• **Main Topics**
  - Asymptotic complexity: $O$, $\Omega$, and $\Theta$.
  - Designing divide-and-conquer algorithms.
  - Setting up runtime recurrences.
  - Solving recurrences using Master theorem (other methods are optional).
  - Faster multiplication.

• **NOT included:**
  - Recursion trees that are not covered by master theorem.
  - Other methods for solving runtime recurrences such as guess-and-check.

• **Proving big-$O$ bounds** (Homework 1, Problem 1. Ex 0.1 in Textbook):
  - If $f_1 = O(g_1(n))$, $f_2 = O(g_2(n))$, then
    * $f_1f_2 = O(g_1g_2)$.
    * $f_1 + f_2 = O(\max\{g_1, g_2\})$.
  - For any constants $a$, $b$, and $c > 1$, $O(\log^a(n)) \leq O(n^b) \leq O(c^n)$,
  - $\ln(n) = \Theta(\log n) = \Theta(\log_2 n) = \Theta(\log_c n)$.

• **Divide-and-conquer and setting up running time recurrences** (Homework 1, Problems 2 and 3. Ex 2.12, 2.16, 2.17, 2.23 in textbook)
  - General structure of a recursive algorithm:
    * Split the problem up.
    * Make $a$ recursive calls to problems of size $n/b$
    * Combine the results.
  - $T(n)$: running time when given input of size $n$.
  - If total cost of split/combine is $O(n^d)$, runtime recurrence is:
    $$T(n) = aT(n/b) + O(n^d).$$
• Master Theorem:

The recurrence:

\[ T(n) = a \cdot T\left(\frac{n}{b}\right) + c \cdot n^d, \quad T(1) = e \]

where \(a > 0, b > 1, c > 0, d \geq 0\) and \(e \geq 0\) are constants, has the solution given below:

**Case 1:** If \(d = \log_b a\) then \(T(n) = O(n^d \log n)\).

**Case 2:** If \(d > \log_b a\) then \(T(n) = O(n^d)\).

**Case 3:** If \(d < \log_b a\) then \(T(n) = O(n^{\log_b a})\).

• Example of using Master theorem to analyze recursion (Homework 1 Problems 2ac, 3. Ex 2.4, 2.5abcde in textbook):

• Fast multiplication of 2 \(n\)-digit numbers using 3 multiplies of \(n/2\)-digit numbers plus \(O(n)\) overhead:

  - Runtime recurrence: \(T(n) = 3T(n/2) + O(n)\).
  - Fits into the Master theorem with \(a = 3, b = 2, d = 1\).
  - \(d < \log_2 3\), total runtime \(O(n^{\log_2 3}) = O(n^{1.59})\).

### Additional Exercises and Solutions from Previous Years

1. Let \(A\) and \(B\) be two matrices to be multiplied. Strassen’s algorithm (which is in the text book, but whose detail is not important for this problem) takes time \(O(n^{\log_2 7})\). Suppose someone discovers a way to obtain the product of two order \(n \times n\) matrices by doing 24 multiplications of two matrices of order \(n/5\) and combining the results in \(O(n^2)\) time. Write the recurrence for the running time of this new algorithm. What is the solution to this recurrence? Is this running time better than the running time for Strassen’s algorithm?

2. (From Fall 2015 Test 1) Let \(x\) be an \(n\)-bit number. The following recursive algorithm computes \(x^k\) for some integer \(k\). (Assume \(k\) is a power of 2.) It uses a bit-multiplication algorithm \(\text{MULTIPLY}(y,z)\) that takes two numbers \(y\) and \(z\) and returns their product. The numbers \(y, z\) and their product are all in bits.

\[ \text{POWER}(x, k) \]

  IF \(k = 1\) then Return \((x)\)
  IF \(k > 1\) Then
Let $y = \text{POWER}(x, k/2)$
Return (MULTIPLY $(y, y)$)

Assuming that the number of bit operations for multiplying two $n$ bit numbers is $n^2$, set up a recurrence for the number of bit operations used by this algorithm to compute $x^n$ and solve the recurrence.

3. (From Fall 2015 Test 1, also on Homework 1, Problem 3) Assume that you are given an $O(n)$-time algorithm $\text{MEDIAN}$ that takes as input an array $A$ of $n$ distinct positive integers and returns the median element in $A$.

Using the algorithm $\text{MEDIAN}$ design an $O(n)$ algorithm that, given an array $A$ of $n$ distinct positive integers and an index $1 \leq k \leq n$, determines the $k$-th smallest element in $A$.

(The median element in $A$ is the $\lceil n/2 \rceil$-th smallest element of $A$.)

4. Given as input an array $A$ of $n$ integers, describe an $O(n \log n)$ time algorithm to decide if the entries of $A$ are distinct. Why does your algorithm run in time $O(n \log n)$?

5. Given a sorted array of $n$ distinct integers $A[1] \cdots A[n]$, describe an $O(\log n)$ divide-and-conquer algorithm to find out whether there is an index $i$ such that $A[i] = i$. Why does your algorithm run in the claimed time bound?

6. You are given an infinite array $A$ in which the first $n$ cells contain integers in sorted order and the rest of the cells are filled with $\infty$. You are not given the value of $n$. Describe an $O(\log n)$ algorithm that takes an integer $x$ as input and finds a position in the array containing $x$, if such a position exists.

7. Describe an $O(n \log n)$ time algorithm that, given a set $S$ of $n$ real numbers and another real number $x$, determines whether or not there exist two elements in $S$ whose sum is exactly $x$.

(Hint: Doing a binary search in a sorted list can be done in $O(\log n)$ time.)

8. Let $A_1, A_2, \cdots, A_k$ be $k$ sorted arrays, each with $n$ elements. Give an $O(nk \log k)$ algorithm to combine them into a single sorted array of $kn$ elements. (Assume $k$ is a power of 2.)

9. Let $A$ be an array of $n$ positive integers. Let $n$ be a multiple of 5. Describe an $O(n)$ algorithm to find if there is an element in $A$ that occurs at least $n/5$ times in $A$. (Hint: Use the linear time algorithm $\text{SELECTION}$ for computing the $k$-th smallest element discussed in class.)