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

COMP2054-ADE
Recurrence Relations

Recurrence Relations

- A recurrence relation is a recursively-defined function
  - But, generally, applied to the case when the function is some measure of resources …
  - and we might only want the big-Oh family properties of the solution
- Suppose that the runtime of a program is T(n), then a recurrence relation will express T(n) in terms of its values at other (smaller) values of n.
  - (By "runtime" is usually meant something 'abstract' such as counting of operations. We do not care about difficulties of true timing in nanoseconds.)

Example: Merge Sort

- Suppose the runtime of merge-sort of an array of n integers is T(n). Then

$$T(n) = 2\,T(n/2) + b + a\,n$$

  - "2 T(n/2)" is due to having to sort the two sub-arrays each of size n/2
  - "b" is the cost of doing the split
  - "a n" is the cost of doing the merge (and any copying to/from the workspace, etc.)

Example: Merge Sort

- Suppose the runtime of merge-sort of an array of n integers is $T(n)$.
  - Gave the recursive case
  - We also need a base-case.
  - We can take

$$T(1) = 1$$

  - As just need to check the array length is 1.
  - If make it some other number then we could just rescale the results for $T(n)$ to match. This convention is just for simplicity and convenience.

Example: Merge Sort

- Suppose the runtime of merge-sort of an array of n integers is T(n).
  - Note that we simplified: if n is odd then we ought to have
    - "T(n/2) + T(n/2+1) + …
  - E.g. at n=9 T(9) = T(4) + T(5) + …
  - However, (generally), ignore such details, as they (generally) make no difference to the final statements of big-Oh behaviour.

Example: Merge Sort

- How would we solve

$$T(n) = 2\ T(n/2) + b + a\ n$$

- We will do some special cases:

Example 1:

- How would we solve

  $T(n) = 2\, T(n/2)$ with $T(1)=1$

(If watching offline, pause and try)

Example 1:

- How would we solve
  $T(n) = 2\,T(n/2)$ with $T(1)=1$
- Given $T(1)=1$, what else can we evaluate?
  - $T(2)$ or $T(1/2)$
    - but want to solve for larger n, not smaller fractions, hence:
  - $T(2) = 2\,T(1) = 2$, and then can get
  - $T(4) = 2\,T(4/2) = 2\,T(2) = 4$
  - $T(8) = 2\,T(8/2) = 2\,T(4) = 8$
  - Etc.
- It seems a good guess $T(2^k) = 2^k$
- But how do we prove it in general?
  - Induction!

Example 1 (cont):

- How would we solve
  $T(n) = 2\ T(n/2)$ with $T(1)=1$
- Claim: forall k. $T(2^k) = 2^k$
- Proof by induction:
  - Base case: true at k=0. Recall: $2^0 = 1$
  - Step case. Suppose true at k (hypothesis), then need to show is true at k+1:
    $T(\ 2^{(k+1)}\ ) = 2\ T(2^{k+1}/2)$ (using the recurrence)
    $= 2\ T(2^k)$ (simplification)
    $= 2\ 2^k$ (using the hypothesis claim)
    $= 2^{(k+1)}$ QED. As matches the claim at k+1

Make sure you understand the structure of this proof!

Example 1 (cont):

- How would we solve
  $T(n) = 2\,T(n/2)$ with $T(1)=1$
- Showed $T(2^k) = 2^k$ for all k in N, that is,
  $T(n) = n$ forall n in $\{1,2,4,8,16\ldots\}$
- What about other values of n ? E.g. what is T(3)?
- Depends what one wants!
  - Usually (in this module) just want the growth rate
    - So we can just be imprecise with, T(n)=n for all n, and so then is $\Theta(n)$
  - Might need to refine the recurrence relation, use ceiling and floors to get integers.
    - Messy! But would be the same scaling answer, so do not (usually) bother

Example 2:

- How would we solve

  $T(n) = 2\ T(n/2) + b$ with $T(1)=1$

(Pause and try)

Example 2:

- How would we solve
  $T(n) = 2\,T(n/2) + b$ with $T(1)=1$
- We know $T(1)=1$, hence
  - $T(2) = 2\,T(1) + b = 2 + b$
  - $T(4) = 2\,T(4/2) + b = 2\,(2 + b) + b = 4 + (2+1)b$
  - $T(8) = 2\,(4 + (2+1)b) + b = 8 + (4+2+1)\,b$
- It seems a good guess
  $T(2^k) = 2^k + (2^{(k-1)}+\ldots+1)b$
  $= 2^k + (2^k - 1)\,b$

So $T(n) = n+(n-1)\,b = (1+b)n-b$ for n in $\{1,2,4,8\ldots\}$

Still $\Theta(n)$

Example 2: (cont)

- How would we solve

  $T(n) = 2\ T(n/2) + b$ with $T(1)=1$
- Claim: $T(2^k) = 2^k + (2^k - 1)\ b$
- Proof by induction:
  - Base case: $k=0$, $T(1) = 1 + (1-1)*b = 1$
  - Step case: assume true at k
  - $T(2^{k+1}) = 2\ T(2^k) + b$

    $= 2\ (2^k + (2^k - 1)\ b\ ) + b$

    $= 2^{k+1} + (2^{k+1} - 2 + 1)\ b$

    $= 2^{k+1} + (2^{k+1} - 1)\ b$

  QED.

Example 3:

- How would we solve

  $T(n) = 2\,T(n/2) + a\,n \qquad \text{with } T(1)=1$

(Pause and try, even if you saw the answer before!)

Example 3:

- How would we solve
  $T(n) = 2\,T(n/2) + a\,n$ with $T(1)=1$
- We know $T(1)=1$, hence try with $n = 2^k$
  - k=1: $T(2) = 2\,T(1) + 2\,a = 2 + 2\,a$
  - k=2: $T(4) = 2\,T(4/2) + 4\,a = 2\,(2 + 2\,a) + 4\,a$
    $= 4 + 2 * 4\,a$
  - k=3: $T(8) = 2\,(4 + 8\,a) + 8\,a = 8 + 3 * 8\,a$
  - k=4: $T(16) = 2\,(8 + 3 * 8\,a) + 16\,a = 16 + 4*16\,a$
- It seems a good guess
  $T(2^k) = 2^k + k\,2^k\,a = 2^k\,(1 + k\,a)$

So $T(n) = n + \log_2(n)\,n\,a$ for $n = \{1,2,4,8\ldots\}$

Now $\Theta(n \log n)$: what we expect of merge-sort !!

Example 3: (cont)

- How would we solve
  $T(n) = 2\,T(n/2) + a\,n \quad$ with $T(1)=1$
- Claim: $T(2^k) = 2^k + k\,2^k\,a = 2^k\,(1 + k\,a)$
- Base case: $k=0 \quad T(1) = 1 + 0 * 1 * a = 1$
- Step case: assume true at k
  - $T(2^{k+1}) = 2\,T(2^k) + 2^{k+1}\,a$
    $= 2\,(\,2^k + k\,2^k\,a\,) + 2^{k+1}\,a$
    $= 2^{k+1} + k\,2^{k+1}\,a + 2^{k+1}\,a$
    $= 2^{k+1} + (k+1)\,2^{k+1}\,a$

  QED

Example 4:

- How would we solve

  $T(n) = 4\ T(n/2) \qquad \text{with } T(1)=1$

Example 4:

- How would we solve
  $T(n) = 4\ T(n/2)$ with $T(1)=1$
- We know $T(1)=1$, hence
  - $k=1$: $T(2) = 4\ T(1) = 4 = 2 * 2$
  - $k=2$: $T(4) = 4\ T(4/2) = 4 * 4 = 16$
  - $k=3$: $T(8) = 4\ (16) = 64 = 8 * 8$
  - $k=4$: $T(16) = 4\ (8*8) = (2*8) * (2*8) = 16 * 16$
- It seems a good guess
  $T(2^k) = (\ 2^k\ )^2$

So $T(n) = n^2$ for n in $\{1,2,4,8\ldots\}$

Hence $\Theta(n^2)$

Example 4:

- How would we solve
  $T(n) = 4\,T(n/2)$ with $T(1)=1$
- Claim $T(2^k) = (\,2^k\,)^2$
- Proof by induction:
  - Base case: $k=1$ $T(1) = 1^2 = 1$
  - Step case: assume true at k.
    - $T(2^{k+1}) = 4\,T(\,2^k\,) = 2 * 2 * 2^k * 2^k = (2^{k+1})^2$ QED.

