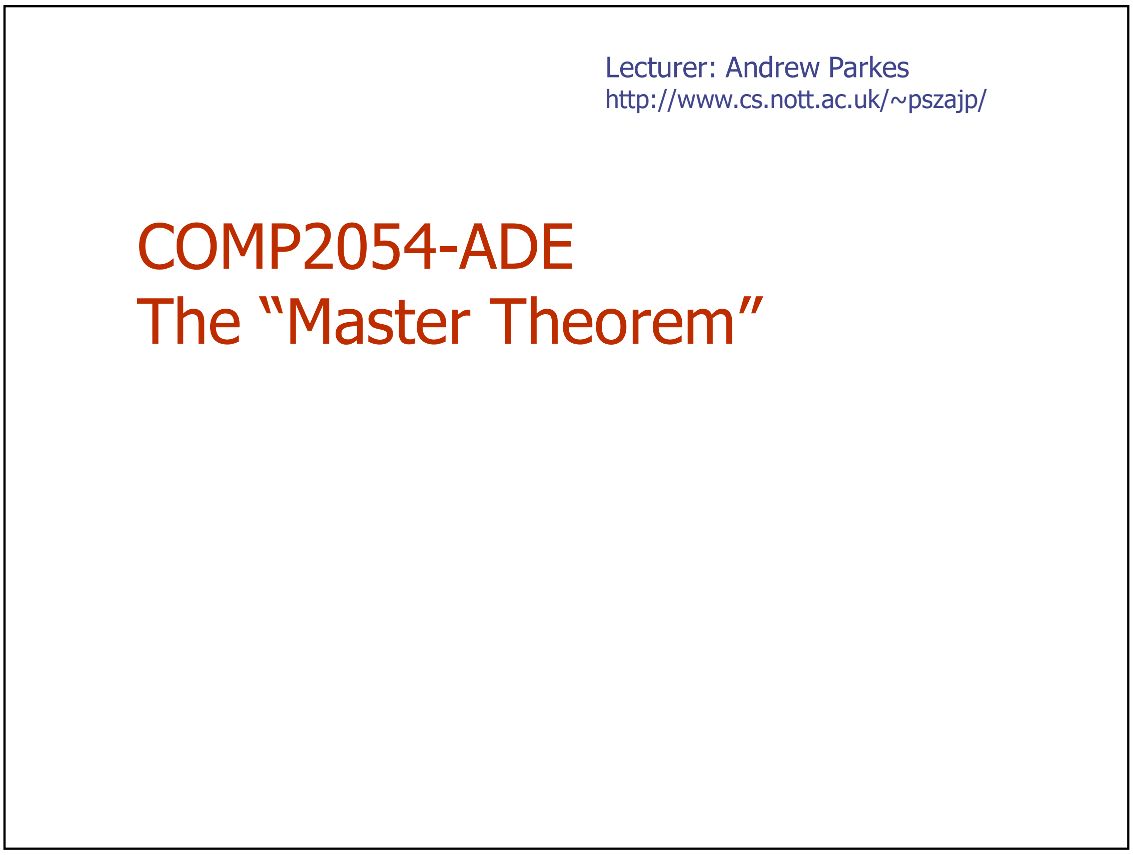
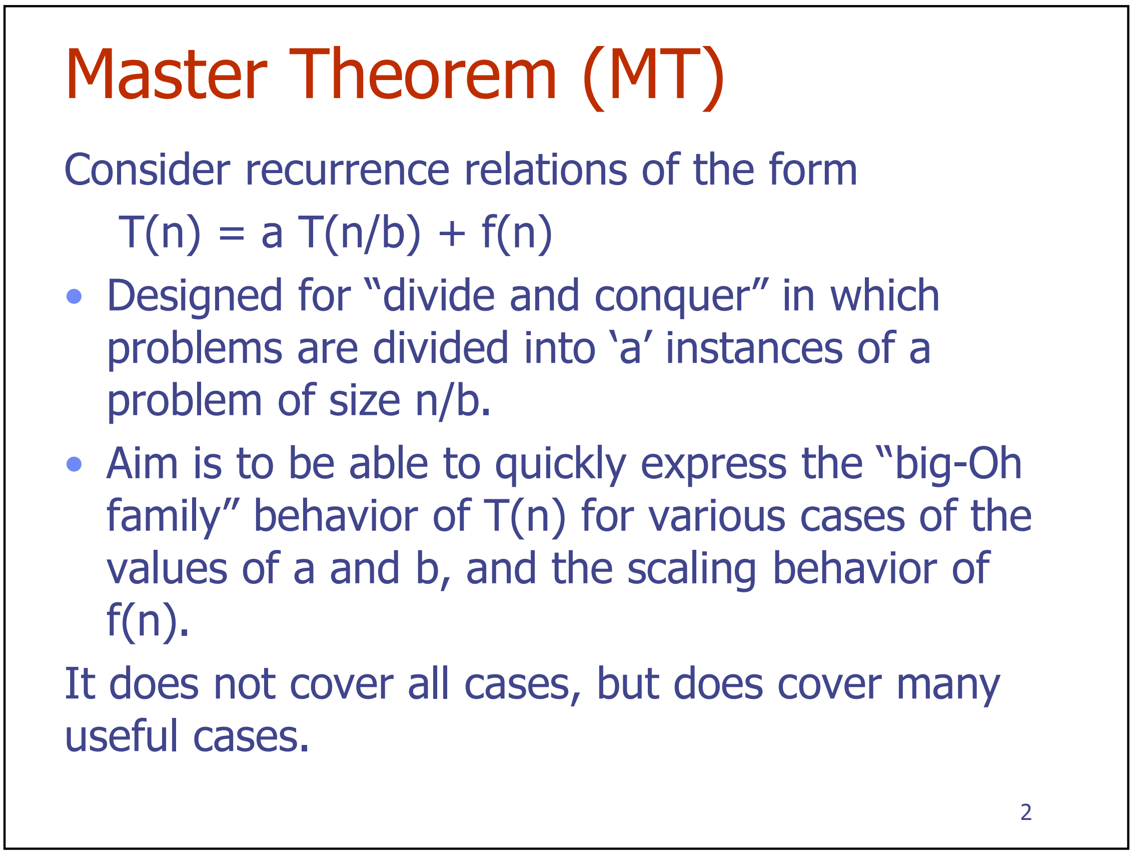


Lecturer: Andrew Parkes
http://www.cs.nott.ac.uk/~pszajp/

# COMP2054-ADE
# The “Master Theorem”

“Master theorem” is a fancy term for a set of rules to quickly handle some common cases of recurrence relations



