Lecture 4: Divide and Conquer

Divide and Conquer

- Merge sort is an example of a divide-and-conquer algorithm
- Recall the three steps (at each level) to solve a divide-and-conquer problem recursively
  - Divide problem into subproblems
  - Conquer problems by solving recursively
  - Combine solutions to solve the original problem
- **Recursive case**: when the subproblems are large enough to be solved recursively
- **Base case**: when subproblems are small enough that we don't need to use recursion any more, we say that the recursion bottoms out

Recurrences

- Recurrences are essential for the divide-and-conquer paradigm
  - Give a natural way to characterize the running times of divide-and-conquer algorithms
- Recurrence
  - An equation or inequality
  - Describes a function in terms of its value on smaller inputs
- Worst case running time for MERGE-SORT

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

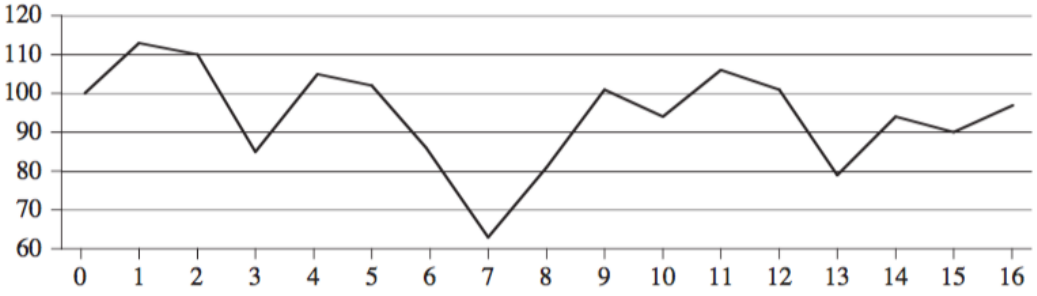
- This recurrence has solution: $T(n) = \Theta(n \lg n)$.
- There are many forms of recurrences
  - A recursive algorithm dividing subproblems into unequal sizes, 2/3 to 1/3 split, with divide and combine times linear
    - $T(n) = T(2n/3) + T(n/3) + \Theta(n)$.
  - Linear search, each subproblem has one element less than the original one
    - $T(n) = T(n-1) + \Theta(1)$.
- Three method for solving recurrences, for obtaining asymptotic Θ or O bounds on the solution
  - **Substitution method**
    - We guess a bound
    - Then use mathematical induction to prove our guess correct.
  - **Recursion-tree method**
    - Converts the recurrence into a tree
    - Nodes represent costs incurred at levels of recursion
    - Use techniques for bounding summations and solve the recurrence
  - **Master method**
    - Provides bounds for recurrences of the form $T(n) = aT(n/b) + f(n)$
    - $a \geq 1$, $b > 1$, $f(n)$ is a given function
    - This is a recurrence that solves a problem that was divided into a subproblems, each subproblem is $1/b$ of the original size, and divide and conquer and combine take $f(n)$ time
    - Has 3 cases, need to memorize
- Sometimes we will use recurrences that are inequalities
  - $T(n) \leq 2T(n/2) + \Theta(n)$, to get an upper bound on $T(n)$, with O-notation
  - $T(n) \geq 2T(n/2) + \Theta(n)$, to get a lower bound on $T(n)$, with Ω-notation
- Some technical details when we state and solve recurrences
  - In a call to MERGE-SORT with n elements, when n is odd we use subproblems of size $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$, we ignore this and treat it as if n were always even
  - Boundary conditions are usually ignored
    - i.e. algorithms for sufficiently small n have $T(n) = \Theta(1)$, then we omit statements on boundary conditions

