1 Recursion Complexity

Substitution	**Recursion-Tree**
Guess Asymptotic	Delete floors and ceils and suppose n of a good form.
Show Answer by Induction	Draw a tree, rooted with added term and recursive calls.

Theorem 1.0.1 (Master Theorem)**.** *If we have recurrence equation $T(n) = aT(n/b) + f(n)$ where $a \geq 1, b > 1$ are integers, $f(n)$ is asymptotically positive. Let $r = \log_b a$, we have :*

1. *If $f(n) = \mathcal{O}(n^{r-\varepsilon})$ for some $\varepsilon > 0$, then $T(n) = \Theta(n^r)$*
2. *If $f(n) = \Theta(n^r)$ then $T(n) = \Theta(n^r \log n)$*
3. *If $f(n) = \Omega(n^{r+\varepsilon})$ for some $\varepsilon > 0$, and $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n (regularity condition) then $T(n) = \Theta(f(n))$.*

Applications to FFT : recursively evaluate polynomials on roots of unity.

2 Hashing

Theorem 2.0.1 (Simple Uniform Hashing Assumption)**.** *Assuming SUHA : "h equally distributes the keys into the table slots", and assuming $h(x)$ can be computed in $\mathcal{O}(1)$, $E\left[T_{search}(n)\right] = \mathcal{O}\left(1 + \frac{n}{m}\right)$, and same for deletion time.*
Formally, SUHA is :

$$\forall y \in [1, |T|] \mathbb{P}\left(h(x) = y\right) = \frac{1}{|T|}$$

$$\forall y_1, y_2 \in [1, |T|]^2 \mathbb{P}\left(h(x_1) = y_1,\ h(x_2) = y_2\right) = \frac{1}{|t|^2}$$

$H = \{h : U \to [0, |T| - 1]\}$ is Universal if :

$$\forall k_1 \neq k_2 \in U,\ |\{h \in H \mid h(k_1) = h(k_2)\}| \leq \frac{|H|}{m}$$

Theorem 2.0.2. *If h is a hash function chosen uniformly at random from a universal family of hash functions. Suppose that $h(k)$ can be computed in constant time and there are at most n keys. Then the expected search time is $\mathcal{O}(1 + \frac{n}{|T|})$*

Theorem 2.0.3. *Let $p \in \mathcal{P}$ such that $U \subseteq [0, p-1]$. Then*

$$H = \left\{h_{a,b}(k) = ((ak + b) \mod p) \mod |T| \mid a \in \mathbb{Z}_p^* b \in \mathbb{Z}_p\right\}$$

is a universal family.

Theorem 2.0.4. *Cuckoo Hashing inserts in constant time, searches in constant time, only for static keys.*

3 Integer Sets

Proposition 3.0.1 (RBTrees)**.** *Red black tree use $\mathcal{O}(n)$ space, $\mathcal{O}(\log n)$ in time for insertion and deletion in the worst case.*

Proposition 3.0.2 (Treaps)**.**
- *Time for a successful search : $\mathcal{O}(\texttt{depth(v)})$ where $\texttt{key}(v) = k$.*
- *Time for an unsuccessful search : $\mathcal{O}(\max\left(\texttt{depth}(v^-), \texttt{depth}(v^+)\right))$ where $\texttt{key}(v^-)$ is the predecessor of the searched key.*

- *Insertion Time :* $\mathcal{O}(\max(\texttt{depth}(v^-), \texttt{depth}(v^+)))$ *where* $\texttt{key}(v^-)$ *is the predecessor of the searched key.*
- *Deletion Time :* $\mathcal{O}(\textit{tree depth})$
- *Split/Merge Time : same as insertion / deletion*

Definition 3.0.1. *A decision tree for a sorting algorithm is a binary tree that shows the possible executions of an algorithm on a set.*

Proposition 3.0.3 (Van Emde Boas Trees)**.** *They maintain successor queries in* $\mathcal{O}(\log\log u)$*, updates in* $\mathcal{O}(\log u)$ *in* $\Theta(u)$ *space.*

Proposition 3.0.4 (y-fast trees)**.** *Predecessor queries are in* $\mathcal{O}(\log\log u)$ *time, updates in* $\mathcal{O}(\log\log u)$ *expected amortised time, since insertion into the x-fast trie happens only once per* $\Theta(\log u)$ *new elements.*

