Ph.D. Comprehensive Examination
Design and Analysis of Algorithms

Main Books

1. Cormen, Leiserton, Rivest, Introduction to Algorithms, MIT Press, 2001.

Additional Books

1. Kenneth H. Rosen, Discrete mathematics and its applications, McGraw-Hill, 1991.
2. Horowitz, Sahni, Rajasekaran, Computer Algorithms, Computer Science Press, 1998.
3. Weiss, Data Structures and algorithm and Algorithm Analysis in C, Addison Wesley, 1997.
4. Sahni, Data Structures, Algorithms, and Applications in C++, McGraw-Hill, 1998.
5. R. Garnier and J. Taylor, Discrete Mathematics for New Technology, Adam Hilger Publishing.

Exam Rules

1. The exam takes about 3 hours.
2. The exam is closed book. However, you can bring a cheat sheet (you can use both sides of an A4 paper)

Topics:

(Just to give you an idea)

1. Introduction
    a. What is an algorithm
    b. Analysis of Algorithm
        i. Model
        ii. Types of complexities
    c. Examples
        i. Matrix operations
        ii. Polynomial evaluation
        iii. GCD Algorithm
2. Mathematical foundations
    a. Functions
    b. Set theory
    c. Logic
    d. Boolean Algebra
        i. Boolean Functions
        ii. Representing Boolean Functions
    e. Logarithm
    f. Probability
    g. Summations
    h. Asymptotic notations
    i. Recursive algorithms and recurrence relations
        i. counting
    j. Solving recurrence relations
        i. Substitution
        ii. Generating Functions
        iii. Linear Difference Equation
    k. Mathematical Reasoning
        i. Methods of Proof
        ii. Mathematical Induction
        iii. Recursive Definitions
    l. Counting
        i. The Pigeonhole Principle
        ii. Permutations and Combinations
        iii. Generation of combinatorial object
            1. Subsets

        2. Permutations
        3. Combinations
    iv. Recurrence Relations
    v. Inclusion-Exclusion
m. Relations
    i. Properties of Relations
    ii. n-ary Relations and Their Applications
    iii. Representing Relations
    iv. Closures of Relations
    v. Equivalence Relations
    vi. Partial Orderings

3. Basic Data Structures
    a. List Structures
        i. Linked lists
        ii. Stacks
        iii. Queues
    b. Graphs
        i. Graph Terminology and Properties
        ii. Representing Graphs and Graph Isomorphism
        iii. Connectivity
        iv. Euler and Hamilton Paths
        v. Planar Graphs
        vi. Graph Problems
            1. Topological Sort
            2. Connected Component
            3. Minimum Spanning Tree
            4. Shortest Path
            5. Maximum Flow
    c. Trees
        i. Binary search tree (BST)
        ii. Balanced BST
            1. AVL Trees
            2. Red-Black Trees
    d. Priority Queues
        i. Heap
        ii. Heapsort
        iii. Leftist Heap
        iv. Binomial Heap
        v. Fibonacci Heap
    e. Hashing
    f. Disjoint Set Union
4. The Divide and Conquer Method
    a. Binary Search
    b. Maximum Subsequence Problem
    c. Sorting
        i. Mergesort
        ii. Quicksort
        iii. Sorting in Linear Time
    d. Selection Problem and Quickselect
    e. Strassen's Matrix Multiplication
    f. FFT
    g. Convex Hull
5. The Greedy Method
    a. Fractional Kanpsack Problem
    b. Job Sequencing with Deadlines
    c. Minimum-Cost spanning tree
        i. Kuruskal's Alg.
        ii. Prim's Alg.
    d. Single-sourse shortest paths
        i. Dijkstra's alg.
    e. Huffman coding
6. Dynamic Programming
    a. Fibonacci numbers
    b. Shortest path
    c. 0/1 Knapsack

