Algorithm & Complexity

Complexity Classes

$$1 \prec \log\log n \prec \log n \prec \sqrt{n} \prec 2^{\log n} \prec n \prec \log(n!)$$
$$= n\log n \prec n^2 \prec 2^n \prec 4^n \prec n! \prec (2n)! \prec 2^{n^2} \prec 2^{2^n}$$

Sort Algorithms

Algorithm	Best Case	Average Case	Worst Case	Space
Linear Search	$\Omega(1)$	$O(n)$	$O(n)$	$O(1)$
Binary Search	$\Omega(1)$	$O(\log n)$	$O(\log n)$	$O(1)$
Selection Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$O(1)$
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$O(1)$
Insertion Sort	$\Omega(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Merge Sort	$\Theta(n\log n)$	$\Theta(n\log n)$	$\Theta(n\log n)$	$O(n)$
Quick Sort	$O(n\log n)$	$O(n\log n)$	$O(n^2)$	$O(\log n)$

Master Theroem

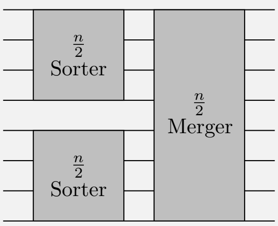
$$T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$$

$$T(n) = \begin{cases} O(n^d), & \text{if } d > \log_b a, \\ O(n^d \log n), & \text{if } d = \log_b a, \\ O(n^{\log_b a}), & \text{if } d < \log_b a. \end{cases}$$

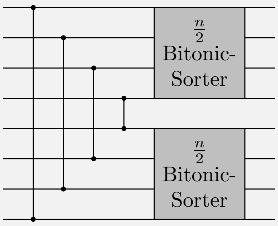
$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$$T(n) = \begin{cases} \Theta(n^{\log_b a}), & \exists \epsilon > 0 : f(n) = O(n^{\log_b a} - \epsilon) \\ \Theta(n^{\log_b a} \lg n), & f(n) = \Theta(n^{\log_b a}) \\ \Theta(f(n)), & \exists \epsilon > 0 : f(n) = \Omega(n^{\log_b a + \epsilon}) \wedge \\ & \exists c < 1 : af\left(\frac{n}{b}\right) \leq cf(n) \end{cases}$$

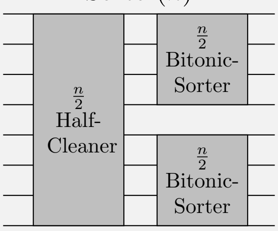
Sorting Network



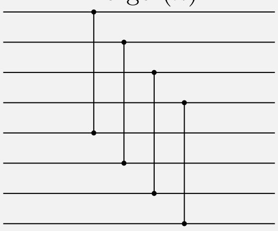
Sorter(n)



Merger(n)



Bitonic-Sorter(n)


Half-Cleaner(n)

Greedy Algorithm

Algorithm 1: Interval Scheduling

```
Sort jobs by finish times so that
  f_1 ≤ f_2 ≤ ··· ≤ f_n;
A ← ∅;
for j = 1 to n do
    if job j is compatible with A then
        A ← A ∪ {j};
return A;
```

Algorithm 2: Interval Partitioning

```
Sort intervals by starting time so that
  s_1 ≤ s_2 ≤ ··· ≤ s_n;
d ← 0;
for j = 1 to n do
    if lecture j is compatible with some
      classroom k then
        schedule lecture j in classroom k;
    else
        allocate a new classroom d + 1;
        schedule lecture j in classroom d + 1;
        d ← d + 1;
return A;
```

Algorithm 3: Minimize Lateness

```
Sort n jobs by deadline so that
  d_1 ≤ d_2 ≤ ··· ≤ d_n;
t ← 0;
for j = 1 to n do
    Assign job j to interval [t, t + t_j];
    s_j ← t, f_j ← t + t_j;
    t ← t + t_j;
return interval [s_j, f_j];
```

