Analysis and Design of Algorithms

design an algo to solve a problem

analyze the algo

main questions about an algo

is it correct?

induction

contradiction
Analysis and Design of Algorithms

design an algo to solve a problem

analyze the algo

main questions about an algo

is it correct?

induction

contradiction
First - is there an answer correct others terminates gives the right answer

is it efficient? polynomial running time \( \mathbb{P} \) tractable
is there a better algorithm?

- e.g. sorting - no comparison based also that is \( O(n \log n) \)

   - efficient
What is the running time?

Computational code only works for very specific benchmarks. What is the running time?

Algorithm worst-case running time $O(n^2)$, nothing very dramatic.
running time based on the number of elementary operations

compare how the number of required operations increases with increasing input

$O$-notation

$T(n) = O(-f(n))$

if $T(n) \leq C \cdot f(n)$ for $n \geq n_0$
$c \cdot f(n) \sim 1.5n$

$T(n) \sim 0.5n$

$\eta_0$

$\eta = O(0.5n)$

Similar: $1.1^n \leq \cdot$
\[ N \geq \text{lower bound} \]

\[ T(n) \geq C f(n) \]

for sufficiently large inputs
\[ \theta \]

\[ \eta_0 \]

\[ \eta \]

\[ c_2 \phi(h) \]

\[ T(h) \]

\[ c_1 \phi(h) \]
Graph Algorithms

Graph Exploration

explored
unexplored
DFS Stuck
BFS queue
Dijkstra's Prim's queue
heapi
fibonacci heap
augmenting algorithms

- run an algorithm
- use its results to solve the problem

(may be with simple changes)
Greedy Algorithms

- Optimization problems
  - looking for the best solution (max, min, etc.)
- have a set of choices
- an edge to follow
- a vertex to add
- an interval to schedule
- pick whatever looks best

... gives best change in the value you are optimizing
benefits

simple

efficient

easy to understand/program

drawbacks

don't solve everything

still need

to

hard to prove they are correct

hard to find a good greedy choice
Divide and Conquer

3 steps
Sort

Divide
Recurse
Combine/Conquer

Base case

Assume sorted
proof of correctness: induction - base case

- termination - prove the input size decreases with each iteration

merge-sort(A, start, end)

merge-sort(left half), T(\frac{n}{2})

merge-sort(right half), T(\frac{n}{2})

merge

\frac{\Theta(n)}{2T(\frac{n}{2}) + \Theta(n)}
Running time in terms of a recurrence relation:

\[ T(n) = 2T\left( \frac{n}{2} \right) + \Theta(n) = \text{nlgn} \]

better than \( n^2 \), \( \checkmark \)

\[ 4T\left( \frac{n}{4} \right) + \Theta(n) = \text{nlgn} \]

need to solve to compare
Substitution – induction

recurrence trees – draw a tree

Master Theorem

\[ T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n) \]
Dynamic Programming

\[ T(n) = 2T(n-1) + O(1) \]
What if you could bound the number of subproblems by a polynomial, solve all the subproblems in some order, once each
Memoization -

store return values in a table based on the input variables

if the input occurs again return the stored value instead of executing the function body

\[ F(n) = F(n-1) + F(n-2) \]
What if your input is a data structure?
\[
\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}
\]

recurrence relation: divide and conquer - \( O(2^n) \)
1. Describe your problem as a table

\[ X[i,j] \text{ the first } i \text{ things} \]
\[ X[i,j,k] \text{ meaning things from } i-j \]
\[ X[i,j,k] \text{ meaning } s_i, \ldots, s_j, s_s, \ldots, s_j \]
2. Describe subproblems as a recurrence relation

\[
\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}
\]

- exponential step-down DEC also
3. Think of a better order to solve subproblems

\[ x(i,j) \] solve in increasing \( j \)

\[ x(i,j-1) \] solve by increasing \( j \)

solve by increasing \((j-i)\)

4. Recover the solution

- add a choice data structure
- backtracking
Data structures

- aug mentoring DS's

\[ \text{insert/delete/range search} \quad 0(\log n) \quad O(\log n^k + t) \]

priority search trees

d-d range trees
Randomization

Randomized algorithms can be faster in practice!

- Expected running times - randomize running time independent of input
- Average-case analysis - assume random input
- Analyze the expected running time
NP - completeness

P - decision problem that can be solved in polynomial time

NP - verified in polynomial time

NP-hard - problem that if it was in P then P = NP

most believe P \neq NP
Some things in NP-hard not in NP.