    d. Matrix Chain Product
    e. All-pairs shortest paths
    f. String Editting
    g. Optimal Binary Search Trees
    h. Flow Shop Scheduling
7. Graph Traversal Techniques
    a. Tree traversal and applications
    b. Depth-first search
    c. Bread-first search
    d. Connectivity algorithms
    e. Biconnectivity algorithms
8. Backtracking and Branch-and-Bound
    a. 8-queens problem
    b. Graph coloring
    c. Hamiltonian Cycles
9. Randomized Algorithms
    a. Monte Carlo Method
    b. Las Vegas Method
10. Lower bound theory
    a. Problem Reduction
    b. Decision Trees
        i. Sorting
        ii. Searching
        iii. Merging
    c. Adversary Arguments
        i. Largest and second largest
        ii. Merging
11. Introduction to the Theory of NP-completeness
    a. Modeling Computation
        i. Languages and Grammars
        ii. Finite-State Machines
        iii. Language Recognition
        iv. Turing Machines
    b. Complexity classes P and NP
        i. Decision Problems
        ii. Nondeterministic algorithms
    c. NP-completeness
        i. Problem reduction
        ii. NP-completeness proofs
        iii. NP-complete problems
            1. Boolean satisfiability problem
            2. k-clique problem
            3. Hamiltonian cycle problem
    d. NP-hard problems

Sample Questions

1. Let S be the set of all binary words of length **n** such that two zeros do not appear consecutively in any word in the set. Find a recurrence relation for the number of elements in S.

2. A full node is a node with two children. Prove that the number of full nodes plus one is equal to the number of leaves in a nonempty binary tree.

3. Let *T* be a minimum spanning tree of a graph *G = (V,E)*. Let *V′* be a subset of *V*, *T′* be the subgraph of *T* induced by *V′*, and *G′* be the subgraph of *G* induced by *V′*. Show that if *T′* is connected, then *T′* is a minimum spanning tree of *G′*.

4. If $l_1, l_2, ..., l_n$ denote the levels of the *n*+1 external nodes of an extended binary tree (each internal node has two children) with *n* internal nodes than show that

$$\sum_{i=1}^{n+1} 2^{-l_i} = 1 \qquad \text{where } n > 0$$

5. AVL tree is a balanced binary search tree. For each node of an AVL tree, the diffrerence between the height of the left-subtree and the height of the right-subtree is at most one. Prove that the height of an AVL tree $O(\log n)$. (Hint: Write a recurrence relation for the minimum number of the nodes in a heigth *h* AVL tree and solve it.)

6. Let P be a problem. The worst case time complexity of P is $O(n^3)$. The worst case time complexity of P is $\Omega(n \log n)$. Let A be an algorithm that solves P. Explain which of the following statements are consistent with this information :
   a. A has worst case time complexity $O(n^4)$.
   b. A has worst case time complexity $O(n \log n)$.
   c. A has worst case time complexity $O(n)$.
   d. A has worst case time complexity $\Theta(n^2)$.
   e. A has worst case time complexity $\Omega(n)$.

7. Solve: $nf(n)-(5n-5)f(n-1)=0$ where $f(1)=10$.

8. Find a recursion for the number of subsets of a given set with **n** elements and solve it.

9. Which function grows faster: $n^{(\ln n)}$ or $(\ln n)^n$?

10. Solve the recursion

$$f(n) = af(n/b) + c \quad ,$$

where $n=b^k$ and $f(1)=d$, by unfolding.

11. Let T(n) denote the number of different binary search trees on n different keys. Give a recurrence relation for T(n). You need not solve the recurrence relation

12. In this problem, we will consider an unusual measure of the performance of a sorting algorithm. Suppose that instead of returning its output in an array, we want an algorithm that emits the elements one at a time using a special O(1) time emit subroutine. For example, here is selection sort using emit:

```
SelectionSort(A):
    for i = n downto 1
            least := i
            for j = 1 to i-1
                    if A[j] < A[least]
                            least := j
            emit A[least]
            swap A[least], A[i]
```

Define the *delay* of the k-th output of a sorting algorithm as the time between its (k-1)-th and k-th calls to emit. So, for example, the delay for the k-th value of SelectionSort above is O(n-k) + O(1).