Matroid

Independent System $(S, \mathbf{C})$

$$A \subset B, B \in \mathbf{C} \Rightarrow A \in \mathbf{C}$$

Matriod $(S, \mathbf{C})$

$$\begin{cases} (S, \mathbf{C}) \text{ is an independent system,} \\ A, B \in \mathbf{C} \wedge |A| < |B| \Rightarrow \exists x \in B \backslash A : A \cup \{x\} \in \mathbf{C} \end{cases}$$

Maximal Independent subset I $\quad \nexists x \notin I : I \cup \{x\}$ is independent.

$u(F) = \min\{|I| | I \text{ is a maximal independent subset of } F\}$
$v(F) = \max\{|I| | I \text{ is an independent subset of } F\}$

Matroid Theorem an independent system $(S, \mathbf{C})$ is a matroid $\Leftrightarrow \forall F \subseteq S, u(F) = v(F)$.
Corollary All maximal independent subsets in a matriod have the same size.
Weighted Independent System $(S, \mathbf{C})$ with a nonnegative function $c : S \to \mathbb{R}^+$.

Greedy-MAX

An independent system $(S, \mathbf{C})$ with cost function c, solving a maximization problem as:

$$\begin{gathered}\max \; c(I) \\ \text{subject to } I \in \mathbf{C}\end{gathered}$$

Algorithm 4: Greedy-MAX

Sort all elements in S into ordering $c(x_1) \geq c(x_2) \geq \cdots \geq c(x_n)$;
$A \leftarrow \varnothing$;
for $i = 1$ *to* n **do**
    if $A \cup \{x_i\} \in \mathbf{C}$ **then**
        $A \leftarrow A \cup \{x\}$;
return A;

$$1 \leq \frac{c(A^*)}{c(A_G)} \leq \max_{F \subseteq S} \frac{v(F)}{u(F)}$$

Corollary If $(S, \mathbf{C}, c)$ is a weighted matroid, then Greedy-MAX algorithm performs the optimal solution.

Strongly Connected Components

Algorithm 5: Kosaraju Algorithm

Run DFS on G^R, and get the reversed visiting `order`;
Run DFS on G within the visiting `order`;

Dynamic Programming

Optimal Substructure
Overlapping subproblems

$$OPT(i, w) = \begin{cases} 0, & j = 0, \\ OPT(i-1, w), & w_i > w \\ \max\{OPT(i-1, w), \\ v_i + OPT(i-1, w - w_i)\}, & \text{else.} \end{cases}$$

Algorithm 6: Knapsack Algorithm

Input: $n, W, w_1, \cdots, w_n, v_1, \cdots, v_n$
Output: Optimal value of knapsack with W
for $w \leftarrow 0$ *to* W **do**
    $M[0, w] \leftarrow 0$;
for $i \leftarrow 1$ *to* n **do**
    for $w \leftarrow 1$ *to* W **do**
        if $w_i > w$ **then**
            $M[i, w] \leftarrow M[i-1, w]$;
        else
            $M[i, w] \leftarrow \max\{M[i-1, w], v_i + M[i-1, w - w_i]\}$;
return $M[n, W]$;

Running time is $\Theta(nW)$: "pseudo-polynomial".

Linear Programming

Primal Form

$$\max \sum_{j=1}^{n} c_j x_j$$

$$s.t. \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad \forall i$$

$$x_j \geq 0, \qquad \forall j$$

Dual Form

$$\max \sum_{i=1}^{n} b_i y_i$$

$$s.t. \sum_{i=1}^{m} a_{ij} y_i \leq v_j, \quad \forall j$$

$$y_i \geq 0, \qquad \forall i$$

Weak – Feasible $\sum_{j=1}^{n} c_j x_j \leq \sum_{i=1}^{m} b_i y_i$
Strong – Optimal $\sum_{j=1}^{n} c_j x_j = \sum_{i=1}^{m} b_i y_i$
Simple Method
Step 1. Converting LP into slack form.
Step 2. Setting all non-basic variables to 0.
Step 3. Selecting non-basic with tightest contraints.
Step 4. Exchange a nonbasic and a basic variable.
Step 5. Repeat from 2 to 4 until coefficients< 0.

