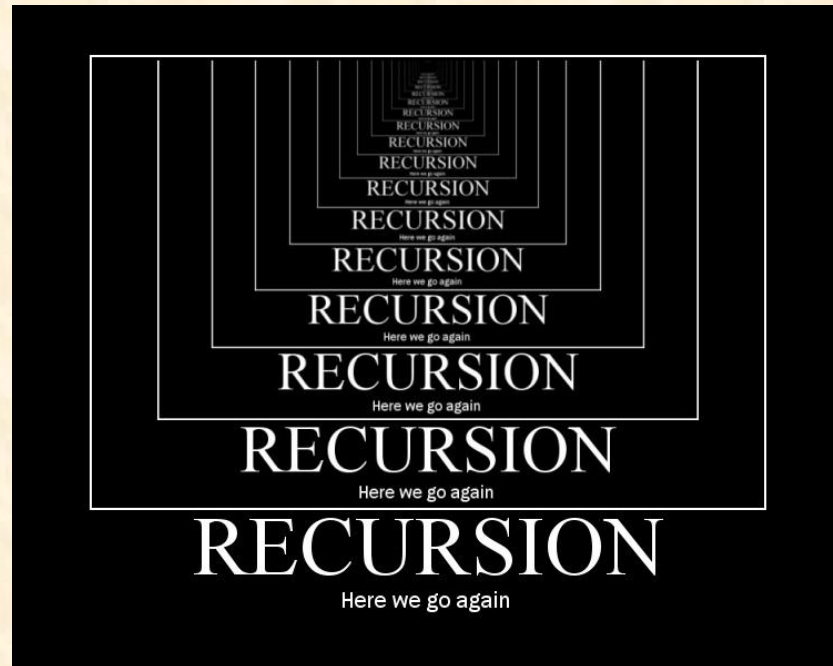


Lecture 4

Recursion



EECS 281: Data Structures & Algorithms

Administrative

- P1 is due 9/27 by midnight
- Remember:
We want to push you. Not fail you.

Project 1: Tips and FAQ

My code is running too slow...what do I do?

- First, make sure you're compiling with `-O3`.
 - Our Makefile does this by default
- If you make a change, you can test speed yourself with `time`.
 - Don't waste a submit!
- Review the "Project 1: The STL and You" slides we released.
 - `ios_base::sync_with_stdio(false);`
 - Use `'\n'` instead of `endl`, unless you actually need to flush output.
 - Use the right data structure; a linked list could function as both a queue and a stack for our application, but a deque is almost certainly faster

Project 1: Tips and FAQ

My code is running too slow...what do I do?

- Other Possibilities:
 - Make sure any functions that get called frequently are being inlined. If you have class templates across multiple files, you may need to put implementations in one file
 - virtual function calls are slower than others; get rid of these
 - Use a 1D vector to store the map, it's faster than 3D
- These are almost certainly a waste of time, UNLESS you have used a profiler to check which parts of your code take the most time
- They're also likely to detract from code quality and clarity. Don't make this sacrifice unless you're really sure you need to!

Project 1: Tips and FAQ

My code is running too slow...what do I do?

- Use a profiler! (For example, `perf`.)

Here's how I found out where my¹ own P1 code was too slow.

- On the AG, I was slow on the big tests – particularly in list input mode.
- On CAEN, make sure to run `module load gcc/6.2.0`
- Compile on CAEN with `make` (this does include the `-O3` flag).
- Run the code through `perf` with a command like:

```
perf record --call-graph dwarf ./ship -q -o M <
Sample-Big-L.txt
```
- This creates a file `perf.data`. Analyze the results stored in this file with:

```
perf report
```

```
- main
+ 87.82% Ship<MapReader_standard, ListReader_stringstream>::createFromFile
+ 5.99% Ship<MapReader_standard, ListReader_stringstream>::printMap
  2.73% SearchAlgorithm<QueueSearch<Ship<MapReader_standard, ListReader_str
```

Project 1: Tips and FAQ

I'm going over memory...what do I do?

