CLRS Readings

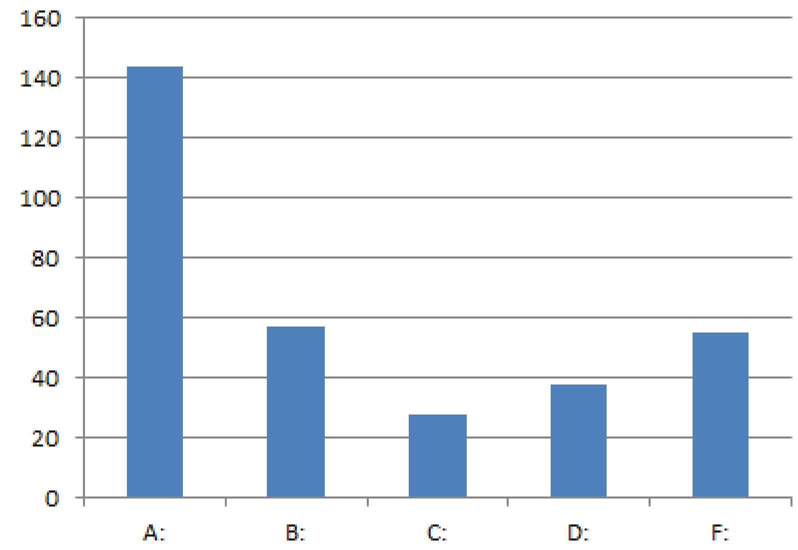
- Chapter 8

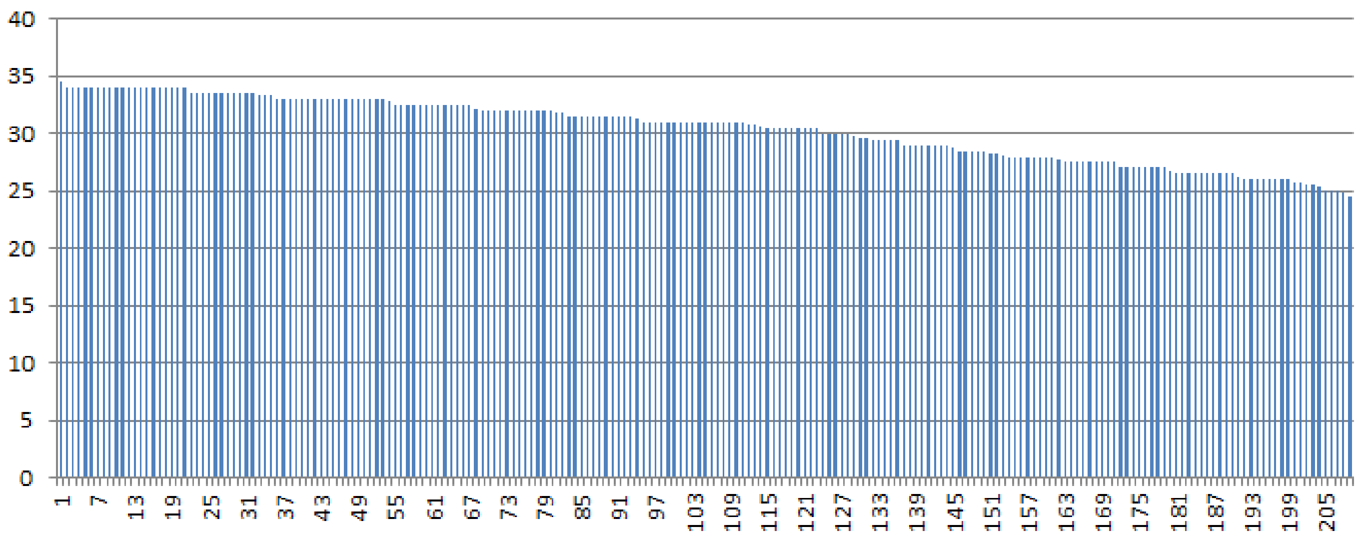
Homeworks

- Hw3 Due 9am Saturday Sept. 30
  - Divide and conquer
  - Written (use LaTeX!)
- Hw4 released Thursday Sept. 28
  - Sorting
  - Written

HW1 Graded

- Average: 28.2/32
  - 88%
- Median: 29.5/32
  - 92%





Midterm

- Monday March 19 in class
  - Covers all content through sorting
  - We will have a review session the weekend before

Sorting in Linear Time

- Cannot be comparison-based
- Need to make some sort of assumption about the contents of the list
  - Small number of unique values
  - Small range of values
  - Etc.