(a) Show that the worst-case delay for the first output is *Omega*(n) for any sorting algorithm.
(b) Give a sorting algorithm whose delay for all outputs except the first is O(1).

13. Here are two algorithms for computing the n-th Fibonacci number, defined by the recurrence F(n) = F(n-1) + F(n-2), with F(1) = F(2) = 1. Show that one of the algorithms is asymptotically more efficient than the other.

```
Fib1(n):
  if n < 2: return 1
  else: return Fib1(n-1) + Fib1(n-2)

Fib2(n):
  let A be an array of n elements A[1]...A[n]
  A[1] := 1
  A[2] := 2
  for i := 3 to n do:
        A[i] = A[i-1] + A[i-2]
  return A[n]
```

14. Prove or disprove the following statements:

a. $\frac{1}{\log_{10} n} = O(\frac{1}{n})$

b. $\log_{10} n + \sqrt{n} = O(n)$

c. $n! = \Omega(n^2 2^n)$

d. $T(1) = 1,\ \mathrm{T(n)}=3\mathrm{T}(\frac{n}{2}) + n + 1$, and $n = 2^m$ for some integer $m > 0$, then $T(n) = \Theta(n^{\log 3})$.

15. Let the running time of an algorithm *A* be described by the recurrence equation

$$T(n) = \sqrt{n}\ T\left(\sqrt{n}\right) + 3n.$$

A competing algorithm *B* has running time $S(n) = O(n^a)$. What is the smallest value for *a* such that *B* is asymptotically slower than *A*?

16. Let the running time of an algorithm A be described by the recurrence equation $T(n) = \sqrt{n}T(\sqrt{n}) + 3n$. A competing algorithm B has running time given by $S(n) = aS(\frac{n}{4}) + n$. What is the largest integer value for a such that B is asymptotically faster than A. Show all the details of your work.

17. Let the running time of an algorithm A be described by the recurrence equation $S(n) = 3S(\frac{n}{a}) + n$. A competing algorithm B has running time given by $S(n) = n^2$. What is the largest integer value for a such that B is asymptotically faster than A. Show all the details of your work.

18. Suppose you are given the following functions:

$\mathbf{T_1(N)= T_1(N/2) + 1,\quad T_1(1)=1}$ $\mathbf{T_2(N)= 2T_2(N/2) + N,\quad T_2(1)=1}$

$\mathbf{T_3(N)=\sqrt{N}}$ $\mathbf{T_4(N)=N^2}$

$\mathbf{T_5(N)=N\log(N)}$

State whether the following statements are true (T) or false (F).

(a) $\mathbf{T_1(N)= O(T_3(N))}$ (b) $\mathbf{T_1(N)= \theta(T_3(N))}$ (c) $\mathbf{T_1(N)= O(T_2(N))}$
(d) $\mathbf{T_2(N)= o(T_3(N))}$ (e) $\mathbf{T_3(N)= \Omega(T_1(N))}$ (f) $\mathbf{T_4(N)= o(T_4(N))}$
(g) $\mathbf{T_5(N)= O(T_2(N))}$ (h) $\mathbf{T_4(N)= \Omega(T_3(N))}$ (i) $\mathbf{T_1(N)= \theta(T_5(N))}$
(j) $\mathbf{T_5(N)= o(T_4(N))}$

19. Consider the following function **CanSum** which return true (1) if there are two integers in the array **a** that sum to exactly **k**.

```
int CanSum(int a[], int n, int k)
{
   int i,j,f=0;
   for (i=0; i<n; i++){
         j=i+1;
         for( ; j<n; ++j)
               if ((a[i]+a[j])==k)
                     f=1;
   }
   return f;
}
```

a. Analyze the running time of the function in Θ-notation.
b. Suppose it takes 1 second to run the function for an array of 1000 elements on the computer A. How long will it take to run the function for 5000 elements on the computer A?
c. Suppose computer B is 4 times faster than computer A. How large a problem can be solved on computer B in 16 seconds?
d. Design an asymptotically better algorithm for the same the same problem. Analyze the run-time of your algorithm.

