Algorithms Review

Introduction

- **Insertion Sort Analysis**
  - **Proving correctness using loop-invariants**
  - **RAM model of computation**
  - **We discussed how we are normally only interested in growth of running time:**
    - » **Best-case linear in O(n), worst-case quadratic in O(n^2)**
- **Divide and Conquer**
  - **General Method**
  - **Merge Sort**
- **Analysis of divide-and-conquer algorithms**
  - **Recurrence Equation**
  - **Merge Sort Analysis**
    - » **Proof by Picture of Recursion Tree、Telescoping、Induction**
    - » **Θ(nlgn) for all cases**

Asymptotic Growth

- **Asymptotic Growth**
  - **O-notation**
  - **Ω-notation**
  - **Θ-notation**
- **Recurrences**
  - **Substitution method**
  - **Recursion-tree method**
  - **Master method**

Sorting

- **Heap Sort**
  - **Heaps**
  - **MAX-HEAPIFY**
  - **BUILD-MAX-HEAP**
  - **HEAPSORT**
- **Priority Queues**
  - **MAXIMUM(S)**
  - **EXTRACT-MAX(S)**
  - **INCREASE-KEY(S,x,k)**
  - **INSERT(S, x)**
  - **Applications**

Sorting

- **QuickSort**
  - **Divide and Conquer**
  - **PARTITION**
  - **Analysis of QuickSort**
    - » **Worst case $\Theta(n^2)$**
    - » **Expected running time: $\Theta(n \lg n)$**
  - **Randomized quicksort**
- **The best worst-case running time for comparison sorting**
  - **Decision-tree Model**
  - **Theorem. Any comparison sorting algorithm requires $\Omega(n \lg n)$ comparisons in the worst case**

Sorting in linear time

- **Counting sort**
  - **How if there are *17* elements not greater than *x* in A?**
    - » **Put the *last one* in position *17*, the *penultimate one* in position *16*,…**
  - **If $k = O(n)$, then counting sort takes $\Theta(n)$ time.**
- **Radix sort**
  - **Sort on least-significant digit first with auxiliary stable sort.**
  - **$\Theta(d\ T(n))$, if we use counting sort and d is constant, $\Theta(n)$**
- **Bucket sort**
  - **Divide the interval [0,1) into n equal-sized subintervals, or bucket, and then distribute the n input number into the buckets.**
  - **$\Theta(n)$ under uniform distribution**

Medians and Order Statistics

- **Order Statistics**
  - **Expected linear time selection**
    - » **Main idea: PARTITION**
    - » **Worst-case $\Theta(n^2)$**
  - **Worst-case linear time selection**
    - » **Generate a good pivot recursively.**

Other D&C

- **Square Matrix Multiplication**
  - **Simple Divide and Conquer algorithm**
    - » $\Theta(n^3)$
  - **Strassen's algorithm**
    - » **8 recursive multiplications of $n/2 \times n/2$ matrices to only 7.**
    - » $\Theta(n^{lg7})$
- **The maximum-subarray problem**
  - **Stock Investing Problem**
  - **Divide and Conquer algorithm**
    - » **3 sub-problems, one of which is not recursive**
    - » $\Theta(nlgn)$

Dynamic Programming

- **Assembly Lines**

$$f_1[j] = \begin{cases} e_1 + a_{1,1} & \text{if } j = 1 \\ \min(f_1[j\text{-}1] + a_{1,j},\ f_2[j\text{-}1] + t_{2,j\text{-}1} + a_{1,j}) & \text{if } j \geq 2 \end{cases}$$

$$f_2[j] = \begin{cases} e_2 + a_{2,1} & \text{if } j = 1 \\ \min(f_2[j\text{-}1] + a_{2,j},\ f_1[j\text{-}1] + t_{1,j\text{-}1} + a_{2,j}) & \text{if } j \geq 2 \end{cases}$$

- **Matrix-chain multiplication**

$$m[i, j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i\text{-}1} p_k p_j \} & \text{if } i < j \end{cases}$$

- **Elements of DP Algorithms**
  - **Optimal Substructure**
  - **Overlapping Subproblem**
  - **Reconstructing an optimal solution**