Amortized Analysis

Aggregate Analysis sum up all the cost of ops to perform amortized analysis.
Accounting Method amortized cost is used to pay for later use. The blance never goes negative.

$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i, \quad \forall n$$

Potential Method with potential function $\Phi, \Phi(D_i) \geq \Phi(D_0)$

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0) \geq \sum_{i=1}^{n} c_i$$

Network Flow

Algorithm 7: Augment(f, c, P)

$\delta \Leftarrow$ bottleneck capacity of augmenting path P;
foreach $e \in P$ **do**
    if $e \in E$ **then**
        $f(e) \leftarrow f(e) + \delta$;
    else
        $f(e^R) \leftarrow f(e^R) - \delta$;
return f;

Algorithm 8: Ford-Fulkerson Algorithm

Input: $G = (V, E), c, s, t$
foreach $e \in E$ **do**
    $f(e) \leftarrow 0$;
$G_f \leftarrow$ residual graph;
while *there exists augmenting path* P **do**
    $f \leftarrow$ Augment(f, c, P);
    update G_f;
return f;

Also min-cut. $O(nmC)$.

Single Shortest Path

Algorithm 9: Dijkstra's Algorithm

$d[s] \leftarrow 0$;
foreach $v \in V - \{s\}$ **do**
  $d[v] \leftarrow \infty$;
$S \leftarrow \varnothing$;
$Q \leftarrow V$ `// priority queue, keyed on` d;
while $Q \neq \varnothing$ **do**
  $u \leftarrow$ Extract-min Q;
  $S \leftarrow S \cup \{u\}$;
  foreach $v \in Adj[u]$ **do**
    if $d[v] > d[u] + w(u,v)$ **then**
      $d[v] \leftarrow d[u] + w(u,v)$

Algorithm 10: Bellman-Ford's Algorithm

$d[s] \leftarrow 0$;
foreach $v \in V - \{s\}$ **do**
  $d[v] \leftarrow \infty$;
for $i \leftarrow 1$ *to* $\mid V \mid -1$ **do**
  foreach *edge* $(u,v) \in E$ **do**
    if $d[v] > d[u] + w(u,v)$ **then**
      $d[v] \leftarrow d[u] + w(u,v)$
foreach *edge* $(u,v) \in E$ **do**
  if $d[v] > d[u] + w(u,v)$ **then return** $\exists$ *a negative-weight cycle*;
  else $d[v] = \mu(s,v)$;

All Pairs Shortest Path

Algorithm 11: Floyd-Warshall's Algorithm

Input: Directed Graph $G = (V,E)$, $V \in \{1, 2, \cdots, n\}$,edge-weight function $w : E \rightarrow \mathcal{R}$
Output: $n \times n$ matrix of shortest-path weights $\mu(i,j)$ for all $i,j \in V$

$C \leftarrow A$;
for $k \leftarrow 1$ *to* n **do**
  for $i \leftarrow 1$ *to* n **do**
    for $j \leftarrow 1$ *to* n **do**
      if $d_{ij} > d_{ik} + d_{kj}$ **then**
        $d_{ij} \leftarrow d_{ik} + d_{kj}$

Algorithm 12: Johnson's Algorithm

1. Find function $h : V \rightarrow \mathcal{R}$ such that $w_h(u,v) \geq 0, \forall (u,v) \in E$ to solve the difference constraints or determine that a negative-weight cycle exists. $O(VE)$;
2. Run Dijkstra using w_h from each vertex $u \in V$ to compute $\mu_h(u,v), \forall v \in V$. $O(VE + V^2 \log V)$;
3. For each pair (u,v) of vertices to compute $\mu(u,v) = \mu_h(u,v) - h(u) + h(v)$. $O(v^2)$;