- Use a memory profiler! (For example, Valgrind's `massif` tool.)
 - On CAEN, make sure to run `module load gcc/6.2.0`
 - Compile on CAEN with `make` (this does include the `-O3` flag).
 - Run the code through `massif` with a command like:

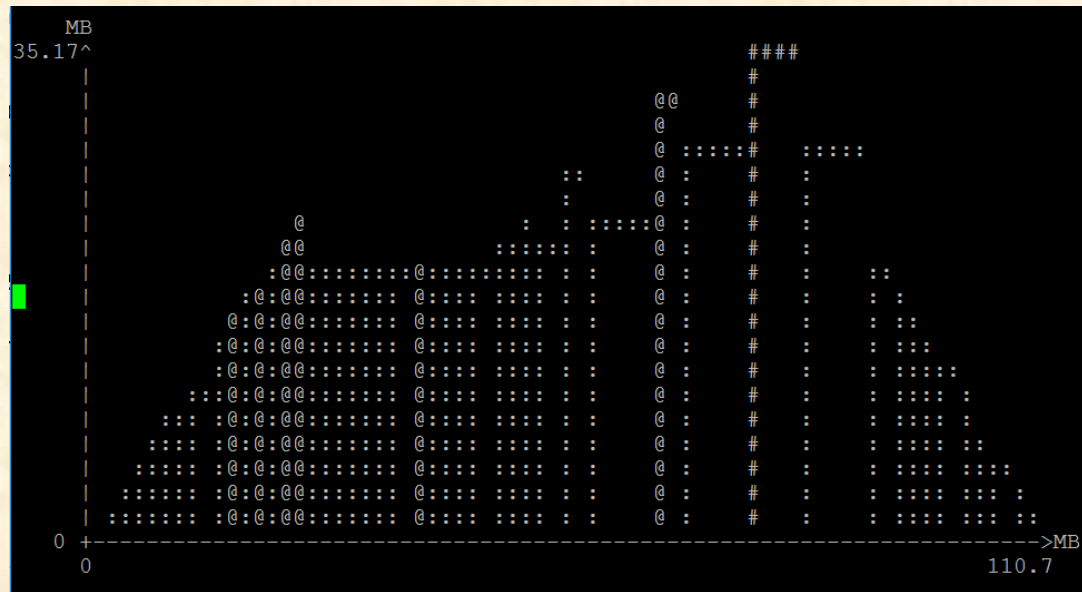
```
valgrind --tool=massif --massif-out-file=massif.out  
--time-unit=B ./ship -q -o M < Sample-Big-L.txt
```
 - This creates a data file called `massif.out`.
 - To generate a report, use the `ms_print` command and pipe to `less`.

```
ms_print massif.out | less
```
 - (To exit `less`, hit `q`.)

Project 1: Tips and FAQ

I'm going over memory...what do I do?

- Use a memory profiler! (For example, Valgrind's `massif` tool.)
- Here's the results for my P1 code (which is fine on memory).
 - A graph of memory use over the course of the program:



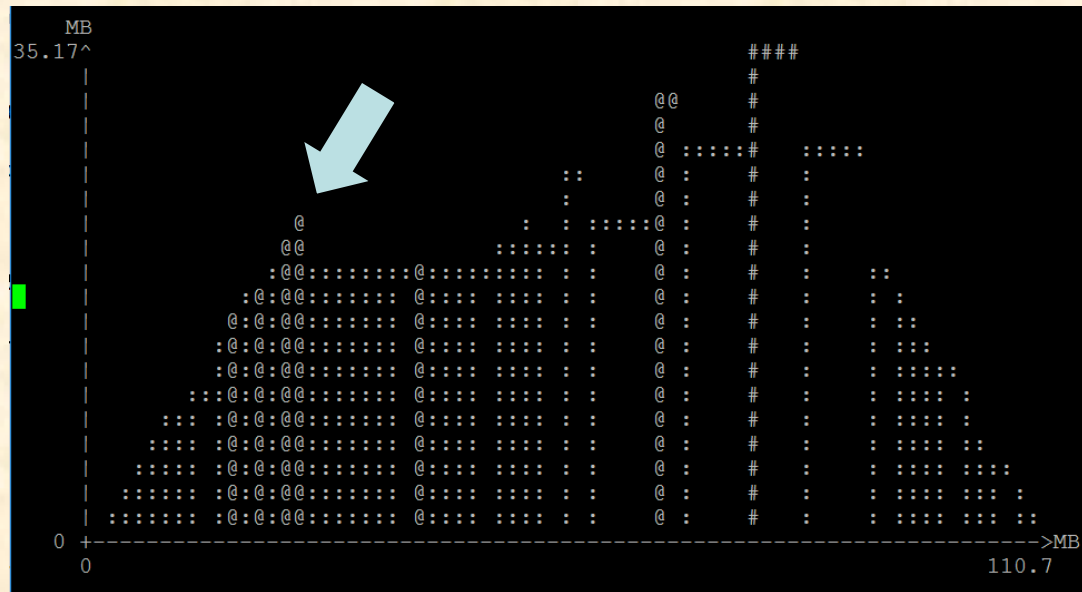
- Several snapshots of all large objects at different times:

```
99.47% (21,402,816B) (heap allocation functions) malloc/new/new[], --alloc-fns, etc.
->97.60% (21,000,000B) 0x402D3D: Ship<MapReader_standard, ListReader_stringstream>::c
| ->97.60% (21,000,000B) 0x401DE4: main (in /home/jjuett/eecs281sp17/p1/ship)
|
->01.87% (402,816B) in 13 places, all below massif's threshold (1.00%)
```

Project 1: Tips and FAQ

I'm going over memory...what do I do?

- Use a memory profiler! (For example, Valgrind's `massif` tool.)
- Here's the results for my P1 code (which is fine on memory).
 - A graph of memory use over the course of the program:



This is my
dictionary

- several snapshots of all large objects at different times:

```
99.47% (21,000,000B) (heap allocation functions) malloc/new/new[], --alloc-fns, etc.
->97.60% (21,000,000B) 0x402D3D: Ship<MapReader_standard, ListReader_stringstream>::c
| ->97.60% (21,000,000B) 0x401DE4: main (in /home/jjuett/eecs281sp17/p1/ship)
|
->01.87% (402,816B) in 13 places, all below massif's threshold (1.00%)
```


Project 1: Tips and FAQ

I'm going over memory...what do I do?

- Look for unnecessary copies of large objects! In this version, I added a bunch of these and ran against `massif` again.



Three dictionaries?!

```
99.47% (63,700,088B) (heap allocation functions) malloc/new/new[], --alloc-fns, etc.
->32.79% (21,000,000B) 0x4056FD: Ship<MapReader_standard, ListReader_stringstream>::c
->32.79% (21,000,000B) 0x401DE1: main (in /home/jjuett/eecs281sp17/p1/ship)
->32.79% (21,000,000B) 0x4028CD: std::vector<std::vector<std::vector<Ship<MapReader_s
->32.79% (21,000,000B) 0x401F94: main (in /home/jjuett/eecs281sp17/p1/ship)
->32.79% (21,000,000B) 0x40338D: QueueSearch<Ship<MapReader_standard, ListReader_stri
->32.79% (21,000,000B) 0x401FAC: main (in /home/jjuett/eecs281sp17/p1/ship)
->01.09% (700,088B) in 16 places, all below massif's threshold (1.00%)
```

Project 1: Tips and FAQ

Q: I'm going over memory...what do I do?

A: Think about the problem and gather data.

- Given the results from massif, what is more important to optimize in terms of space?
 - Structs contained in my dictionary.
 - Structs contained in my queue/stack/deque.
- Consider testing with a line to print out the size of your queue/stack on each iteration as a sanity check.
 - How large is the queue/stack compared to the dictionary?

→ For a 1000x1000 map in queue mode, the max size was ~4000

Project 1: Tips and FAQ

Q: I'm going over memory...what do I do?

A: Worry about the data structures that matter most!

- Don't worry about single variables
 - Loop variables
 - Single variables inside `main()`
- Worry about `vector<Data> data;`
 - What's in the Data object?
 - Is it organized efficiently?
 - Did you use `reserve/resize` wisely?