## Master Theorem (MT)

Consider recurrence relations of the form

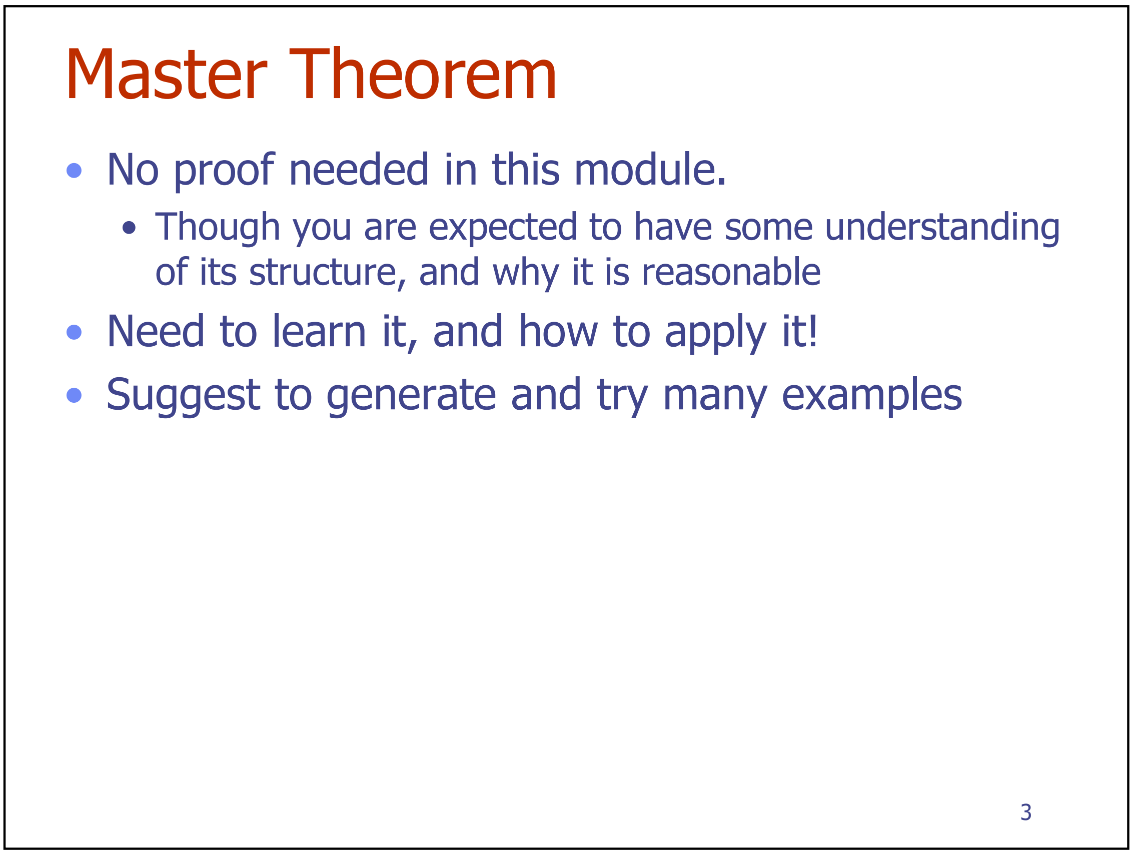
$T(n) = a\ T(n/b) + f(n)$

- Designed for "divide and conquer" in which problems are divided into 'a' instances of a problem of size n/b.
- Aim is to be able to quickly express the "big-Oh family" behavior of T(n) for various cases of the values of a and b, and the scaling behavior of f(n).

It does not cover all cases, but does cover many useful cases.

2

It handles cases of the form we often see in recursive "divide and conquer" algorithms.

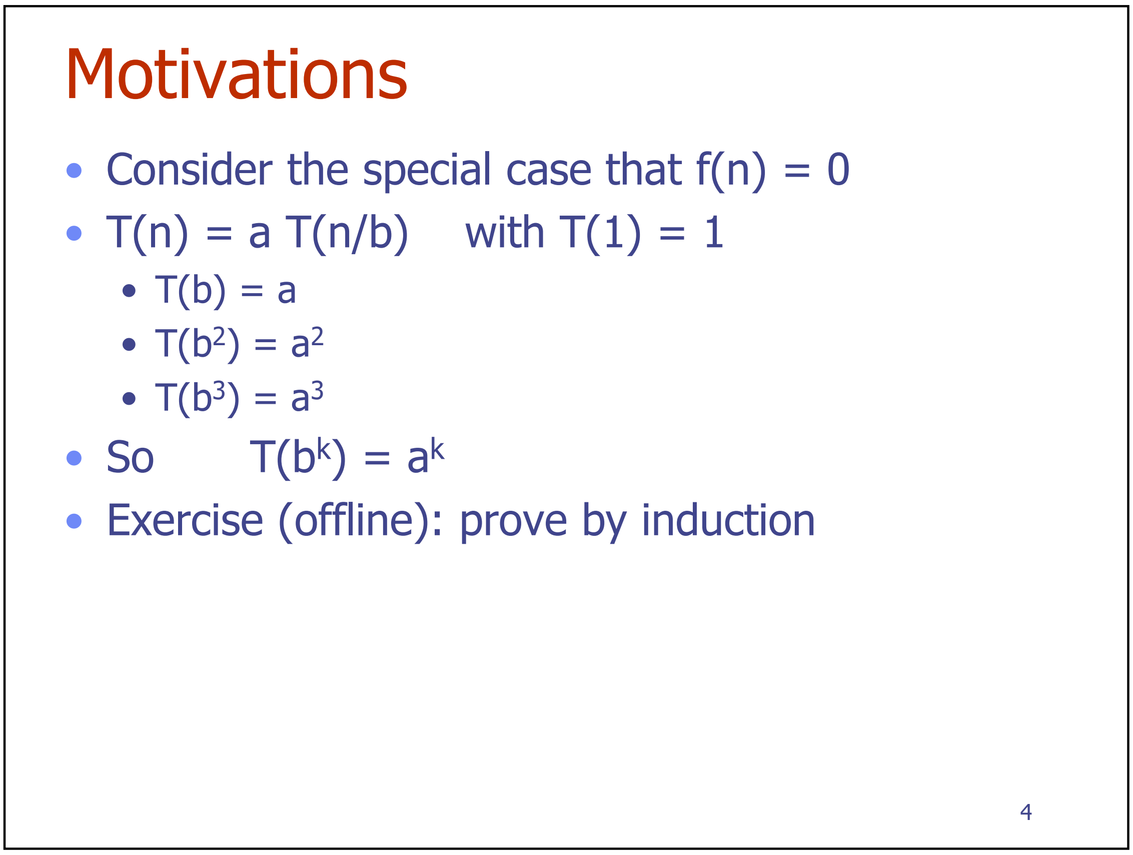


## Master Theorem

- No proof needed in this module.
  - Though you are expected to have some understanding of its structure, and why it is reasonable
- Need to learn it, and how to apply it!
- Suggest to generate and try many examples

3

Just need to learn it. But most importantly to be able to apply it.