20. Given a list L of real numbers and a tolerance $c>0$. Find as efficient an algorithm as you can to count the number of pairs of entries which differ in value by at most c. (The complexity of your algorithm should be better than $O(n^2)$.)

21. Given a sorted sequence of distinct integers $a_1, a_2, \ldots a_n$ and given an integer x. The problem is to find, if there exists, two indices i and j such that $a_i + a_j = x$. Give a O(n) time algorithm. (Partial credit will be given to a O(n log n) time algorithm.

22. The binomial coefficients can be defined by the recurrence equation:

$C(n,k) = C(n-1,k-1) + C(n-1,k)$ for $n>0$ and $k>0$

$C(n,0) = 1$ for $n \geq 0$

$C(0,k) = 0$ for $k>0$

C(n,k) is also called "*n* choose *k*" and denoted $\binom{n}{k}$. It is the number of ways to choose *k* distinct objects from a set of *n* objects. Describe and analyze following two ways to compute *C(n,k)*.

a. Divide and conquer algorithm
b. Dynamic programming algorithm

23. In an array of *n* distinct elements *A[1], ... ,A[n],* the rank of *A[i]=1* if exactly *l* of the *A[j]*'s are less than *A[i]*. In particular, three quartiles of an array of *n* elements, denoted by Q1,Q2, and Q3, are elements with ranks $\left\lceil \frac{n}{4} \right\rceil$, $\left\lceil \frac{n}{2} \right\rceil$ and $\left\lceil \frac{3n}{4} \right\rceil$, respectively. Give an *O(n)*-time algorithm to find the three quartiles of an array of size *n*; *n>3*. Explain why your (suggested) algorithm is of the specified order.

24. Given a sequence *X* of numbers, a subsequence of *X* is defined to be any contiguous portion of *X*. For example, if *X=5,7,1,3,6,3,7,1* then *7,1,3* and *3,6,3,7,1* are subsequences of *X*.
    a. Find the number of such subsequences.
    b. Give an $O(n^2)$ algorithm to find a subsequence where the summation of the numbers in the subsequence is exactly *k*.
    c. Give an O(n) algorithm to find a subsequence where the summation of the numbers in the subsequence is exactly *k*.

25. The partitioning problem is:
    Given *n* integers, is there a way to partition the integers into two disjoint subsets so that the sums of the integers in each of the two subsets are equal? Formally, given $x_1, \ldots, x_n$, is there a subset *I* of *{1,2, ... ,n}* such that

$$\sum_{i \in I} x_i = \sum_{i \notin I} x_i \ .$$

    Give a dynamic programming algorithm for this partitioning problem. Estimate the running time of your algorithm.

26. Consider the sorting problem of a very large number of keys that cannot be put into the active memory all at one time. Let *n* denote the number of keys to be sorted and *m* be the number that can be sorted in active memory. Let T_0 denote the tape which contains all data, all *n* of them, and T_1 , T_2 , T_3 are three additional tapes (a tape is an external memory device). The following algorithm describes a sorting procedure.

    Phase I. [Run construction and distribution]
    while there are more records on T_0 do
        1. Read *m* records (perhaps less the last time)
        2. Sort them using an internal sorting algorithm.
        3. If the previous run (a sorted set of *m*) was put on T_2, then put this on T_3; else on T_2.

    end
    Rewind the tapes.

