1. Preliminaries. We say \( f(n) = O(g(n)) \) if \( f(n) \leq cg(n) \) for all \( n \geq n_0 \) for some constants \( c \) and \( n_0 \). We say \( f(n) = \Omega(g(n)) \) if and only if \( g(n) = O(f(n)) \). Also, \( f(n) = \Theta(g(n)) \) if \( f(n) = O(g(n)) \) and \( f(n) = \Omega(g(n)) \).

A partial list of functions in increasing order is: \( O(1), (\log n)^{\epsilon}, \log n, (\log n)^{1+\mu}, n^\epsilon, n^{1+\mu}, 2^n, 2^{n^{1+\mu}} \) where \( 0 < \epsilon < 1 \) and \( \mu > 0 \) are constants.

Stirling’s approximation: \( n! \approx (n/e)^n \sqrt{2\pi n} \).
\[
\sum_{i=1}^{n} i = n(n + 1)/2. \sum_{i=1}^{n} i^2 = n(n + 1)(2n + 1)/6. \sum_{i=1}^{n} i^3 = n^2(n + 1)^2/4. \]

2. Master Theorem. Consider the recurrence relation: \( T(n) = aT(n/b) + f(n) \), where \( a \geq 1 \) and \( b > 1 \) are constants. **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 \( n^{\log_b a} = \Theta(f(n)) \), then \( T(n) = \Theta(n^{\log_b a}) \). **Case 3:** If \( f(n) = \Omega(n^{\log_b a + \epsilon}) \) for some constant \( \epsilon > 0 \) and \( af(n/b) \leq cf(n) \) for some constant \( c < 1 \), then, \( T(n) = \Theta(f(n)) \).

3. Randomized Algorithms. A Monte Carlo algorithm runs for a prespecified amount of time and its output is correct with high probability. By high probability we mean a probability of \( 1 - n^{-\alpha} \), for any constant \( \alpha \). A Las Vegas algorithm always outputs the correct answer and its run time is a random variable. We say the run time of a Las Vegas algorithm is \( \tilde{O}(f(n)) \) if the run time is \( \leq c\alpha f(n) \) for all \( n \geq n_0 \) with probability \( \geq (1 - n^{-\alpha}) \) for some constants \( c \) and \( n_0 \).

4. Dictionaries and Priority Queues: A dictionary supports the operations: SEARCH (for an arbitrary element), INSERT (an arbitrary element), and DELETE (an arbitrary element). A (max) priority queue supports: INSERT (an arbitrary element), SEARCH (for the maximum element), and DELETE (the maximum element).

5. Hashing with Chaining: Here we employ an array \( a[1 : m] \) of lists and a hash function \( h(.) \). Any element \( x \) will be inserted into the list \( a[h(x)] \).

INSERT, DELETE, and SEARCH operations take \( O(1 + \alpha) \) time on the average, where \( \alpha = n/m \). In the worst case each operation takes \( \Omega(n) \) time.

6. Binary Search Trees: A binary search tree (BST) is a binary tree where each node has a key. Key at each node is greater than any key in its left subtree and smaller than any key in its right subtree. A BST can be used to support a dictionary as well as a priority queue. Each operation of interest (SEARCH, INSERT, DELETE) will take \( O(h) \) time to process, where \( h \) is the height of the tree.

The expected height of a BST on \( n \) nodes is \( O(\log n) \). In the worst case the height can be \( \Omega(n) \).

7. Heaps and Heapsort: A (max) heap is a complete binary tree where a key is stored at each node. The key at any node will be greater than the keys of its children.

A (max) heap supports the following operations: SEARCH (for the maximum), INSERT (an arbitrary element), and DELETE (the maximum). Each operation can be completed in \( O(\log n) \) time, \( n \) being the number of elements in the heap. A heap can be used to sort elements. Heapsort on \( n \) elements takes \( O(n \log n) \) time.

Given a sequence of numbers we can form a heap out of these elements using the Heapify algorithm. If called on a tree of height \( h \), Heapify takes \( O(h) \) time to complete. A heap out of \( n \) elements can be formed in \( O(n) \) time.

8. A 2-3 Tree: can be used to support a dictionary as well as a priority queue. Each operation of interest will take \( O(\log n) \) time in the worst case.

9. Mergesort sorts \( n \) arbitrary keys in \( O(n \log n) \) time. Quicksort sorts \( n \) keys in an expected run time of \( O(n \log n) \). Its worst case run time is \( O(n^2) \).