Exercise (offline): Ensure you understand the details

Example 5:

- How would we solve

  $T(n) = 4\,T(n/2) + d\,n \quad \text{with } T(1)=1$

Example 5:

- How would we solve
  $T(n) = 4\ T(n/2) + d\ n$ with $T(1)=1$
- We know $T(1)=1$, hence
  - k=1: $T(2) = 4\ T(1) + 2\ d = 4 + 2\ d = 2^2 + 2 * 1 * d$
  - k=2: $T(4) = 4\ T(4/2) + 4\ d = 4\ (\ 4 + 2\ d) + 4\ d$
    $= 16 + 12\ d = 4^2 + 4*3*d$
  - k=3: $T(8) = 4\ (16 + 12\ d) + 8\ d = 8^2 + 8*7\ d$
- It seems a good guess that
  $T(n) = n^2 + n(n-1)\ d$
  Exercise: proof it by induction.
- So $T(n)$ is $\Theta(n^2)$. "Value of d does not matter"

Example 6:

- $T(n) = T(n/2) + d \qquad\qquad T(1) = 1$
  - E.g. From binary search of a sorted array

Example 6:

- $T(n) = T(n/2) + d$ $\qquad T(1) = 1$
  - E.g. From binary search of a sorted array
    - k=1: $T(2) = T(1) + d = 1 + d$
    - k=2: $T(4) = T(2) + d = 1 + 2d$
    - k=3: $T(8) = T(4) + d = 1 + 3d$
    - k=4: $T(16) = T(8) + d = 1 + 4d$
  - Guess $T(2^k) = 1 + k\,d$
  - Hence: $T(n) = 1 + d \log_2(n)$
  - Exercise (offline): prove by induction.
  - That is, T(n) is $\Theta(\log n)$, as expected for binary search

Simple sorting?

- Bubble sort etc. do not naturally generate recurrence relations as they are not naturally recursive.
  - But could be phrased that way
  - Bubble sort
  - $T(n) = T(n-1) + d\ n$
    - $d\ n$ for a pass of the outer loop
    - $T(n-1)$ for the remaining passes – which now only need to process n-1 numbers.

Example 7:

- T(n)= T(n - 1) + d n T(1) = 1
  - (Bubble sort, etc.)
  - T(2) = T(1) + 2 d = 1 + 2 d
  - T(3) = (1 + 2 d) + 3 d = 1 + (2 + 3) d
  - T(4) = (1 + (2+3)d) + 4d = 1 + (2+3+4)d
- Guess T(n) = 1 + (2+…+n) d
  = 1 + (n(n+1)/2 -1) d
- Exercise (offline): prove by induction
- Observe it is $\Theta(n^2)$ as expected.

Solving Recurrence

- General pattern
  1. Starting from the base case, use the recurrence to work out many cases, by directly substituting and working upwards in values of n
  2. Inspect the results, look for a pattern and make a hypothesis for the general results
  3. Attempt to prove the hypothesis – typically using some form of induction

  Often then extract the large n behavior using big-Oh family

  Can be long, tedious, and error-prone, but many cases are covered by a general rule with the name of “Master theorem” (next lecture)

Expectations

- Be able to extract recurrence relations from algorithms
    - Typically, used for recursive algorithms and especially "divide and conquer"

- Be able to explicitly solve (fairly simple) cases
    - Apply the recursion formula for sequence of (small) n
    - Guess pattern
    - Prove using induction
    - (You should generate multiple examples yourself and practice at solving them, and doing the induction proofs)