    We often omit floors, ceilings, and boundary conditions

The Maximum-subarray Problem

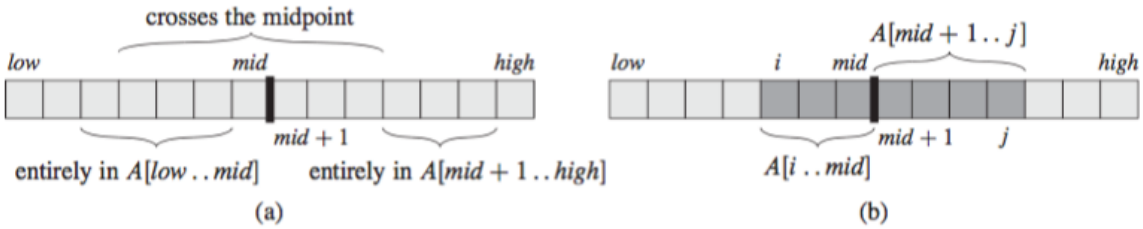
- Invest in the Volatile Chemical Corporation
  - Stock price is volatile
- Allowed to buy one unit of stock only one time and then sell it at a alter date
- Allowed to know what the price of the stock will be in the future
- **Goal**: maximize profit

- Lowest price at day 7
  - Buy at lowest price and sell at highest price? NOT always
- Highest profit: buy at 7 (day 2) and sell at 10 (day 3)
  - Day 2 is not the one with the lowest price, nor day 3 is the one with the highest price
- A Brute-force Solution
  - Try every possible pair of buy and sell dates (buy precedes sell date)
  - Period of n days has $\binom{n}{2}$ pairs of dates $\binom{n}{2}$ is $\Theta(n^2)$
  - Best we could hope is to evaluate each pair in constant time and approach would take $\Omega(n^2)$
  - Can we do better?
- A Transformation
  - Look at the problem in a different way
  - Find a sequence of days in which the net change from the first day to the last one is maximum
  - Treat the daily changes as an array A
  - Now we look for the nonempty, contiguous subarray of A whose values have the largest sum, the maximum subarray
  - Maximum subarray of $A[1..16]$ is $A[8..11]$ with sum 43
    - Buy just before day 8 (after day 7) and sell after day 11, earning profit of \$43 per share
  - Need to check $\binom{n-1}{2} = \Theta(n^2)$ subarrays for a period of n days
    - Bruteforce solution still takes $\Theta(n^2)$
- A Solution Using Divide-and-conquer
  - We have an array $A[low..high]$
  - In divide and conquer we divide the subarray into 2 subarrays of equal size as possible
    - Find midpoint of the subarray, and get 2 subarrays $A[low..mid]$ and $A[mid+1..high]$
  - But any contiguous subarray $A[i..j]$ of $A[low..high]$ must fall in:
    - Entirely in subarray $A[low..mid]$, $low \le i \le j \le mid$
    - Entirely in subarray $A[mid+1..high]$, $mid < i \le j \le high$
    - Crossing the midpoint, $low \le i \le mid < j \le high$.
  - A maximum subarray of $A[low..high]$ must have the greatest sum over all the three cases stated before
  - We can find subarrays of $A[low..mid]$ and $A[mid+1..high]$ recursively
  - Not for subarrays crossing the midpoint
    - We do it without recursion

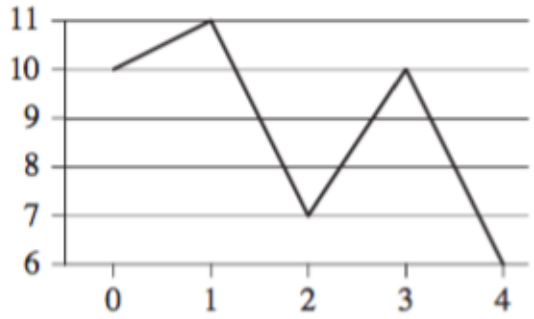


Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

Information about the price of stock in the Volatile Chemical Corporation after the close of trading over a period of 17 days. The horizontal axis of the chart indicates the day, and the vertical axis shows the price. The bottom row of the table gives the change in price from the previous day.