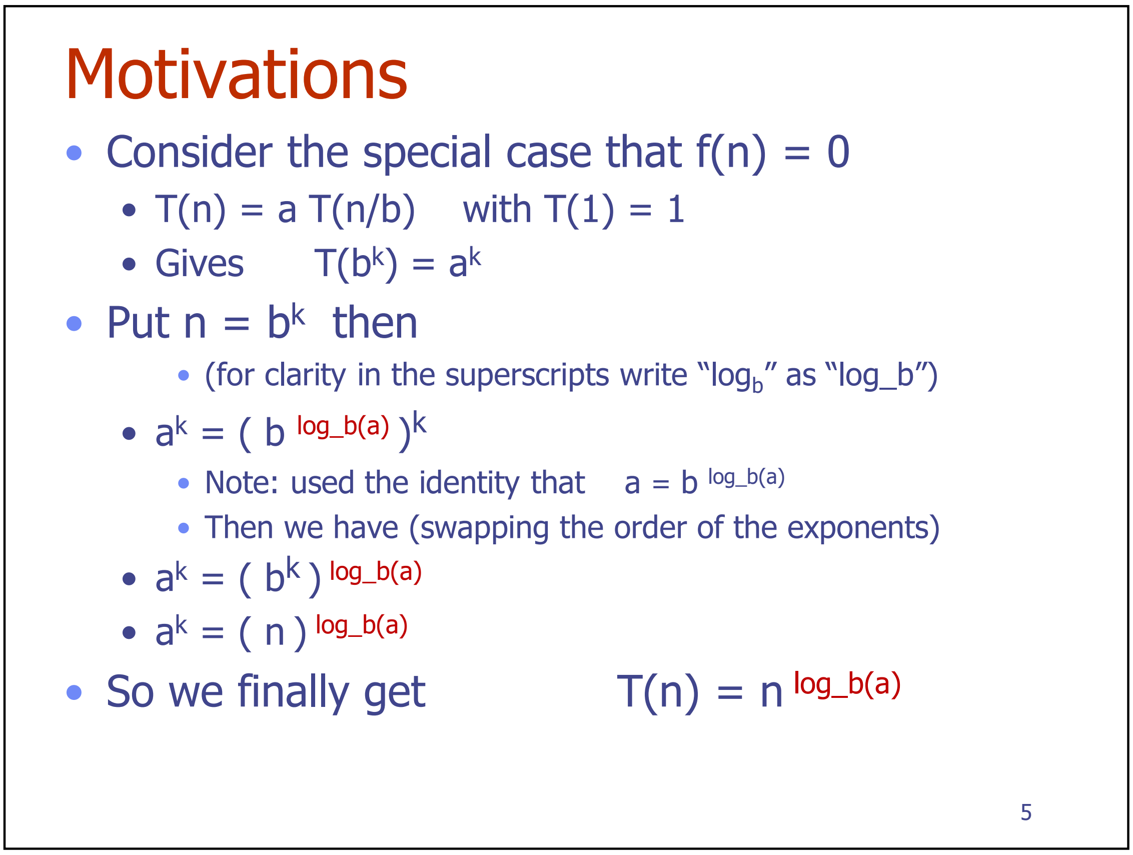


## Motivations

- Consider the special case that $f(n) = 0$
- $T(n) = a\,T(n/b)$ with $T(1) = 1$
  - $T(b) = a$
  - $T(b^2) = a^2$
  - $T(b^3) = a^3$
- So $T(b^k) = a^k$
- Exercise (offline): prove by induction

4

First the general theorem needs some context to see what kinds of properties of a,b, f are likely to be relevant.
So consider a simple case.
We can easily directly get an exact answer in this case.

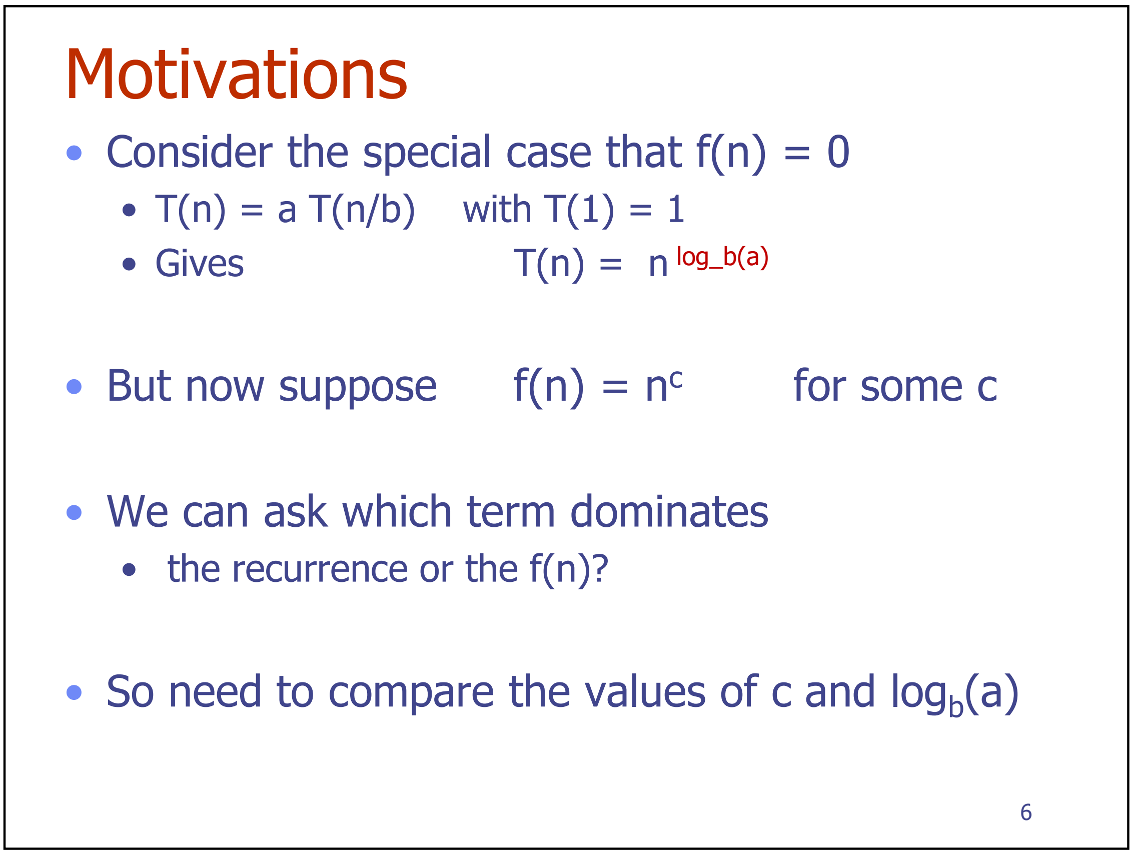


## Motivations

- Consider the special case that $f(n) = 0$
  - $T(n) = a\,T(n/b)$ with $T(1) = 1$
  - Gives $T(b^k) = a^k$
- Put $n = b^k$ then
    - (for clarity in the superscripts write "$\log_b$" as "log_b")
  - $a^k = ( b^{\text{log\_b(a)}} )^k$
    - Note: used the identity that $a = b^{\text{log\_b(a)}}$
    - Then we have (swapping the order of the exponents)
  - $a^k = ( b^k )^{\text{log\_b(a)}}$
  - $a^k = ( n )^{\text{log\_b(a)}}$
- So we finally get $T(n) = n^{\text{log\_b(a)}}$

5

But we need to express b^k in terms of n.
For purposes of motivation , we can also consider a case that is purely the f(n) term.
In which the answer is directly given.
Now start to think in terms of which of these would be the dominant term in big-Oh terms.
Let's just suppose f is a power law n^c.
We need to be able to compare.
After a bit of algebra, we find the recurrence term has a power log_b(a) and the f term has power c. So the answer is likely to depend on the comparison of these.