Phase II. [Merging the runs]

i1 = 2 , i2 = 3 [indexes of input tapes]

j1 = 0, j2 = 1 [indexes of output tapes]

while there is more than one run do

1. Merge the first run on T_{i1} with the first run on T_{i2} and put the result on T_{j1}
2. Merge the next run on T_{i1} with the next run on T_{i2} and put the result on T_{j2}
3. Repeat steps 1 and 2 (putting the output alternately on T_{j1} and T_{j2}) until the end of the data on T_{i1} and T_{i2}.
4. Rewind the tapes. Add 2 (modulo 4) to i1, i2, j1, j2 and repeat steps 1, 2, 3 until done.

end

a. For n = 25 and m = 4, show how this algorithm will proceed to sort

20, 3, 25, 11, 17, 4, 6, 5, 18, 2, 21, 12, 19, 1, 7, 14, 15, 9, 8, 13, 10, 23, 22, 24, 16.

b. Analyze this algorithm, i.e., find the running time of this algorithm (in closed form) for n, m > 3. Make appropriate assumptions.

27. Show how to multiply two linear polynomials $ax+b$ and $cx+d$ using only tree multiplications. (Hint: One of the multiplications is $(a+b)*(c+d)$.) Using this fact, give a divide-and-conquer algorithm for multiplying two polinomials of degree-bound n. Analyze the run-time (number of multiplications) of your algorithm.

28. Given a set of n integers, is there a way to partition the set into two disjoint subsets so that the sums of the integers in each of the two subsets are equal?
    c. Give a greedy algorithm for this problem and discuss whether it gives the optimal solution or not.
    d. Give a dynamic programming algorithm for this problem.

    Analyze the run-time of your algorithms.

29. Let ω_m and ω_n be, respectively, primitive m-th and n-th roots of unity. The **two-dimensional discrete Fourier Transform** (2-D DFT) of a matrix $a[0..m-1,0..n-1]$ is a matrix $b[0..m-1,0..n-1]$ defined as

$$b_{k,l} = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} a_{i,j} p_k^i q_l^j ,$$

where, for $k = 0,1,...,m-1$, $p_k = \omega_m^k$, and, for $l = 0,1,...,n-1$, $q_l = \omega_m^l$.

a. How much time does it take to compute the array v directly from above equation.
b. Show that v can be computed in $O(mn \log (mn))$ time using the FFT. Assume both m and n are powers of 2. Recall that the FFT is a fast algorithm to calculate DFT. DFT of a vector $x[0..n-1]$ is the vector $y[0..n-1]$ where

$$y_k = \sum_{j=0}^{n-1} x_j \omega_n^{kj} .$$

30. The MAXMIN problem is to find the maximum and the minimum values in an array of n values. Design and analyze a divide and conquer algorithm for MAXMIN problem.

31. Assume you have a rod of length l. You would like to divide this rod into n+1 pieces by cutting the rod from the positions l_1, l_2, ..., l_n from one end of the rod. The cost of a cut is proportional to the length of the rod before the cut. Write an algorithm to find the sequence of cuts which minimizes the total cost of dividing the rod. Analyze the running time.

32. Consider the following problem:

    Given a sequence of n numbers, find whether the sequence is in sorted order or not.

    (a) Design an algorithm for this problem and analyze its running time.
    (b) Argue that the lower bound of this problem is $\Omega(n)$.
    (c) Assume that you can use only the operation of checking whether the sequence is ordered or not (i.e., you cannot compare the elements of the sequence.) Give an algorithm to order n numbers and analyze its running time.

33. What is the complexity class NP and NP-complete. You wish to prove that a problem, A, is NP-complete. What are the key steps to prove "A is NP-complete"?

34. The set-partition problem takes as input a set S of numbers. The question is whether the numbers can be partitioned into two sets A and B=S-A such that the sum of the numbers in both sets are equal.
    a. Give a greedy algorithm for this problem and discuss whether it gives the optimal solution or not.
    b. Show that the set-partition problem is in complexity class NP.
    c. Explain how we can prove that a problem is in complexity class NP-Complete.
    d. Consider the special case of the set-partition problem where the numbers are integers. Prove that this problem is in complexity class P by providing an efficient dynamic programming algorithm.