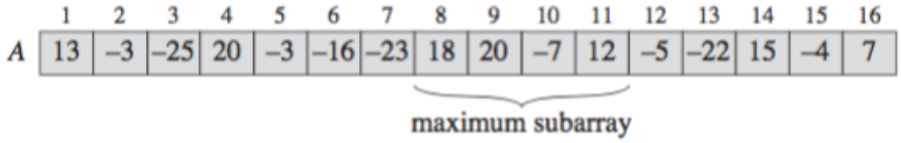


(a) Possible locations of subarrays of $A[low \ldots high]$ entirely in $A[low \ldots mid]$ entirely in $A[mid+1 \ldots high]$, or crossing the midpoint mid. (b) Any subarray of $A[low \ldots high]$ crossing the midpoint comprises two subarrays $A[i \ldots mid]$ and $A[mid+1 \ldots j]$, where $low \le i \le mid$ and $mid < j \le high$.



Day	0	1	2	3	4
Price	10	11	7	10	6
Change		1	−4	3	−4

An example showing that the maximum profit does not always start at the lowest price or end at the highest price. Again, the horizontal axis indicates the day, and the vertical axis shows the price. Here, the maximum profit of \$3 per share would be earned by buying after day 2 and selling after day 3. The price of \$7 after day 2 is not the lowest price overall, and the price of \$10 after day 3 is not the highest price overall.



The change in stock prices as a maximum-subarray problem. Here, the subarray $A[8 \cdots 11]$, with sum 43, has the greatest sum of any contiguous subarray of array A.

FIND-MAX-CROSSING-SUBARRAY$(A, low, mid, high)$

1. $left\text{-}sum = -\infty$
2. $sum = 0$
3. for $i = mid$ down to low
4. $\quad sum = sum + A[i]$
5. $\quad$ if $sum > left\text{-}sum$
6. $\quad\quad left\text{-}sum = sum$
7. $\quad\quad max - left = i$
8. $right\text{-}sum = -\infty$
9. $sum = 0$
10. for $j = mid + 1$ to $high$
11. $\quad sum = sum + A[j]$
12. $\quad$ if $sum > right\text{-}sum$
13. $\quad\quad right\text{-}sum = sum$
14. $\quad\quad max\text{-}right = j$
15. return$(max\text{-}left, max\text{-}right, left\text{-}sum + right\text{-}sum)$

- What is its running time?
  - Number of iterations: $(mid - low + 1) + (high - mid) = high - low + 1 = n$
- Lines 1,2 is the base case
- Lines 3-11 handle recurssive case
  - Line 3 does the divide part, computes mid
  - Left subarray $A[low..mid]$, right subarray $A[mid+1..high]$
  - Lines 4 and 5 do the conquer by recursion
  - Lines 6 to 11 are the combine part
    - Line 6 finds max subarray crossing the midpoint
    - Line 7 tests if left subarray contains max sum
    - Line 9 tests if right subarray contains max sum
    - Line 11 returns max subarray if crosses midpoint

FIND-MAXIMUM-SUBARRAY$(A, low, high)$

1. if $high == low$
2. $\quad$ return $(low, high, A[low])$ // base case: only one element
3. else $mid = \lfloor (low + high)/2 \rfloor$
4. $\quad (left\text{-}low, left\text{-}high, left\text{-}sum) =$
5. $\quad$ FIND-MAXIMUM-SUBARRAY(A, low, mid)
6. $(right\text{-}low, right\text{-}high, right\text{-}sum) =$
7. $\quad$ FIND-MAXIMUM-SUBARRAY$(A, mid + 1, high)$
8. $(cross\text{-}low, cross\text{-}high; cross\text{-}sum) =$
9. $\quad$ FIND-MAX-CROSSING-SUBARRAY$(A, low, mid, high)$
10. if $left\text{-}sum \ge right\text{-}sum$ and $left\text{-}sum \ge cross - sum$
11. $\quad$ return $(left\text{-}low, left\text{-}high, left\text{-}sum$
12. elseif $right\text{-}sum \ge left\text{-}sum$ and $right\text{-}sum \ge cross\text{-}sum$
13. $\quad$ return $(right\text{-}low, right\text{-}high, right\text{-}sum)$
14. else return $(cross\text{-}low, cross\text{-}high, cross\text{-}sum)$