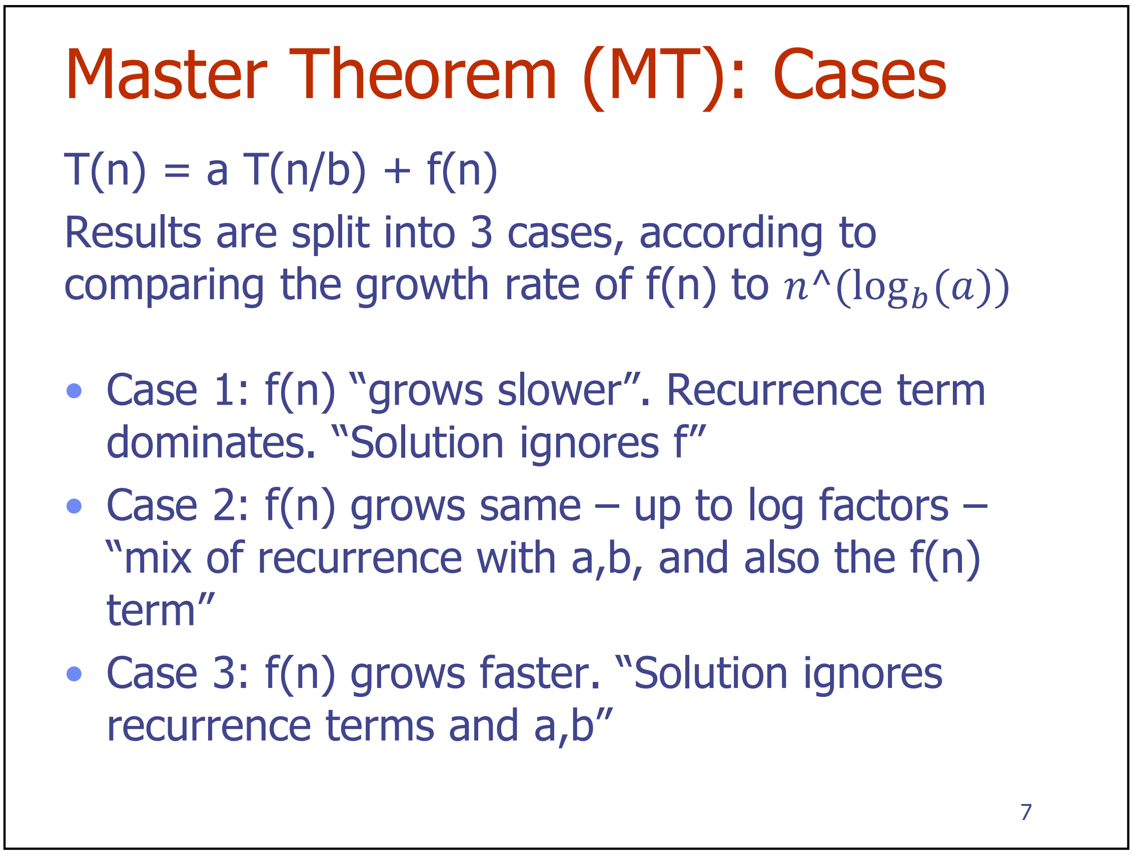


## Motivations

- Consider the special case that $f(n) = 0$
  - $T(n) = a\,T(n/b)$ with $T(1) = 1$
  - Gives $T(n) = n^{\text{log\_b(a)}}$
- But now suppose $f(n) = n^c$ for some c
- We can ask which term dominates
  - the recurrence or the f(n)?
- So need to compare the values of c and $\log_b(a)$

6

But we need to express b^k in terms of n.
For purposes of motivation , we can also consider a case that is purely the f(n) term.
In which the answer is directly given.
Now start to think in terms of which of these would be the dominant term in big-Oh terms.
Let's just suppose f is a power law n^c.
We need to be able to compare.
After a bit of algebra, we find the recurrence term has a power log_b(a) and the f term has power c. So the answer is likely to depend on the comparison of these.



## Master Theorem (MT): Cases

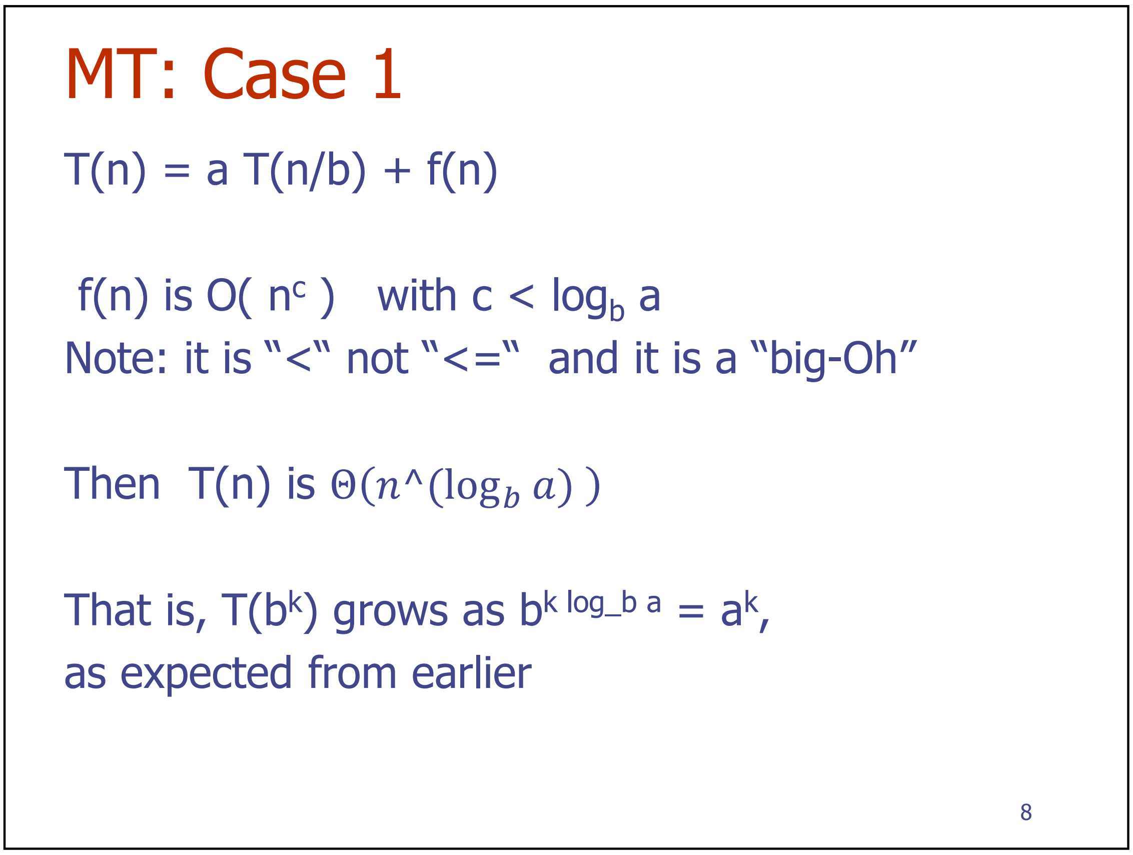
T(n) = a T(n/b) + f(n)

Results are split into 3 cases, according to comparing the growth rate of f(n) to $n$^$(\log_b(a))$

- Case 1: f(n) "grows slower". Recurrence term dominates. "Solution ignores f"
- Case 2: f(n) grows same – up to log factors – "mix of recurrence with a,b, and also the f(n) term"
- Case 3: f(n) grows faster. "Solution ignores recurrence terms and a,b"

7

Then naturally get 3 cases. Recurrence wins. Or f wins, and the middle case where they are equal.