Recall: Job Interview Question

- Implement this function

// returns x^n

```
int power(int x, unsigned int n);
```

- The obvious solution uses $n - 1$ multiplications
 - $2^8 = 2 * 2 * \dots * 2$
- Less obvious: $O(\log n)$ multiplications
 - Hint: $2^8 = ((2^2)^2)^2$
 - Does it work for 2^7 ?
- Write two solutions

Idea

- Previously, we used a subproblem "one step smaller".

$$x^n = \begin{cases} 1 & n == 0 \\ x * x^{n-1} & n > 0 \end{cases}$$

- Now, let's try splitting the problem into two "halves".

$$x^n = \begin{cases} 1 & n == 0 \\ x^{n/2} * x^{n/2} & n > 0, \text{even} \\ x * x^{\lfloor n/2 \rfloor} * x^{\lfloor n/2 \rfloor} & n > 0, \text{odd} \end{cases}$$

Revisiting Recursion

Recursive #1

```
int power(int x, int n) {  
    if (n == 0) {  
        return 1;  
    }  
    return x * power(x, n - 1);  
}
```

Recurrence: $T(n) = T(n-1) + c$
Complexity: $\Theta(n)$

Recursive #2

```
int power(int x, int n) {  
    if (n == 0) {  
        return 1;  
    }  
    int result = power(x, n / 2);  
    result *= result;  
    if (n % 2 != 0) { // x is odd  
        result *= x;  
    }  
    return result;  
}
```

Recurrence: $T(n) = T(n / 2) + c$
Complexity: $\Theta(\log n)$

Solving Recurrences Example

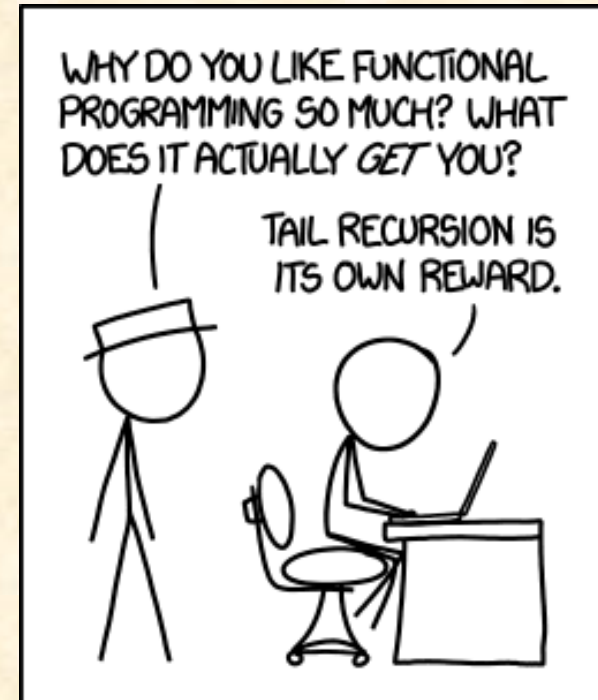
$$T(n) = \begin{cases} c_0 & n == 0 \\ T(n/2) + c_1 & n > 0 \end{cases}$$

```
int power(int x, int n) {  
    if (n == 0) {  
        return 1;  
    }  
    int result = power(x, n / 2);  
    result *= result;  
    if (n % 2 != 0) { // x is odd  
        result *= x;  
    }  
    return result;  
}
```

Revisiting Tail Recursion

- Recall your EECS 280:
- When a function is called, it gets a *stack frame*, which stores the local variables
- A simply recursive function generates a stack frame for each recursive call
- A function is *tail recursive* if there is no pending computation at each recursive step
 - "Re-use" the stack frame rather than create a new one
- Tail recursion and iteration are equivalent

<http://xkcd.com/1270/>



Recursion and the Stack

```
1 int factorial(int n) {  
2     if (n == 0)  
3         return 1;  
4     return n * factorial(n - 1);  
5 } // factorial()  
6  
7 int main(int, char *[]) {  
8     factorial(5);  
9 } // main()
```

The Program Stack (1)

- When a function call is made
 - 1a.** All local variables are saved in a special storage called *the program stack*
 - 2a.** Then argument values are pushed onto *the program stack*
- When a function call is received
 - 2b.** Function arguments are popped off the stack
- When **return** is issued within a function
 - 3a.** The return value is pushed onto *the program stack*
- When **return** is received at the call site
 - 3b.** The return value is popped off the *the program stack*
 - 1b.** Saved local variables are restored