Analyzing the Divide-and-conquer Algorithm

- Which recurrence describes the running time of the FIND-MAXIMUM-SUBARRAY procedure?
- Assumption: the original problem size is a power of 2, all subproblem sizes are integers
- $T(n)$, denotes running time of FIND-MAXIMUM-SUBARRAY on subarray of size n
- Line 1, constant time, line 2, constant time, line 3, constant time
  - Base case $T(1) = \Theta(1)$.
- Subproblems in recursive part (lines 4, 5) work on subarray of size $n/2$
  - We spend $T(n/2)$ time solving each, we solve 2 of them
  - We spend $2T(n/2)$
- FIND-MAX-CROSSING-SUBARRAY takes $\Theta(n)$
- Lines 7-11 take $\Theta(1)$ time
- Then, for the recursive case we have:
  - $T(n) = \Theta(1) + 2T(n/2) + \Theta(n) + \Theta(1)$
  - $= 2T(n/2) + \Theta(n)$.
- Combining base case and recursive case we have the recurrence:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

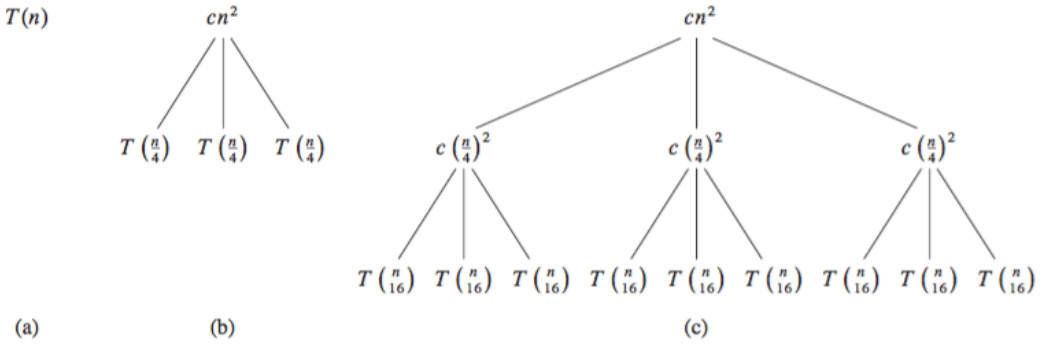
- The solution to this recurrence is: $T(n) = \Theta(n \lg n)$.