## MT: Case 1

$T(n) = a\,T(n/b) + f(n)$

$f(n)$ is $O(n^c)$ with $c < \log_b a$

Note: it is "<" not "<=" and it is a "big-Oh"

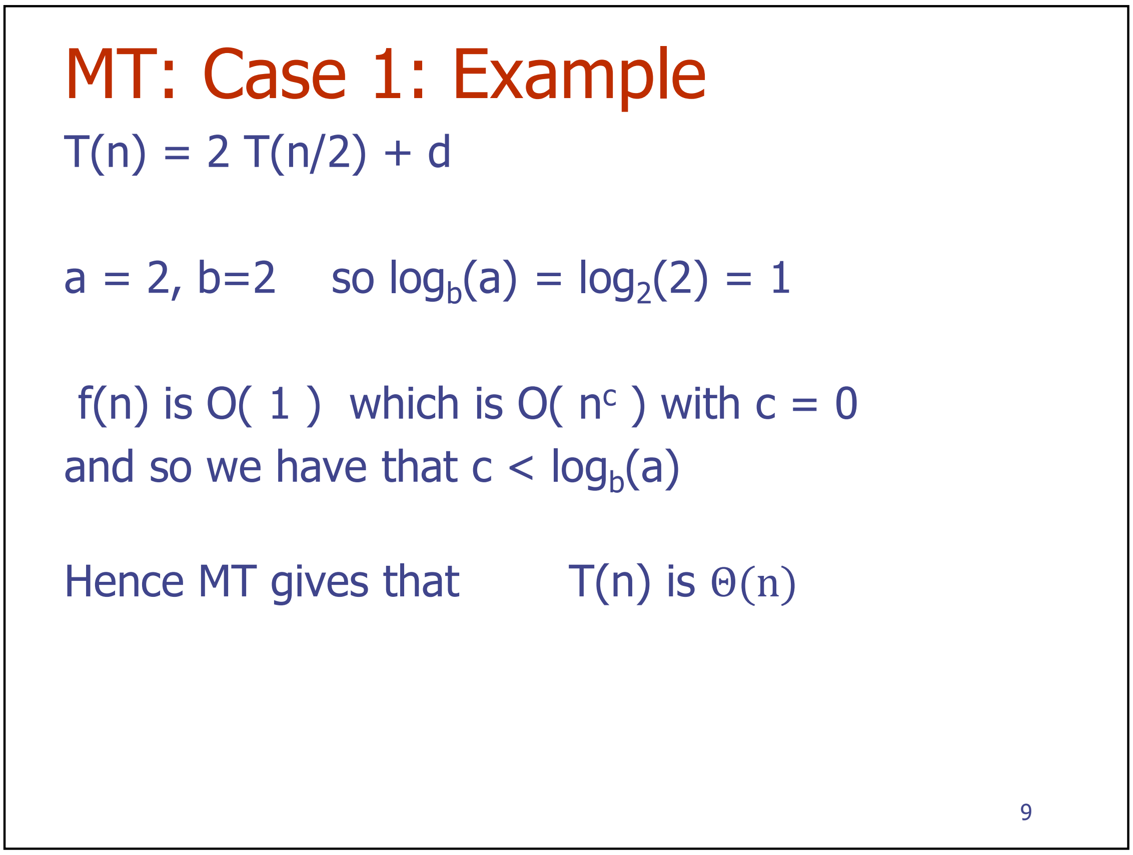
Then $T(n)$ is $\Theta(n$^$(\log_b a))$

That is, $T(b^k)$ grows as $b^{k \log\_b a} = a^k$,

as expected from earlier

8

Case 1. if the f grows strictly less then the f term, then basically we can ignore the f term in getting the simple case.
Note this is a bit like the "drop smaller terms" rules for extracting the Big-Oh family behaviour of functions.
Which we can just see is from rearranging the solution that we had before.



## MT: Case 1: Example

$T(n) = 2\,T(n/2) + d$

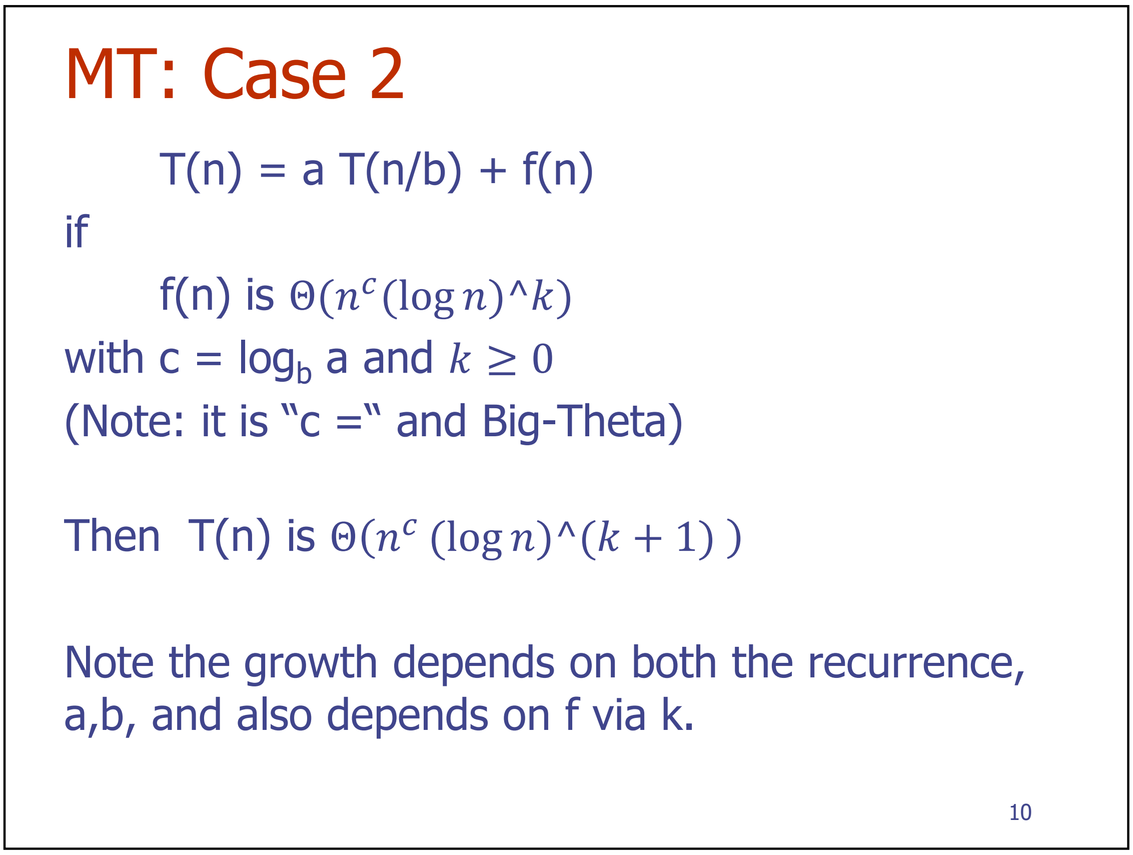
$a = 2, b=2$ so $\log_b(a) = \log_2(2) = 1$

$f(n)$ is $O(\ 1\ )$ which is $O(\ n^c\ )$ with $c = 0$
and so we have that $c < \log_b(a)$

Hence MT gives that $T(n)$ is $\Theta(n)$

9

Simple example. Note this is just substation. And note we do need to be familiar with logs again!!