The Program Stack (2)

- Program stack supports nested function calls
 - Six nested calls = six sets of local variables
- There is only one program stack (per thread)
 - NOT *the program heap*, (where dynamic memory is allocated)
- *Program stack* size is limited
- The number of nested function calls is limited
- Example: a bottomless (buggy) recursion function will exhaust *program stack* very quickly

Recursion vs. Tail Recursion

```
1 int factorial(int n) {  
2     if (n == 0)  
3         return 1;  
4     return n * factorial(n - 1);  
5 } // factorial()
```

Recursive

$\Theta(n)$ time complexity
 $\Theta(n)$ space complexity
(uses n stack frames)

```
6 int factorial(int n, int res = 1) {  
7     if (n == 0)  
8         return res;  
9     else  
10         return factorial(n - 1, res * n);  
11 } // factorial()
```

Tail recursive

$\Theta(n)$ time complexity
 $\Theta(1)$ space complexity
(reuses 1 stack frame)

* Note the default argument used to seed the result parameter. Alternatively, we could have used the "helper function" pattern you saw in EECS 280.

Recall: Recurrence Relations

- A *recurrence relation* describes the way a problem depends on a subproblem.
 - We can write recurrences for a computation:

$$x^n = \begin{cases} 1 & n == 0 \\ x * x^{n-1} & n > 0 \end{cases}$$

- We can also write recurrences for the time taken!

$$T(n) = \begin{cases} c_0 & n == 0 \\ T(n-1) + c_1 & n > 0 \end{cases}$$

- Or even for the amount of memory used!

$$M(n) = \begin{cases} c_0 & n == 0 \\ M(n-1) + c_1 & n > 0 \end{cases}$$

This would be for
the non-tail
recursive version.

A Tail Recursive `power` Function

```
1. int power(int x, unsigned y, int result = 1) {  
2.     if (y == 0) {  
3.         return result;  
4.     }  
5.     else if (y % 2 == 0) { // even  
6.         return power(x * x, y / 2, result);  
7.     }  
8.     else { // odd  
9.         return power(x * x, y / 2, result * x);  
10.    }  
11.} // power()
```

Practical Considerations

- Program stack is limited in size
 - It's actually pretty easy to exhaust this!
e.g. Computing the length of a very long vector using a "linear recursive" function with $\Theta(n)$ space complexity
- For a large data set
 - "Linear" recursion is a bad idea
 - Use tail recursion or iterative algorithms instead
- This doesn't mean everything should be tail recursive
 - Some problems can't be solved in $\Theta(1)$ space!

Binary Search Complexity

$$T(n) = \begin{cases} c_0 & n == 0 \\ T(n/2) + c_1 & n > 0 \end{cases} \rightarrow \Theta(\log n)$$

- Fits this same recurrence, because we cut the search space in half each time, and recurse into only one half.
- What if we had a different problem where we cut the space in three pieces? Or what if we cut in two pieces, but had to recursively process both? What if we had to do additional, non-constant work after the recursion?
 - Lots of recurrences like this! We'll introduce the "Master Theorem", which serves as a shortcut for solving them.

Common Recurrence Equations

Recurrence	Example	Big-O Solution
$T(n) = T(n / 2) + c$	Binary Search	$O(\log n)$
$T(n) = T(n - 1) + c$	Sequential Search	$O(n)$
$T(n) = 2T(n / 2) + c$	Tree Traversal	$O(n)$
$T(n) = T(n - 1) + c_1 * n + c_2$	Selection/etc. Sorts	$O(n^2)$
$T(n) = 2T(n / 2) + c_1 * n + c_2$	Merge/Quick Sorts	$O(n \log n)$

Solving Recurrences

- Recursion tree method [CLRS]
- Substitution method
 1. Write out $T(n)$, $T(n - 1)$, $T(n - 2)$
 2. Substitute $T(n - 1)$, $T(n - 2)$ into $T(n)$
 3. Look for a pattern
 4. Use a summation formula