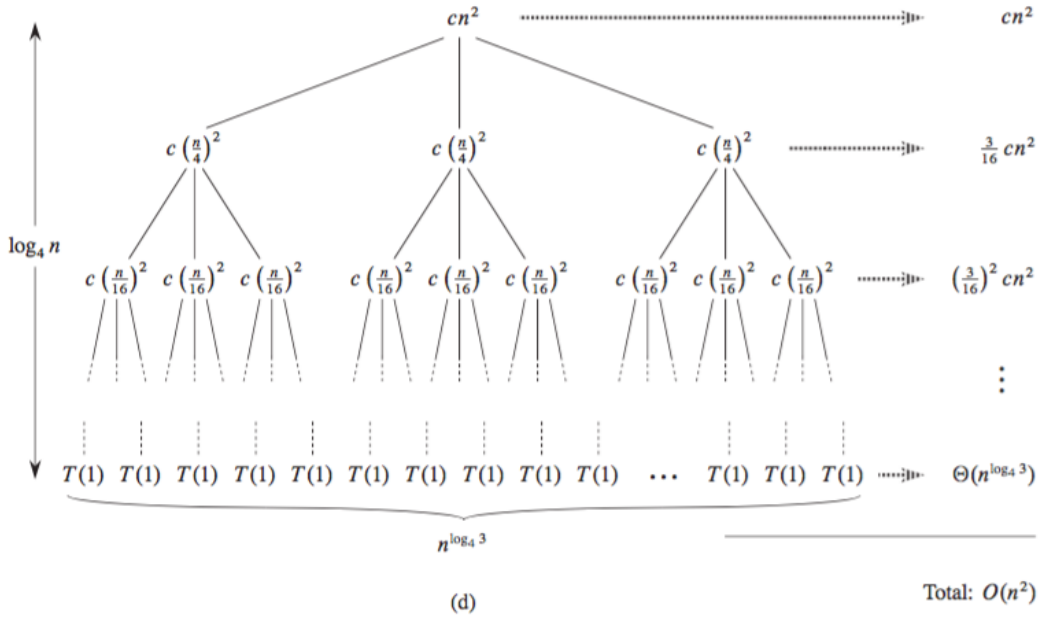
The Substitution Method

- Now we need to solve recurrences
- Substitution method
  - 2 steps
    - Guess the form of the solution
    - Use mathematical induction to find the constants and show that the solution works
  - Powerful method
  - Only applies in cases that the answer is easy to guess
  - To establish upper or lower bounds of a recurrence
- The Substitution Method: Example
  - For the recurrence $T(n) = 2T(\lfloor n/2 \rfloor) + n$.
  - We guess a solution, upper bound $T(n) = O(n \lg n)$ for $c > 0$
    - Assume this holds for $m < n$, in particular for $m = \lfloor n/2 \rfloor$
    - We prove $T(n) \leq cn \lg n$ for a constant $c > 0$
  - This yields $T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$.
  - We substitute into the recurrence: $T(n) \leq 2(c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n$
    - $\leq cn \lg(n/2) + n$
    - $= cn \lg n - cn \lg 2 + n$
    - $= cn \lg n - cn + n$
    - $\leq cn \lg n$
    - the last step holds if $c \geq 1$.
- We still need to prove that this solution is true for boundary conditions
- We do this by showing that boundary conditions are suitable for base cases of the inductive proof
- Choose constant c large enough for $T(n) \leq cn \lg n$ works for boundary conditions
- For asymptotic notation we require to prove $T(n) \leq cn \lg n$ for $n \geq n_0$, we choose n_0
- We avoid difficult boundary condition for $T(1) = 1$ for the induction test (not the recurrence)
  - In the inductive proof with $n = 1$, $T(1) \leq c1 \lg 1 = 0$, then it doesn?t correspond with $T(1) = 1$, and base case fails.
  - But we need to prove $T(n) \leq cn \lg n$ for $n \geq n_0$
  - And we are free to choose n_0
  - We remove the troublesome boundary by removing from the recurrence by seting $n_0 > 3$ and in the inductive proof with $n > 1$
  - And we replace $T(1)$ by $T(2)$ and $T(3)$ as base cases of the inductive proof
  - We make the base cases $T(2)$ and $T(3)$ instead of $T(1)$
  - Derive from the recurrence, given that $T(1) = 1$
    - $T(2) = 4$
    - $T(3) = 5$
  - $T(2) \leq c2 \lg 2 = 2c$, $c \geq 2$
  - $T(3) \leq c3 \lg 3 = 4.75c$, $c \geq 2$
- Making a Good Guess
  - There is no general way to guess correct solutions to recurrences
  - Requires experience and creativity
  - Some heuristics
    - Use recursion trees to generate good initial values
    - If a recurrence is similar to another one, use a similar solution

The Recursion-tree Method

- Visualize the iteration of a recurrence
  - Draw a recursion tree and obtain a good initial solution
  - We use the substitution method to proof
- Recursion tree
  - Each node represents the cost of a subproblem in the set of calls to recursive functions
  - We sum costs per level and determine the total cost of all levels of recursion
  - Useful when the recurrence describes execution time of a divide-and-conquer algorithm
- To solve recurrence $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$
- We create the recursion tree for $T(n) = 3T(n/4) + cn^2$
  - We include the coefficient $c > 0$
  - We assume that n is an exact power of 4
- The size of the problem decreases with the depth of the tree
  - Eventually, we reach the boundary condition
- How far from the root do we get?
  - The size of the subproblem for a node at depth i is $n/4^i$
  - The size gets to $n = 1$ when $n/4^i = 1$, or $i = \log_4 n$
  - Then, the tree has $\log_4 n + 1$ levels $0, 1, 2, \ldots, \log_4 n)$
- After we determine the cost for each level of the tree
  - Each level has 3 times more nodes than the previous level
  - The number of nodes at depth i is 3^i
  - Each node at depth i for $i = 0, 1, 2, \ldots, \log_4 n - 1$ has a cost of $c(n/4^i)^2$
  - Multiplying, we see that the cost of all nodes at depth i for $i = 0, 1, 2, \ldots \log_4 n - 1$ is $3^i c(n/4^i)^2 = (3/16)^i cn^2$
  - The last level at depth $log_4 n$ has $3^{\log_4 n} = n^{log_4 3}$ nodes, each with a cost of $T(1)$ with a total cost of $n^{log_4 3} T(1)$ with $\Theta(n^{log_4 3})$.