## MT: Case 2

T(n) = a T(n/b) + f(n)

if

f(n) is $\Theta(n^c (\log n)^\wedge k)$

with c = $\log_b$ a and $k \geq 0$

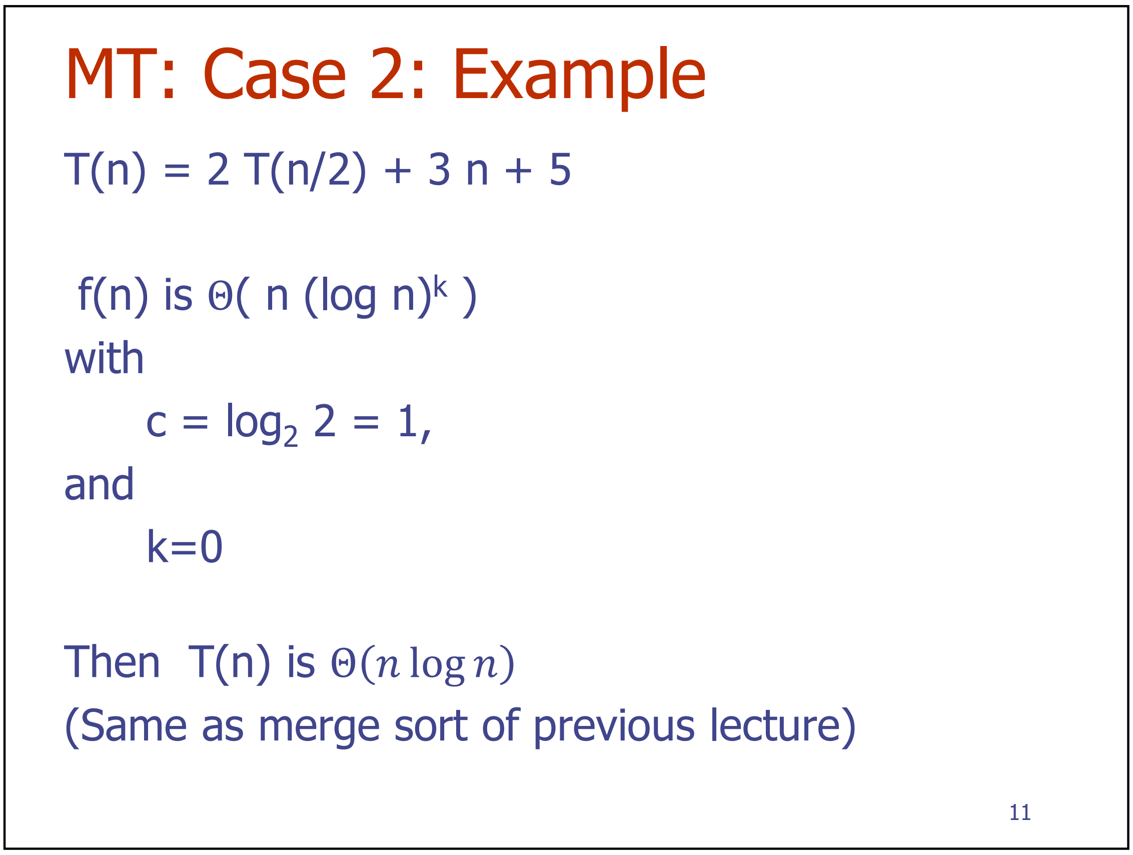
(Note: it is "c =" and Big-Theta)

Then T(n) is $\Theta(n^c \, (\log n)^\wedge(k+1)\,)$

Note the growth depends on both the recurrence, a,b, and also depends on f via k.

10

Now case 2. The two terms have similar growth up to a log term.
The c=logb(a) is for the similar growth, but the theorem actually allows us to include some power of a log.
The statement is that we get the same, but the power of the log is increased by one.



## MT: Case 2: Example

T(n) = 2 T(n/2) + 3 n + 5

f(n) is $\Theta( n (\log n)^k )$

with

$c = \log_2 2 = 1$,

and

k=0

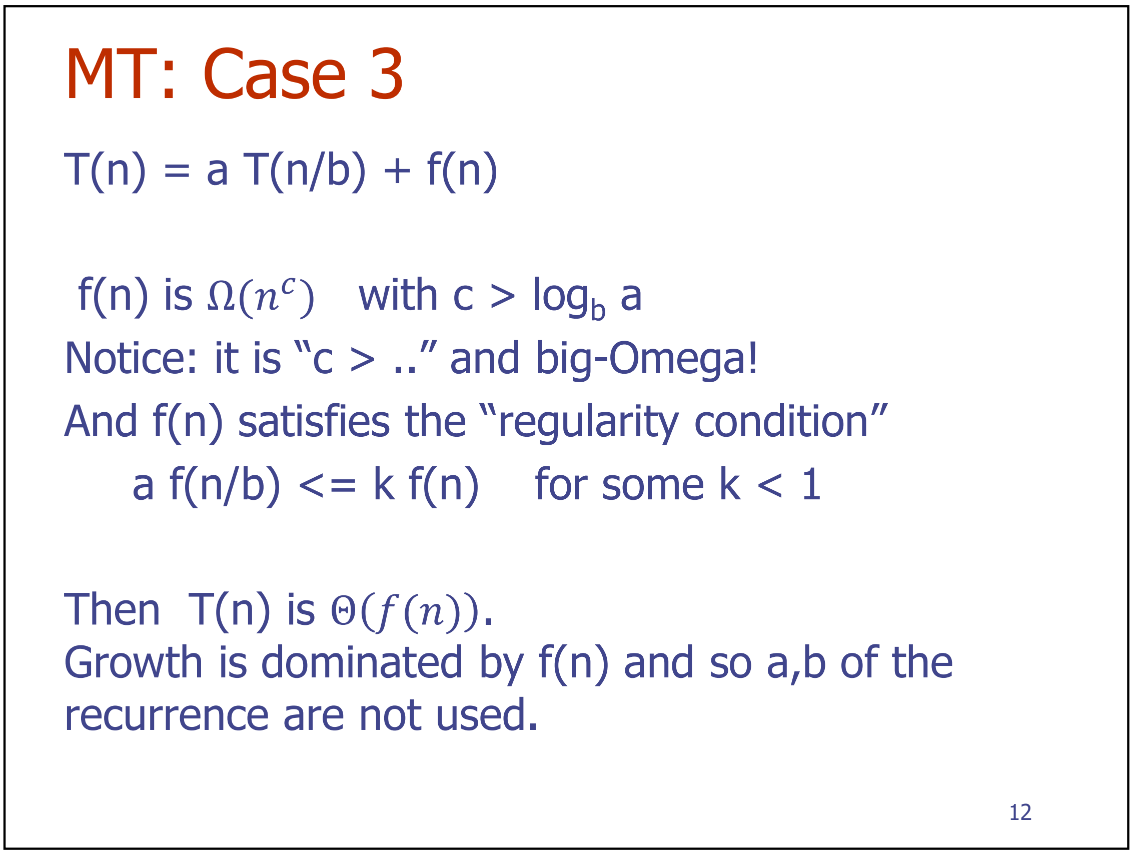
Then T(n) is $\Theta(n \log n)$

(Same as merge sort of previous lecture)

11

Simple example is the case from mergesort. Note that k=0 is allowed.

There is no ‘obvious’ log term but always have to remember that it can be hidden by the zero power.
The case just says we increment the power of the log.
Hence immediately get the n log n of mergesort.
“Quick and easy”.
And note that can easily handle extra terms in the f function



## MT: Case 3

T(n) = a T(n/b) + f(n)

f(n) is $\Omega(n^c)$ with c > $\log_b a$

Notice: it is "c > .." and big-Omega!