Solving Recurrences

- We have used the substitution method to solve recurrence equations
- Another way to solve recurrence equations is the Master Method (AKA Master Theorem)

Master Theorem

Let $T(n)$ be a monotonically increasing function that satisfies:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$$T(1) = c_0$$

where $a \geq 1$, $b \geq 2$. If $f(n) \in \Theta(n^c)$, then:

$$T(n) \in \begin{cases} \Theta(n^{\log_b a}) & \text{if } a > b^c \\ \Theta(n^c \log_2 n) & \text{if } a = b^c \\ \Theta(n^c) & \text{if } a < b^c \end{cases}$$

Example

$$T(n) = 3T\left(\frac{n}{2}\right) + \frac{3}{4}n + 1 \quad \text{What are the parameters?}$$

$$a = 3$$

$$b = 2$$

$$c = 1$$

Therefore which condition? Since $3 > 2^1$, case #1.

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \quad T(n) \in \begin{cases} \Theta(n^{\log_b a}) & \text{if } a > b^c \\ \Theta(n^c \log_2 n) & \text{if } a = b^c \\ \Theta(n^c) & \text{if } a < b^c \end{cases}$$
$$f(n) \in \Theta(n^c)$$

Example

$$T(n) = 3T\left(\frac{n}{2}\right) + \frac{3}{4}n + 1 \quad \text{What are the parameters?}$$

$$a = 3$$

$$b = 2$$

$$c = 1$$

Therefore which condition? Since $3 > 2^1$, case #1.

Thus we conclude that:

$$T(n) \in \Theta(n^{\log_b a}) = \Theta(n^{\log_2 3})$$

Exercise

$$T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n} + 7 \quad \text{What are the parameters?}$$

$$a =$$

$$b =$$

$$c =$$

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \quad T(n) \in \begin{cases} \Theta(n^{\log_b a}) & \text{if } a > b^c \\ \Theta(n^c \log_2 n) & \text{if } a = b^c \\ \Theta(n^c) & \text{if } a < b^c \end{cases}$$

$$f(n) \in \Theta(n^c)$$

Exercise

$$T(n) = T\left(\frac{n}{2}\right) + \frac{1}{2}n^2 + n \quad \text{What are the parameters?}$$

$$a =$$

$$b =$$

$$c =$$

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \quad T(n) \in \begin{cases} \Theta(n^{\log_b a}) & \text{if } a > b^c \\ \Theta(n^c \log_2 n) & \text{if } a = b^c \\ \Theta(n^c) & \text{if } a < b^c \end{cases}$$
$$f(n) \in \Theta(n^c)$$

When Not to Use

- You cannot use the Master Theorem if:
 - $T(n)$ is not monotonic, such as $T(n) = \sin(n)$
 - $f(n)$ is not a polynomial, i.e. $f(n)=2^n$
 - b cannot be expressed as a constant, i.e.

$$T(n) = \sqrt{n}$$

- There is also a special fourth condition if $f(n)$ is not a polynomial; see later in slides

When Not to Use

Example:

$$T(n) = T(n - 1) + n$$

Master Theorem not applicable

$$T(n) \neq aT(n / b) + f(n)$$

Fourth Condition

- There is a 4th condition that allows polylogarithmic functions

If $f(n) \in \Theta(n^{\log_b a} \log^k n)$ for some $k \geq 0$,

Then $T(n) \in \Theta(n^{\log_b a} \log^{k+1} n)$

- This condition is fairly limited,
- No need to memorize/write down

Fourth Condition Example

- Say that we have the following recurrence:

$$T(n) = 2T\left(\frac{n}{2}\right) + n \log n$$

- Clearly $a=2$, $b=2$, but $f(n)$ is not polynomial
- However: $f(n) \in \Theta(n \log n)$ and $k = 1$

$$T(n) = \Theta(n \log^2 n)$$

If $f(n) \in \Theta(n^{\log_b a} \log^k n)$ for some $k \geq 0$,

Then $T(n) \in \Theta(n^{\log_b a} \log^{k+1} n)$

Exercise: Binary Max?