Constructing a recursion tree for the recurrence $T(n) = 3T(n/4) + cn^2$ Part (a) shows $T(n)$, which progressively expands in (b)–(d) to form the recursion tree. The fully expanded tree in part (d) has height $\log_4 n$ (it has $\log_4 n + 1$ levels).

- We sum the costs of all levels to get the cost of all the tree

$$\begin{aligned} T(n) &= cn^2 + \frac{3}{16}cn^2 + (\frac{3}{16})^2 cn^2 + \dots \\ &\quad \dots + (\frac{3}{16})^{log_4 n-1} cn^2 + \Theta(n^{log_4 3}) \\ &= \sum_{i=0}^{log_4 n-1} (\frac{3}{16})^i cn^2 + \Theta(n^{log_4 3}) \\ &= \frac{(3/16)^{log_4 n} - 1}{(3/16) - 1} cn^2 + \Theta(n^{log_4 3}). \end{aligned}$$

- We can use an infinite and decreasing geometric series as upper bound, equation A.6

$$\begin{aligned} T(n) &= \sum_{i=0}^{\log_4 n-1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\ &< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\ &= \frac{1}{1-(3/16)} cn^2 + \Theta(n^{\log_4 3}) \\ &= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \\ &= O(n^2) \end{aligned}$$

When the summation is infinite and $|x| < 1$, we have the finite decreasing geometric series

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

The Master Method

- Master method provides cookbook method for recurrences of the form
  - $T(n) = aT(n/b) + f(n)$
  - where $a \geq 1$ and $b > 1$ are constants
  - $f(n)$ is asymptotically positive
- The recurrence describes the execution time of an algorithm that
  - Divides a problem of size n into a subproblems
  - Each subproblem of size n/b
  - a and b are positive constants
  - The a subproblems are solved recursively
    - In time $T(n/b)$
  - The cost of dividing the problem and combining the results is given by $f(n)$
- The master theorem
  - Given $a \geq 1$ and $b > 1$ constants, given $f(n)$ a function
  - Given $T(n)$ defined by non-negative integers by recurrence
  - $T(n) = aT(n/b) + f(n)$
  - n/b can be $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$, then $T(n)$ has the following asymptotic bounds as:
- The Master Theorem:
  - **Case 1**: If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$ then $T(n) = \Theta(n^{\log_b a})$
  - **Case 2**: If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a \lg n})$
  - **Case 3**: If $f(n) = O(n^{\log_b +\epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(n^{\log_b a})$
  - In all cases we compare $f(n)$ with $n^{log_b a}$
    - The solution to the recurrence is dominated by the largest of the 2 functions
    - Case 1: $n^{log_b a}$ is the largest, the solution is $T(n) = \Theta(n^{log_b a})$
    - Case 3: $f(n)$ is larger, the solution is $T(n) = \Theta(f(n))$
    - Case 2: the two functions are of the same size, we multiply for a logarithmic factor, $T(n) = \Theta(n^{log_b a} \lg n) = \Theta(f(n) \lg n)$
  - Some technical aspects
    - In case 1:
      - $f(n)$ must be polinomically smaller than $n^{log_b a}$
      - Asymptotically smaller than $n^{log_b a}$ by a factor of n^ϵ for a constant $\epsilon > 0$
    - In case 3:
      - $f(n)$ must be polynomically larger than $n^{log_b a}$
      - It should also satisfy the regularity condition that $af(n/b) \leq cf(n)$ This condition is satisfied by most functions polynomically bounded that we can find.
    - the 3 cases don't cover all possibilities of $f(n)$

Example 1

Solve the recurrence:

$$g(1) = 10,\ g(N) = g(N-1) + 7,$$

for $N \geq 2$ and determine its complexity class.