And f(n) satisfies the "regularity condition"

a f(n/b) <= k f(n) for some k < 1

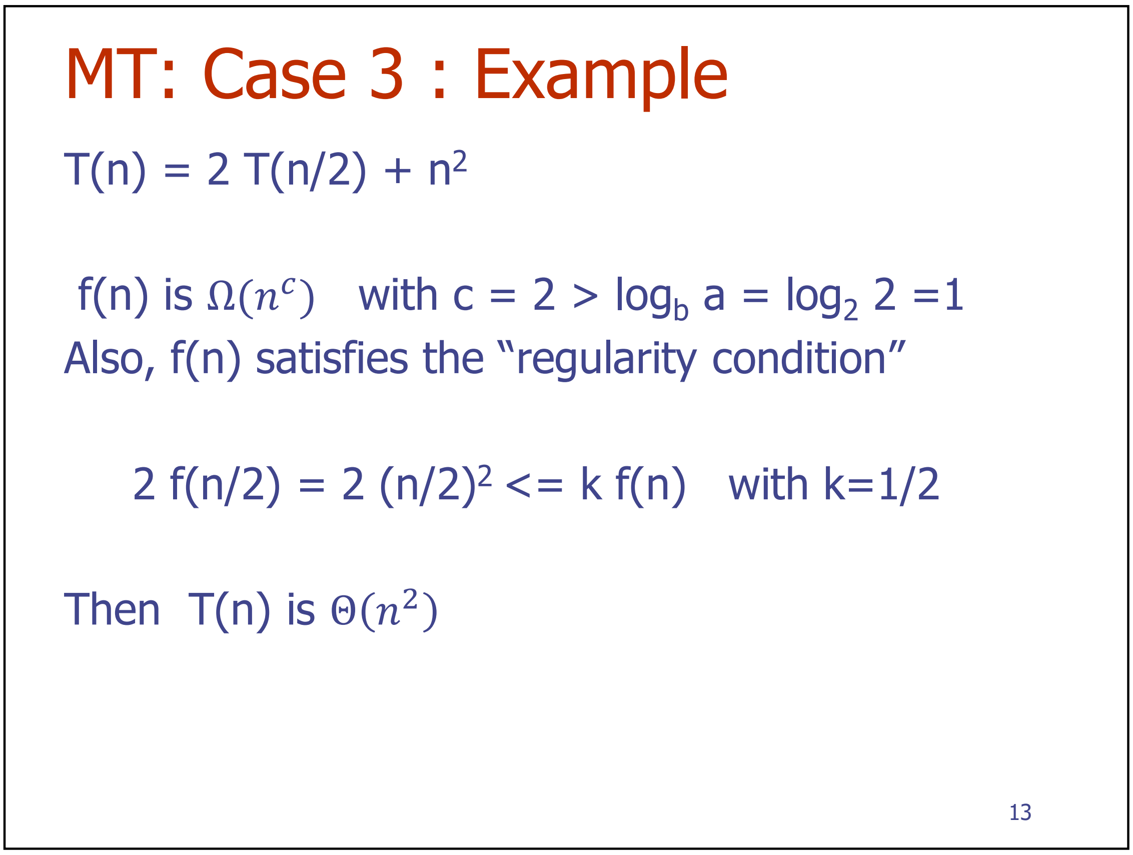
Then T(n) is $\Theta(f(n))$.

Growth is dominated by f(n) and so a,b of the recurrence are not used.

12

Last case; f wins.

Note that it has to be Omega to express that f is definitely winning. Compare to the big-Oh in case 1, and Theta in case 2.
Then there is nothing to do.



## MT: Case 3 : Example

$T(n) = 2\,T(n/2) + n^2$

$f(n)$ is $\Omega(n^c)$ with $c = 2 > \log_b a = \log_2 2 = 1$

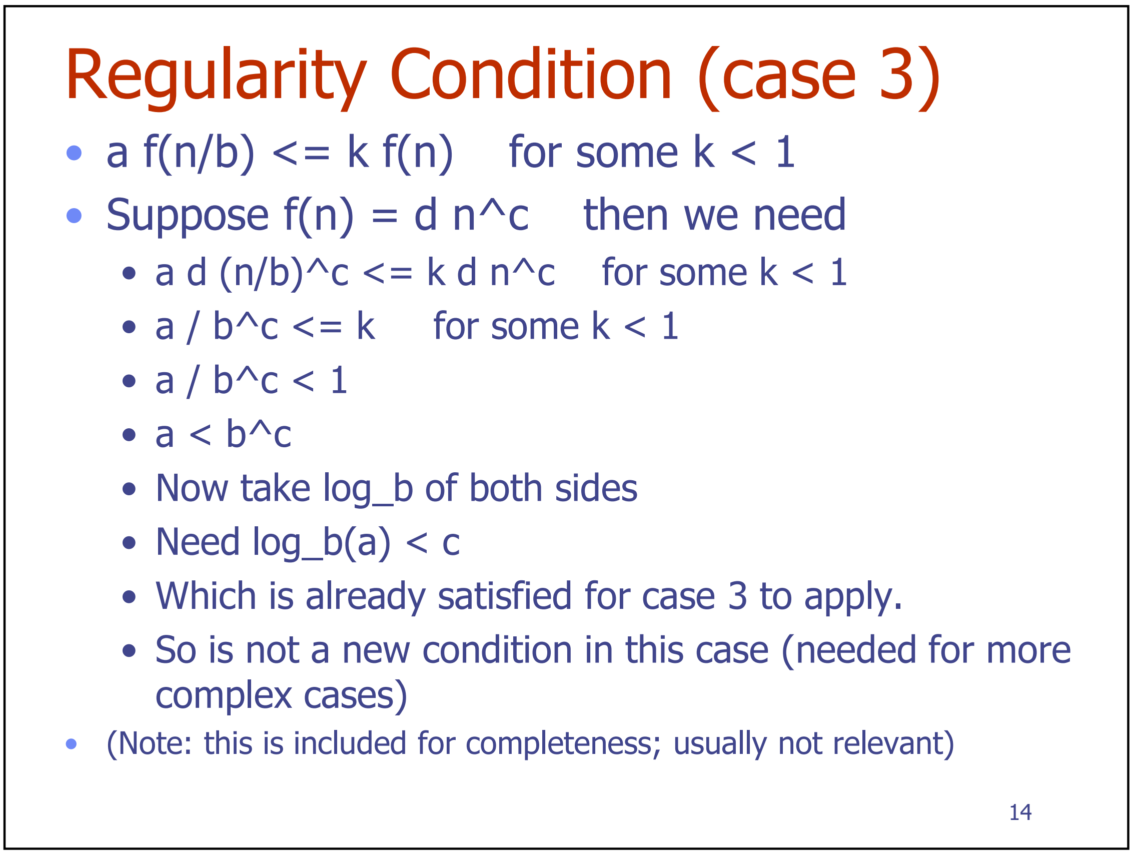
Also, $f(n)$ satisfies the "regularity condition"

