Math (L2)

Geometric series

$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$ whenever $|x| < 1$.

Harmonic series

$\sum_{k=1}^{n} \frac{1}{k} \sim \log n$

Stirling

$\log n! \in \Theta(n \lg n)$

Binomial theorem

Master Theorem (L2)

Suppose $T(n) = aT(n/b) + f(n)$.

For any $\epsilon > 0$, - if $f(n) \in O(n^{\log_b a - \epsilon})$, then $T(n) \in \Theta(n^{\log_b a})$; - if $f(n) in \Theta(n^{\log_b a})$, then $T(n) \in \Theta(n^{\log_b a} \log n)$; - if $f(n) in \Omega(n^{\log_b a + \epsilon})$, then $T(n) \in \Theta(f(n))$.

Binary trees (L3)

- a complete binary tree with n levels has $2^n - 1$ nodes
- a pre-heap is the result of deleting zero r more leaves consecutively from the right of a complete binary tree
- represent pre-heap as an array A with length equal to number of nodes, so that
  - root has index 1
  - the left, right children have indices $2n$, $2n + 1$
- a preheap with H levels has at least 2^{H-1} and at most $2^H - 1$ nodes
- a preheap with H levels has at most 2^{H-1} leaves, because
  - it has at most $2^H - 1$ nodes, and
  - a pre-heap with n nodes has $\lceil n \rceil$ nodes

Heaps (L3)

- a heap is a labeled binary tree, such that
  - as a tree it's a pre-heap, and
  - the key of each node is greater than those of its children
- operations are
  - HEAPIFY(A,i)
    - transforms a preheap A so that the subtree rooted at i is a heap
    - swaps A[i] with the larger of left/right child if it's larger than A[i]„ then recurses

        * running time is $O(\log n)$ where n is the length of A
    - BUILD_HEAP(A)
        * transforms preaheap A into a heap
        * calls HEAPIFY(A,i) iteratively backward from i=len(A)/2 to i=1
        * running time is $O(n)$ where n is the length of A
- heap data structure supports sorting method
    - HEAPSORT(A)
        * sorts pre-heap A
        * implementation
            · BUILD_HEAP(A)
            · from i=A.len to i=2
            · exchange A[i], A[1]
            · decrement heap-size
            · call HEAPIFY(A, 1)
    * running time O(n $\log n$)

Priority Queues (L3)

- generally useful ADT with these operations
    - INSERT, MAX, EXTRACT_MAX, INCREASE_KEY
- heap implementation like this
    - EXTRACT_MAX
        * swap first and last elements
        * decrement length
        * heapify on root
        * return last element
    - INCREASE_KEY
        * assign new (larger) key
        * while larger than parent: exchange with parent
    - INSERT
        * append to array, then "float up" as in INCREASE_KEY

Quicksort (L4)

- another sorting routine
    - implementation of QUICKSORT(A,p,r) like this:
        * while p < r:
            · q = PARTITION(A,p,r)
            · QUICKSORT(A,p,q-1)
            · QUICKSORT(A,p,q+1)
    - implementation of PARTITION(A,p,r)
        * take A[r] to be the "pivot element"
        * set left-right boundary to be 0
        * for i = 1 to r-1:
            · if A[i] < pivot
            · increment left-right boundary

        - swap A[i] with the value at the new left-right boundary
    * swap A[r] with the first value after the left-right boundary

Lower bound on comparison sorting (L5)

- any sort implemented with a binary decision tree has worst-case Omega(nlog n)

Bucketsort (L6)

- suppose A contains n integers uniformly drawn from 1..M with M>n
- allocate values into n buckets, each upper bounded by M/n, 2M/n .. nM/n
- this has running time O(n)
    - basically, constant time to put each number in its bucket
    - and constant time to sort each bucket

Medians and Order statistics (L6)

- to get cth largest/smallest entry for constant c
- can use HEAPIFY(A), then EXTRACT_MAX/MIN for O(log n)
- but to get, e.g., median, this is O(nlog n), $somightaswelljustsortRANDOMIZED\_SELECT(A, i)uses1$
-     - this runs in O(n) time

Graphs, paths and trees (L7)

- a path is a sequence of vertices of a graph, connected by edges
- it is simple iff it contains no vertex more than once
- a loop is a path with same first and last vertex
- a simple loop is a path containing at least 3 nodes such that
    - its first (=last) node occurs exactly twice
    - no other node occurs more than once
- a tree is a connected undirected graph with no simple loops
    - there is exactly one simple path from any node to any other
- a rooted tree is a tree with one vertex distinguished as its root
    - node x is ancestor of node y if the path from root to y contains x

Traversing binary trees (L7)

- modes of traversal
    - preorder: node, then left, then right
        * lambda f(x): g(x) f(x.left) f(x.right)
    - inorder: left, then node, then right
        * lambda f(x): f(x.left) g(x) f(x.right)
    - postorder: right, then node, then left
        * lambda f(x): f(x.left) f(x.right) g(x)

Binary Search Trees (L7)

- a binary search tree is a binary tree such that
  - each descendant of the left child of x is no greater than x
  - each descendant of the right child of x is no less than x
- the successor of a node x is the node after x according to inorder traversal
- facts about successor:
  - succ(x) is either an ancestor or a descendant of x
  - if x has a right child, then succ(x) is the leftmost in the right subtree rooted at x
  - otherwise, if x has a successor, then this is the least ancestor of x such that x descends from the left child of x
  - the operations FIND, MIN, MAX, PRED, SUC are all O(h) for h the height
  - the expected cost of search is O(log(n)) for n the number of nodes