Solution. For $N \geq 2$, we find

$$\begin{aligned} g(N) &= 7 + g(N-1) && \text{(step 1)} \\ &= 7 + 7 + g(N-2) && \text{(step 2)} \\ &= 7 + 7 + 7 + g(N-3) && \text{(step 3)} \\ &\cdots \\ &= \left(\textstyle\sum_{k=1}^{i} 7\right) + g(N-i) && \text{(step } i\text{)} \\ &\cdots \\ &= \left(\textstyle\sum_{k=1}^{N-1} 7\right) + g(1) && \text{(step } N-1\text{)} \\ &= 7(N-1) + 10. \end{aligned}$$

Since the following limit:

$$\lim_{N\to\infty} \frac{g(N)}{N} = \lim_{N\to\infty} \frac{7(N-1)+10}{N} = 7$$

is a nonzero constant, we find

$$g(N) = \Theta(N). \qquad \square$$

Example 2

Solve the recurrence:

$$g(1) = d,\ g(N) = g(N-1) + cN,$$

for $N \geq 2$ and determine its complexity class.

Solution. Assume that c and d are constant. For $N \geq 2$, we find

$$\begin{aligned} g(N) &= cN + g(N-1) && \text{(step 1)} \\ &= cN + c(N-1) + g(N-2) && \text{(step 2)} \\ &= cN + c(N-1) + c(N-2) + g(N-3) && \text{(step 3)} \\ &\cdots \\ &= cN + \left(\textstyle\sum_{k=1}^{i-1} c(N-k)\right) + g(N-i) && \text{(step } i\text{)} \\ &\cdots \\ &= cN + \left(\textstyle\sum_{k=1}^{N-2} c(N-k)\right) + g(1) && \text{(step } N-1\text{)} \\ &= cN + c\left(\textstyle\sum_{k=1}^{N-2} N\right) - c\left(\textstyle\sum_{k=1}^{N-2} k\right) + g(1) \\ &= cN + cN(N-2) - c(1/2)(N-2)(N-1) + d \end{aligned}$$

Since

$$\lim_{N\to\infty} \frac{g(N)}{N^2} = \lim_{N\to\infty} \frac{(1/2)cN^2 + (1/2)cN - c + d}{N^2} = \frac{1}{2}c$$

we find $g(N) = \Theta(N^2)$ whenever $c \neq 0$ and $g(N) = \Theta(1)$ whenever $c = 0$. $\square$

Example 3

Solve the recurrence:

$$g(0) = d,\ g(N) = g(N-3) + c,$$

for $N \geq 2$ and determine its complexity class.

Solution. Assume that c and d are constant. For simplicity, we assume that N is a multiple of 3, say $N = 3m$ for some integer $m \geq 1$. Then we find

$$\begin{aligned} g(N) &= c + g(N-3) && \text{(step 1)} \\ &= c + c + g(N-6) && \text{(step 2)} \\ &= c + c + c + g(N-9) && \text{(step 3)} \\ &\cdots \\ &= \left(\textstyle\sum_{k=1}^{i} c\right) + g(N-3i) && \text{(step } i\text{)} \\ &\cdots \\ &= \left(\textstyle\sum_{k=1}^{m} c\right) + g(0) = cm + d && \text{(step } m\text{)} \end{aligned}$$

Since

$$\lim_{N\to\infty} \frac{g(N)}{N} = \lim_{N\to\infty} \frac{c(N/3)+d}{N} = c/3$$

we find $g(N) = \Theta(N)$ whenever $c \neq 0$ and $g(N) = \Theta(1)$ whenever $c = 0$. $\square$

Example 4

(a) Solve the recurrence:

$$g(1) = 0,\ g(N) = g(N/2) + N^8$$

for $N \geq 2$.