$$2\,f(n/2) = 2\,(n/2)^2 <= k\,f(n) \quad \text{with } k=1/2$$

Then $T(n)$ is $\Theta(n^2)$

13

Try an example.
Lets

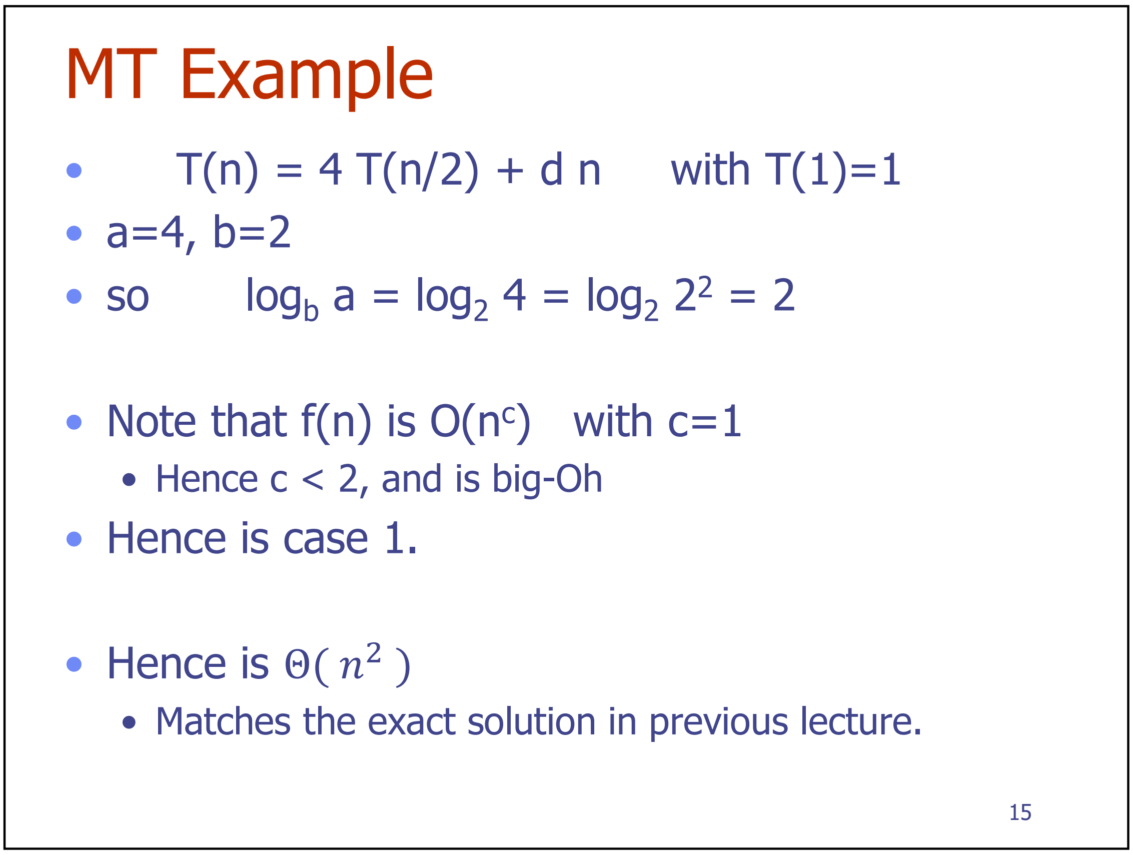


## Regularity Condition (case 3)

- a f(n/b) <= k f(n)    for some k < 1
- Suppose f(n) = d n^c    then we need
  - a d (n/b)^c <= k d n^c    for some k < 1
  - a / b^c <= k    for some k < 1
  - a / b^c < 1
  - a < b^c
  - Now take log_b of both sides
  - Need log_b(a) < c
  - Which is already satisfied for case 3 to apply.
  - So is not a new condition in this case (needed for more complex cases)
- (Note: this is included for completeness; usually not relevant)

14

There is a technical issue that we need for case 3 include for completeness, but mostly we will ignore as it is usually satisfied.
It says f behaves nicely and should be checked.

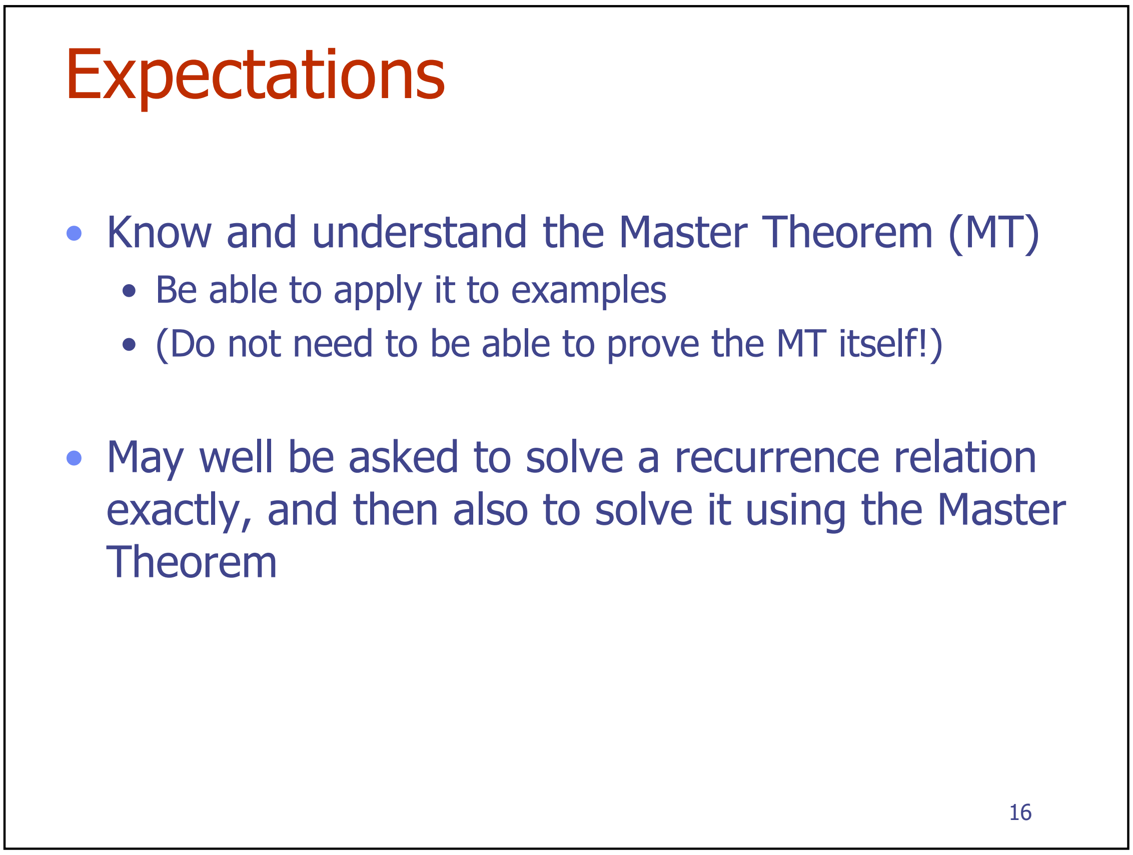


## MT Example

- $T(n) = 4\,T(n/2) + d\,n$ with $T(1)=1$
- $a=4$, $b=2$
- so $\log_b a = \log_2 4 = \log_2 2^2 = 2$
- Note that $f(n)$ is $O(n^c)$ with $c=1$
  - Hence $c < 2$, and is big-Oh
- Hence is case 1.
- Hence is $\Theta(n^2)$
  - Matches the exact solution in previous lecture.

15

Another example.
We read off the b,a c values and then compute logb(a) and compare to c.
Then we find is case 1.
Hence immediately get quadratic.



## Expectations

- Know and understand the Master Theorem (MT)
  - Be able to apply it to examples
  - (Do not need to be able to prove the MT itself!)
- May well be asked to solve a recurrence relation exactly, and then also to solve it using the Master Theorem

16

So summary, is need to be able to use the MT. No need to know how to prove it!