Counting Sort

- Idea: Count how many things are less than each element

$L =$

3	6	6	1	3	4	1	6
1	2	3	4	5	6	7	8

1. Range is $[1, k]$ (here $[1,6]$)
   make an array C of size $k + 1$
   populate with counts of each value

> For i in L:
> $++C[L[i]]$

$C =$

2	0	2	1	0	3
1	2	3	4	5	6

running sum

2. Take "running sum" of C
   to count things less than each value

> For $i = 1$ to $\mathrm{len}(C)$:
> $C[i] = C[i-1] + C[i]$

$C =$

2	2	4	5	5	8
1	2	3	4	5	6

To sort: last item of value 3 goes at index 4

Counting Sort

- Idea: Count how many things are less than each element

$L =$

3	6	6	1	3	4	1	6
1	2	3	4	5	6	7	8

$C =$

2	2	4	5	5	7
1	2	3	4	5	6

Last item of value 6 goes at index 8

For each element of L (last to first):
Use C to find its proper place in B
Decrement that position of C

For $i = 1$ to $\text{len}(L)$:

$$B\left[C\left[L[i]\right]\right] = A[i]$$

$$C\left[A[i]\right] = C\left[A[i]\right] - 1$$

$B =$

							6
1	2	3	4	5	6	7	8

Counting Sort

- Idea: Count how many things are less than each element

$L =$

3	6	6	1	3	4	1	6
1	2	3	4	5	6	7	8

$C =$

1	2	3	5	5	8
1	2	3	4	5	6

Last item of value 1 goes at index 2

For each element of L(last to first):
Use C to find its proper place in B
Decrement that position of C

For $i = 1$ to $\text{len}(L)$:

$$B\left[C\left[L[i]\right]\right] = A[i]$$

$$C\left[A[i]\right] = C\left[A[i]\right] - 1$$

$B =$

	1						6
1	2	3	4	5	6	7	8

Run Time: $O(n + k)$

Memory: $O(n + k)$

Counting Sort

- Why not always use counting sort?
- For 64-bit numbers, requires an array of length $2^{64} > 10^{19}$
  - 5 GHz CPU will require > 116 years to initialize the array
  - 18 Exabytes of data
    - Total amount of data that Google has

12 Exabytes

Radix Sort

- Idea: Stable sort on each digit, from least significant to most significant

103	801	401	323	255	823	999	101	113	901	555	512	245	800	018	121
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Place each element into a "bucket" according to its 1's place

800	801 401 101 901 121	512	103 323 823 113		255 555 245			018	999
0	1	2	3	4	5	6	7	8	9

Radix Sort

- Idea: Stable sort on each digit, from least significant to most significant

Place each element into a "bucket" according to its 10's place

0	1	2	3	4	5	6	7	8	9
800	801 401 101 901 121	512	103 323 823 113		255 555 245			018	999

0	1	2	3	4	5	6	7	8	9
800 801 401 101 901 103	512 113 018	121 323 823		245	255 555				999

Radix Sort

- Idea: Stable sort on each digit, from least significant to most significant

0	1	2	3	4	5	6	7	8	9
800 801 401 101 901 103	512 113 018	121 323 823		245	255 555				999

Place each element into a "bucket" according to its 100's place

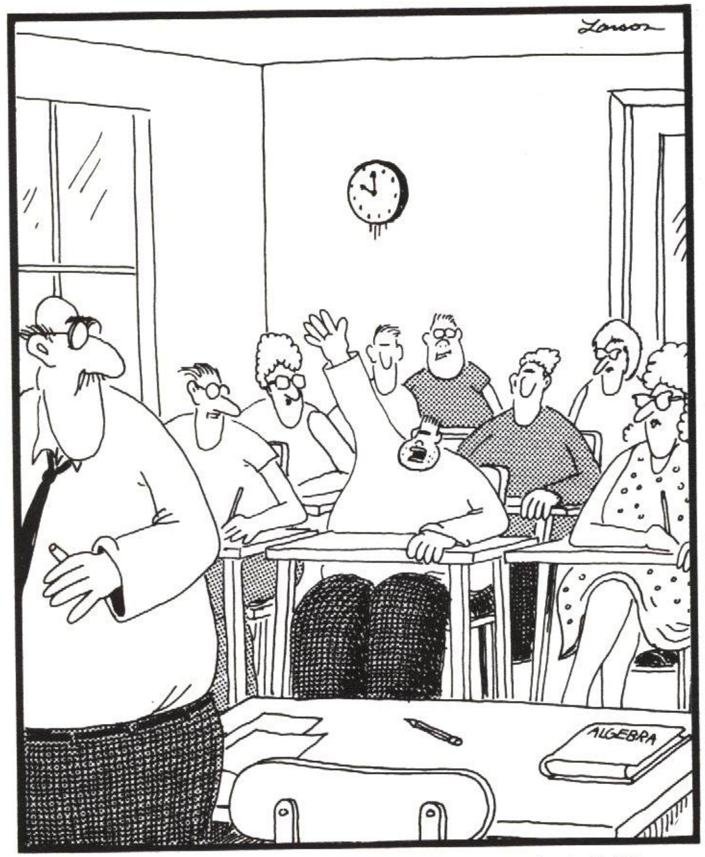
0	1	2	3	4	5	6	7	8	9
018	101 103 113 121	245 255	323	401	512 555			800 801 823	901 999