- A friend of yours claims to have discovered a (revolutionary!) new algorithm for finding the maximum element in an unsorted array:
 1. Split the array into two halves.
 2. Recursively find the maximum in each half.
 3. Whichever half-max is bigger is the overall max!
- Your friend says this algorithm leverages the power of "binary partitioning" to achieve better than linear time.
 - This sounds too good to be true. Give an intuitive argument why.
 - Use the master theorem to formally prove this algorithm is $\Theta(n)$.

Job Interview Question

Write an efficient algorithm that searches for a value in an $n \times m$ table (two-dim array).

This table is sorted along the rows and columns — that is,

$$\text{table}[i][j] \leq \text{table}[i][j + 1],$$

$$\text{table}[i][j] \leq \text{table}[i + 1][j]$$

- Obvious ideas: linear or binary search in every row
 - nm or $n \log m$... too slow

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

Potential Solution #1: Quad Partition

- Split the region into four quadrants, eliminate one.
- Then, recursively process the other 3 quadrants.
- Write a recurrence relation for the time complexity in terms of n , the length of one side:

$$T(n) = 3T(n/2) + c$$

Why $n/2$ and not $n/4$ if it's a quadrant?
Remember, n is the length of one side!

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

n 42

Potential Solution #1: Binary Partition

- Split the region into four quadrants.
- Scan down the middle column, until you find "where the value should be" if it were in that column.¹
- This allows you to eliminate two quadrants. (Why? Which ones?)
- Recursively process the other two.
- Write a recurrence relation, again in terms of the side length n :

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


1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

n

¹ Of course, you might get lucky and find the value here!

Potential Solution #3: Improved Binary Partition

How can we improve this?

- Split the region into four quadrants.
- Scan down the middle column,  you find "where the value should be" if it were in that column.¹
- This allows you to eliminate two quadrants. (Why? Which ones?)
- Recursively process the other two.
- Write a recurrence relation, again in terms of the side length n :

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

Use binary search!

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

n

¹ Of course, you might get lucky and find the value here!

Exercise: Use the Master Theorem

- Use the master theorem to find the complexity of each approach:
- Quad Partition:

$$T(n) = 3T(n/2) + c$$

- Binary Partition:

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

- Improved Binary Partition:

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

Solution: Use the Master Theorem

- Use the master theorem to find the complexity of each approach:
- Quad Partition:

$$T(n) = 3T(n/2) + c$$

$$T(n) = \Theta(n^{\log_2(3)}) \approx \Theta(n^{1.58})$$

- Binary Partition:

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

$$T(n) = \Theta(n \log n)$$

- Improved Binary Partition:

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

$$T(n) = \Theta(n)$$

Another Solution!

Stepwise Linear Search

```
1 bool stepWise(int mat[][N_MAX], int N, int target,  
2               int &row, int &col) {  
3     if (target < mat[0][0] || target > mat[N - 1][N - 1])  
4         return false;  
5     row = 0; col = N - 1;  
6     while (row <= N - 1 && col >= 0) {  
7         if (mat[row][col] < target)  
8             row++;  
9         else if (mat[row][col] > target)  
10            col--;  
11        else  
12            return true;  
13    } // while  
14    return false;  
15 } // stepWise()
```

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

Runtime Comparisons

- Source code and data ($M = N = 100$) available at
<http://www.leetcode.com/2010/10/searching-2d-sorted-matrix.html>
<http://www.leetcode.com/2010/10/searching-2d-sorted-matrix-part-ii.html>
<http://www.leetcode.com/2010/10/searching-2d-sorted-matrix-part-iii.html>
- Runtime for 1,000,000 searches

Algorithm	Runtime
Binary search	31.62s
Diagonal Binary Search	32.46s
Step-wise Linear Search	10.71s
Quad Partition	17.33s
Binary Partition	10.93s
Improved Binary Partition	6.56s

Questions for Self-Study

- Consider a recursive function that only calls itself. Explain how one can replace recursion by a loop and an additional stack.
- Go over the Master Theorem in the CLRS textbook
- Which cases of the Master Theorem were exercised for different solutions in the 2D-sorted-matrix problem?
- Solve the same recurrences by substitution w/o the Master Theorem
- Write (and test) programs for each solution idea, time them on your data