Solution. For simplicity, assume N is a power of 2, say $N = 2^m$ for some integer $m \geq 1$. We find

$$\begin{aligned} g(N) &= N^8 + g(N/2) && \text{(step 1)} \\ &= N^8 + (N/2)^8 + g(N/4) && \text{(step 2)} \\ &= N^8 + (N/2)^8 + (N/4)^8 + g(N/8) && \text{(step 3)} \\ &\cdots \\ &= \left(\textstyle\sum_{k=0}^{i-1} (N/2^k)^8\right) + g(N/2^i) && \text{(step } i\text{)} \\ &\cdots \\ &= \left(\textstyle\sum_{k=0}^{m-1} (N/2^k)^8\right) + g(1) && \text{(step } m\text{)} \\ &= N^8 \textstyle\sum_{k=0}^{m-1} \left(\frac{1}{2^8}\right)^k \\ &= N^8 \left(\frac{1-(\frac{1}{2^8})^{\lg N}}{1-(\frac{1}{2^8})}\right) \\ &= \tfrac{256}{255} N^8 - \tfrac{256}{255} \end{aligned}$$

Since the following limit is a nonzero constant:

$$\lim_{N\to\infty} \frac{g(N)}{N^8} = \lim_{N\to\infty} \frac{\frac{256}{255}N^8 - \frac{256}{255}}{N^8} = \frac{256}{255} \qquad (\star)$$

we find $g(N) = \Theta(N^8)$. $\square$

(b) Is there a value of c such that $g(N) = \Theta(N^c)$? If so, what is c equal to?

Solution. Yes, $c = 8$ as proven in Eq. $(\star)$. Therefore, g is proportional to a polynomial. □

(c) Is there a value of d such that $g(N) = \Theta(d^N)$? If so, what is d equal to?

Solution. Clearly, $d \neq 1$ because $\Theta(1) \neq \Theta(N^8)$. Assume there exists a $d \neq 1$ such that $g(N) = \Theta(d^N)$. Then

$$\lim_{N\to\infty} \frac{g(N)}{d^N}$$

is a nonzero constant L. However we find that

$$L = \lim_{N\to\infty} \frac{g(N)}{d^N}$$

$$= \lim_{N\to\infty} \frac{\frac{256}{255}N^8 - \frac{256}{255}}{d^N}$$

$$= \begin{cases} 0 & \text{if } d > 1 \\ \infty & \text{if } 0 < d < 1 \end{cases}$$

(after applying L'Hopital's rule eight times in the case $d > 1$). From this contradiction we conclude that $g(N)$ is not proportional to any exponential (nor constant). □

Example 5

Solve the recurrence:

$$g(1) = 1,\ g(N) = (1/\alpha)g(N/2),$$

for $N \geq 2$ where α is a constant and determine its complexity class.

Solution. For simplicity, assume N is a power of 2, say $N = 2^m$ for some integer $m \geq 1$ and let $\beta = (1/\alpha)$. We find

$$\begin{aligned} g(N) &= \beta g(N/2) && \text{(step 1)} \\ &= \beta^2 g(N/4) && \text{(step 2)} \\ &= \beta^3 g(N/8) && \text{(step 3)} \\ &\cdots \\ &= \beta^i g(N/2^i) && \text{(step } i\text{)} \\ &\cdots \\ &= \beta^m g(1) && \text{(step } m\text{)} \\ &= \beta^m \\ &= \left(\tfrac{1}{\alpha}\right)^{\lg N} = N^{\lg(1/\alpha)} \end{aligned}$$

Therefore, $g(N) = \Theta(N^{\lg(1/\alpha)})$. □

Example 6

Solve the recurrence:

$$g(1) = 10,\ g(N) = (g(N/2))^5$$

for $N \geq 2$.

Solution. For simplicity, assume N is a power of 2, say $N = 2^m$ for some integer $m \geq 1$. We find

$$\begin{aligned} g(N) &= (g(N/2))^5 && \text{(step 1)} \\ &= (g(N/4))^{25} && \text{(step 2)} \\ &= (g(N/8))^{125} && \text{(step 3)} \\ &\cdots \\ &= (g(N/2^i))^{5^i} && \text{(step } i\text{)} \\ &\cdots \\ &= (g(1))^{5^m} && \text{(step } m\text{)} \\ &= 10^{5^{\lg N}} \\ &= 10^{N^{\lg 5}} \end{aligned}$$

Therefore, $g(N) = \Theta(10^{N^{\lg 5}})$. □