Run Time: $O\big(d(n+b)\big)$

d = digits in largest value

b = base of representation

End of Midterm Exam Materials!



"Mr. Osborne, may I be excused? My brain is full."

CS4102 Algorithms

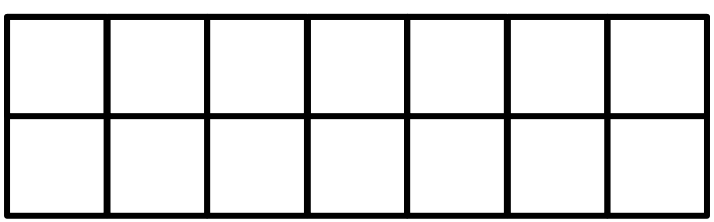
Nate Brunelle

Spring 2018

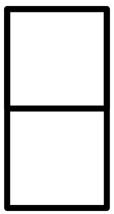
Warm up

How many ways are there to tile a $2 \times n$ board with dominoes?

How many ways to tile this:



With these?



Today’s Keywords

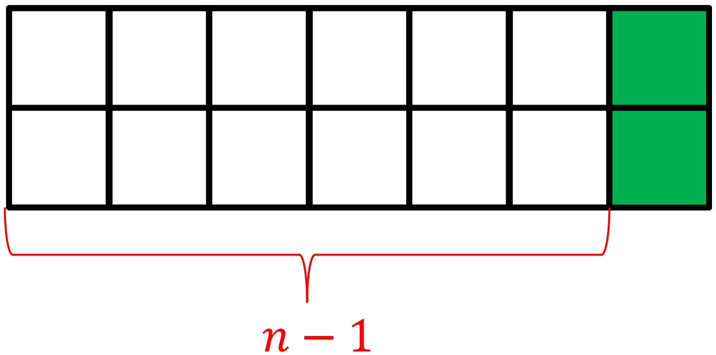
- Dynamic Programming
- Log Cutting

CLRS Readings

- Chapter 15

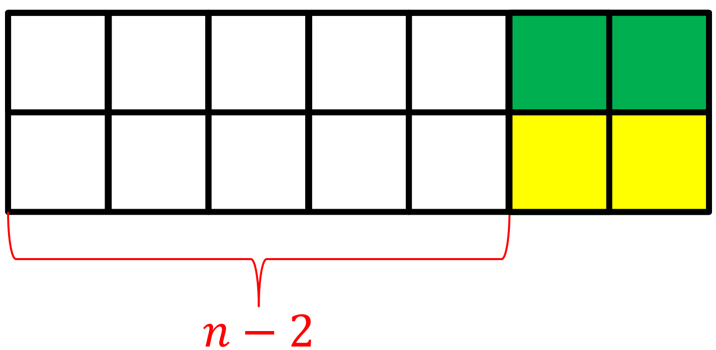
How many ways are there to tile a $2 \times n$ board with dominoes?

Two ways to fill the final column:



$$Tile(n) = Tile(n-1) + Tile(n-2)$$

$$Tile(0) = Tile(1) = 1$$

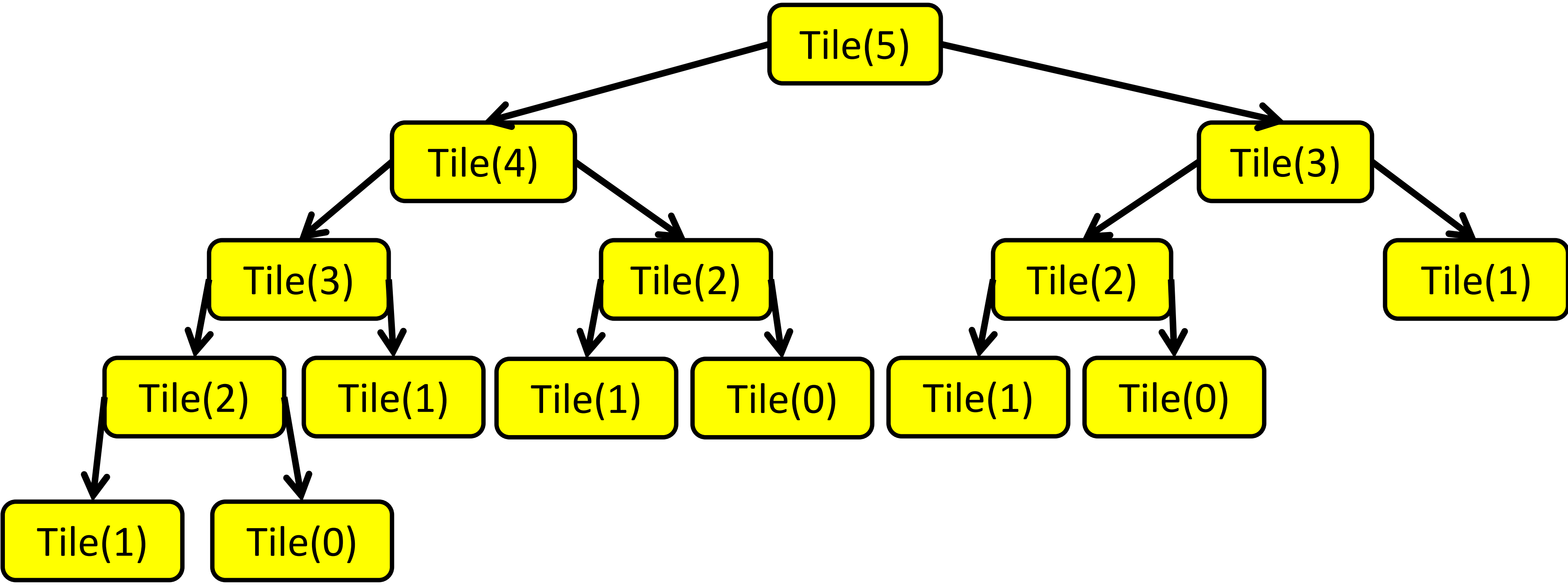


How to compute $Tile(n)$?

```
Tile(n):
        if n < 2:
                return 1
        return Tile(n-1)+Tile(n-2)
```

Problem?

Recursion Tree



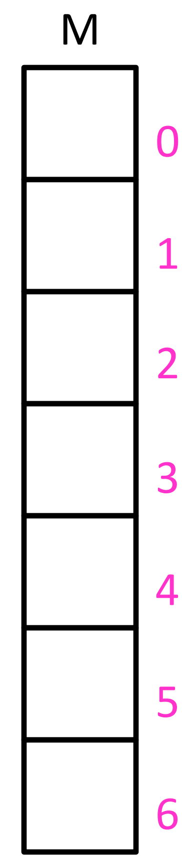
Many redundant calls!

Run time: $\Omega(2^n)$

Better way: Use Memory!

Computing $Tile(n)$ with Memory

```
Initialize Memory M
Tile(n):
      if n < 2:
            return 0
      if M[n] is filled:
            return M[n]
      M[n] = Tile(n-1)+Tile(n-2)
      return M[n]
```



Computing $Tile(n)$ with Memory "Top Down"

```
Initialize Memory M
Tile(n):
      if n < 2:
            return 1
      if M[n] is filled:
            return M[n]
      M[n] = Tile(n-1)+Tile(n-2)
      return M[n]
```

M

M	
1	0
1	1
2	2
3	3
5	4
8	5
13	6

Recursive calls happen in a predictable order

Better $Tile(n)$ with Memory "Bottom Up"

Tile(n):

Initialize Memory M

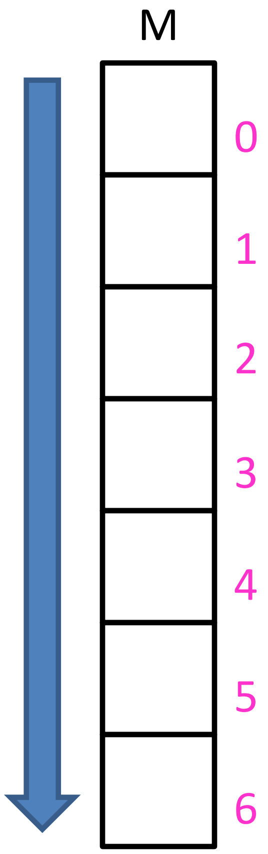
M[0] = 1

M[1] = 1

for i = 2 to n:

M[i] = M[i-1] + M[i-2]

return M[n]



Dynamic Programming

- Requires Optimal Substructure
  - Solution to larger problem contains the solutions to smaller ones
- Idea:
  1. Identify recursive structure of the problem
  2. Select a good order for solving subproblems
     - Usually smallest problem first