Minimum Spanning Tree

Algorithm 13: Kruskal's Algorithm

$A \leftarrow \varnothing$;
foreach $v \in V$ **do**
  Make-Set(v);
sort the edges of E into nondecreasing order by weight w;
foreach $(u,v) \in E$*, taken in nondecreasing order by weight* **do**
  if *Find-Set(u)$\neq$ Find-Set(v)* **then**
    $A \leftarrow A \cup (u,v)$;
    Union(u,v);
return A;

Algorithm 14: Prim's Algorithm

Input: Connected, undirected graph $G = (V,E)$ with weight function $w : E \rightarrow \mathcal{R}$
Output: A spanning tree T (connects the vertices) of minimum weight $w(T) = \sum_{(u,v)\in T} w(u,v)$

$Q \leftarrow V$;
$key[v] \leftarrow \infty, \forall v \in V$;
$key[s] \leftarrow 0$, for arbitrary $s \in V$;
while $Q \neq \varnothing$ **do**
  $u \leftarrow$ Extract-min(Q);
  foreach $v \in Adj[u]$ **do**
    if $v \in Q$ *and* $w(v) < key[v]$ **then**
      $key[v] \leftarrow w(u,v)$;
      $\Pi[v] \leftarrow u$;
return $\{v, \Pi[v]\}$ *forms MST*;

Single Shortest Path Algorithms

Circumstance	Algorithm	Time Complexity
Unweighted	BFS	$O(V+E)$
Non-negative Edge Weights	Dijkstra	$O(E+V\log V)$
General	Bellman-Fold	$O(VE)$
Directed Acyclic Graph (DAG)	Topological sort +1 and Bellman-Ford	$O(V+E)$

All Pair Shortest Path Algorithms

Circumstance	Algorithm	Time Complexity
Unweighted	$\lvert V \rvert$ ×BFS	$O(VE)$
Non-negative Edge Weights	$\lvert V \rvert$ ×Dijkstra	$O(VE + V^2 \log V)$
General (`baseline`)	$\lvert V \rvert$ ×Bellman-Fold	$O(V^2E)$
General	Floyd-Warshall	$O(V^3)$
General	Johnson	$O(VE + V^2 \log V)$

Turing Machine

Language System. If $f : \{0,1\}^* \to \{0,1\}^*$ is computable in time $T(n)$ by a TM M using alphabet set Γ, then it is computable in time $4 \log |\Gamma| T(n)$ by a TM $\tilde{M}$ using the alphabet $\{0, 1, \Box, \triangleright\}$.
Multi-Tape. If $f : \{0,1\}^* \to \{0,1\}^*$ is computable in time $T(n)$ by a TM M using k tapes, then it is computable in time $5kT(n)^2$ by a single-tape TM $\tilde{M}$.
Bidirectional Tape. If $f : \{0,1\}^* \to \{0,1\}^*$ is computable in time $T(n)$ by a bidirectional TM M, then it is computable in time $4T(n)$ by a TM $\tilde{M}$ with one-directional tape.
TM-Computable. M TM-Computes f if,

$$\forall a_1, \cdots, a_n, b \in \mathbb{N},$$
$$M(a_1, \cdots, a_n) \downarrow b \text{ iff } f(a_1, \cdots, a_n) = b$$

Other Domains. A function $f : D \to D$ extends a numeric function $f^* : \mathbb{N} \to \mathbb{N}$. We say that f is computable if f^* is computable.

$$f^* = \alpha \circ f \circ \alpha^{-1}$$

where $\alpha : D \to \mathbb{N}$ is an effective injection.

NP Reduction

$$P \subseteq NP \subseteq EXP$$

Reduction. Problem X polynomial reduces to problem Y if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps
- Polynomial number of calls to oracle that solves problem Y

$$X \leq_P Y$$