Review For the Final Exam

Exam Content

1. Time complexity calc
2. Sort and divide-and-conquer
3. Dynamic programming
4. Greedy algorithms
5. Search algorithms
6. Amortized analysis
7. Graph theory
8. String algorithms

> **Note**:
>
> According to reliable information, pseudocode is not required except for divide-and-conquer, dynamic programming and greedy algorithms, and this final review is extreamly meant to get high points regardless of wether you turely understand these knowledges or not or if you will be capabel of using these knowledges IRL. This review is ONLY targeted at PONTS ON PAPER.

Being able to wirte good code in real life has nothing to do with getting high points in exams.

*said by **me***

Time Complexity Calculation

Notations

First of all, we need strict mathematical definitions of the $O, \Omega, \Theta, o, \omega, \theta$ notations to further discuss the time complexity of the algorithms.

***Definition* 1: The O-notation**

$$O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0\}$$

***Definition* 2: The o-notation**

$$o(g(n)) = \{f(n) : \text{for any positive constant } c > 0, \text{there exists a constant } n_0 > 0 \text{ such that } 0 \le f(n) < cg(n) \text{ for all } n \ge n_0\}$$

***Definition* 3: The Θ-notation**

$$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2 \text{ and } n_0 \text{ such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0\}$$

***Definition* 4: The Ω-notation**

$$\Omega(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0\}$$

***Definition* 5: The ω-nonation**

$$\omega(g(n)) = \{f(n) : \text{for any positive constant } c > 0, \text{there exists a constant } n_0 > 0 \text{ such that } 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0\}$$

Solving the Recurrences

Subsitution Method

1. Guess the solution
2. Subsitutite and prove through induction

Recurtion-tree Method

1. Draw the recurtion tree
2. Sum up the nodes at the same depth
3. Calculate the tree height
4. Sum up all costs

Master Method

***Theorem* 1: The Master Theorem**

For recurrences like $T(n) = a\left(\frac{T}{b}\right) + f(n)$:

1. If $f(n) = O\left(n^{\log_b (a-\varepsilon)}\right)$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n \log_b a)$.
2. If $f(n) = \Theta\left(n^{\log_b a}\right)$, then $T(n) = \Theta\left(n^{\log_b a}\right)$.
3. If $f(n) = O\left(n^{\log_b (a+\varepsilon)}\right)$ for some constant $\varepsilon > 0$, and if $af\left(\frac{n}{b}\right) \le cf(n)$ for some constant $c < 1$ and all sufficiently largh n, then $T(n) = \Theta(f(n))$.

Sort and divide-and-conquer

Divide-and-Conquer

Steps of divide-and-conquer:

- Divide: Divide the problem into subproblems

- Conquer: Solve the subproblems recursively. If the subproblems are small enough, solve them in a straightforword manner.
- Combine: Combine the solutions to the subproblems to construct the solution to the original problem.

Merge Sort

Chess Board Cover

Multiplication of Big numbers

Linear Time Select (PPT)

Dynamic Programming

Proof is not required in the final, so we only need to know the problem it can solve and the procedure of applying dp.

The key to the solution to the problems is the clearly defined optimal structure and the recursive solution, or simply, the equation.

Knapsack Problem

Matrix Chain Multiplication

Longest Common Subsequence

Optimal Binary Tree

Greedy algorithms

Greedy algorithms can only solve problems which the optimal solution can be obtained by adding the optiaml solution **at the moment** together.

Activity Selection Problem

Huffman Codes

Minimum Spanning Tree

Search Algorithms

DFS and BFS

DFS using stack, BFS using queue.

Hill Climbing

Similar to gradient desent.

Best-First

Creat a heap that satisify a estimate function, use the heap to determine the search order. Priority queue.

Branch and Bound

Pruning branches that cannot lead to the optimal solution in advance to reduce cost.

A^* Algorithm

Staffing Problem

Travelling Salesman Problem