Dynamic Programming

- **Memoization**
- **Longest Common Subsequence**

$$c[i,j] = \begin{cases} c[i-1, j-1] + 1 & \text{if } i,\ j > 0 \text{ and } x_i = y_j \\ \max(c[i,\ j-1], c[i-1,\ j]) & \text{otherwise} \end{cases}$$

- **Max Sum**
  - **b[j]=max(b[j-1]+a[j], a[j]), 1<=j<=n.**

Greedy

- **Activity-selection problem**
  - **Optimal Substructure**
  - **Greedy algorithm**
- **Elements of the greedy strategy**
  - **Greedy-choice property and Optimal substructure**
  - **Fractional Knapsack Problem**
  - **0-1 Knapsack Problem (dynamic programming)**
- **Huffman codes**

Single-source shortest paths

- **Single-source shortest paths**
  - **Nonnegative edge weights**
    - » **Dijkstra's algorithm: O(E+VlgV)**
  - **General**
    - » **Bellman-Ford: O(VE)**

All-pairs shortest paths

- **All-pairs shortest paths (similar to matrix multiply)**
  - $d^{(m)}_{ij} = \min_k\{d^{(m-1)}_{ik}+a_{kj}\}$
  - **(min,+) multiplication**
  - **Improved matrix multiplication algorithm.**
- **Floy-Warshall algorithm**
  - $c_{ij}^{(k)}=\min_k\{c_{ij}^{(k-1)},c_{ik}^{(k-1)}+c_{kj}^{(k-1)}\}$
- **Johnson's algorithm**
  - **Graph reweighting**
    - » $h: V \rightarrow R$, **reweight** $(u,v) \in E$ **by** $w_h(u,v)=w(u,v)+h(u)-h(v)$.
  - **Algorithm:**
    - » **Find a function** $h: V \rightarrow R$, **such that** $w_h(u,v) \geq 0$ **for all** $(u,v) \in E$ **by using Bellman-Ford**
    - » **using** w_h **from Run Dijkstra's algorithms each vertex** $u \in V$
    - » **For each** $(u,v) \in V*V$, **compute** $\delta(u,v)=\delta_h(u,v)-h(u)+h(v)$

Back Tracking

- **Back Tracking Paradigm**
  - **A design technique, like divide-and-conquer.**
  - **Useful for optimization problems and finding feasible solutions.**
  - **Constraints**
    - » **Explicit Constraints**
    - » **Implicit Constraints**
  - **General Method**
    - » **Identify space state tree → Generate problem state → Is solution state? → Is answer state?**
    - » **Searching in DFS way**
- **N Queen problem**
  - **Algorithm**
  - **Bound function**

Branch and Bound Algorithms

- **General Method**
- **Least Cost Search**
  - $\hat{c}(X)$: $\hat{c}(X) = f(h(X)) + \hat{g}(X)$

Computability

- **Computability**
  - **Undecidable Problem**
    - » **Hilbert's 10th Problem.**
    - » **Post's Correspondence Problem.**
    - » **Halting Problem.**
  - **NP-Complete**
    - » **P, EXP, NP, NP-Complete**
    - » **SAT is the first NP-Complete problem**
    - » **Other NP-Complete problems: 3-color, TSP, …**

Sample Problem

- **Answer T/F for the following:**
  - **Because inserting a key into a binary heap requires (*lgn*) time in the worst case, building a heap of size *n* from scratch requires (*nlgn*) time in the worst case.**
- **Single Choice**
  - **The worst-case running time of Insertion Sort is (   )**

    A. $\Theta(n^2)$ **B.** $\Theta(nlgn)$

    C. $\Theta(n)$ **D.** $\Theta(n^3)$
- **Evaluate the following recursions**
  - **T(n) = T(7n/8)+n**

Sample Problem

- **Find a maximum-length common subsequence of *X* and *Y* with sequence *X* = <*A*, *B*, *C*, *B*, *D*, *A*, *B*> and *Y* = <*B*, *D*, *C*, *A*, *B*, *A*>.**

- **Comprehensive**
  - **Strategy?**
  - **Sorting?**
- **Design?**