4 String Algorithms

Proposition 4.0.1 (KMP)**.** *Suppose that we have computed* $B[1], \ldots, B[k-1]$*. We will now compute* $B[k]$*.*
By property 3, if $P[k] = P[B[k-1]+1]$*, then* $B[k] = B[k-1]+1$*.*
Else, if $P[k] \neq P[B[k-1]+1]$*, consider* $B^2[k-1] = B[B[k-1]]$*. If* $P[k] = P[B^2[k-1]+1]$*, set* $B[k] = B^2[k-1]+1$*, else consider* $B^3[k-1]$*, and so on.*

This algorithm is correct and in $\mathcal{O}(m)$*.*

Proposition 4.0.2 (Aho-Corasick)**.** *Aho-Corasick solves multiple pattern matching with space complexity* $\mathcal{O}(m)$ *and time complexity* $\mathcal{O}(m+n)$ *if no pattern is a substring of another else in* $\mathcal{O}(n+m+\#occ)$*.*

Proposition 4.0.3. *A suffix tree for a string of length* n *has* n *leaves, and at most* $2n-1$ *nodes and* $2n-2$ *edges. Storing the labels on the edges can take* $\Theta\left(|T|^2\right)$*. To save space we represent each label as two numbers, the left and the right endpoints in* T*.*

Theorem 4.0.1. *Pattern Matching queries are answered in* $\mathcal{O}(m+occ)$ *using suffix trees that can be build in linear time.*

5 Disjoint-Set

Theorem 5.0.1. *Using linked-lists and the weighted-union strategy, doing* m `make_set`*,* `union_set` *and* `find_set`*,* n *of which are* `union_set` *takes* $\mathcal{O}(m + n\log n)$

Theorem 5.0.2. *A sequence of* m `make_set, union_set, find_set`*, out of which* n *are* `make_set` *takes* $\mathcal{O}(m\alpha(n))$ *time. In other words, one operation takes* $\mathcal{O}(\alpha(n))$ *amortised time.*

6 Graphs

Theorem 6.0.1 (White-path Theorem)**.** *In a DFS forest of a digraph, a vertex* v *is a descendant of a vertex* u *if and only if at time* $u.d$*, there is a* (u,v)*-path made of undiscovered vertices.*

Proposition 6.0.1. G^{SCC} *is a DAG and is computed by Kosaraju's two pass algotihm in linear time.*

Proposition 6.0.2 (Cut and Paste Technic)**.** *Let* $G = (V, E)$ *and let* T *be a spanning tree of* G*. Let* $uv \in E(G) - T$ *and let* T_{uv} *be the unique path linking* u *and* v *in* T*. Then for every edge* xy *of* T_{uv} $T \setminus \{xy\} \cup \{uv\}$ *is a spanning tree of* T*.*

Theorem 6.0.2. *Kruskal returns the MST in $\mathcal{O}(m \log n)$. Prim returns the MST in $\mathcal{O}(m+n \log n)$ with Fibo Heap and $\mathcal{O}(m \log n)$ with a min-heap.*

Theorem 6.0.3 (Rado-Edmonds)**.** *The greedy algorithm for a problem is optimal for any weight function if and only if $(E, \mathcal{I})$ is a matroid.*

7 Parametrized Complexity

Definition 7.0.1. *A parametrized algorithmic problem is a problem where a certain parameter k is given in addition to the input. The complexity is studied as a function of n and k.*

Definition 7.0.2.

- *A parametrized problem is Fixed-Parameter Tractable if there is an algorithm deciding $\mathcal{P}$ in time $f(k)n^c$ where f is computable and c is constant.*
- *A parametrize problem $\mathcal{P}$ is XP if there is an algorithm deciding $\mathcal{P}$ in time $n^{f(k)}$ for some computable f and constanc c.*

Definition 7.0.3. *The main idea of the branching method is to reduce the problem to solving a bounded number of problems with parameter k' < k.*

Definition 7.0.4. *Let $\mathcal{P}$ be a parametrized problem and f a computable function.*
A kernel of size $f(k)$ is an algorithm that, given (x, k), runs in polynomial-time in $|x| + k$ and outputs an instance x', k' such that :

- $x, k \in \mathcal{P} \Leftrightarrow x', k' \in \mathcal{P}$
- $\left|x^{'}\right| \leq f(k)$ *and* $k' \leq k$.

We say the kernel is polynomial is f is polynomial.

Theorem 7.0.1. *A parametrized problem is FPT if and only if it is decidable and has a kernel.*

8 Approximation Algorithm

Definition 8.0.1. *An algorithm has an approximation ratio of $\rho(n)$ if for any input of size n the cost ALG of the solution produced by the algorithm is with a factor of $\rho(n)$ of the cost OPT of an optimal solution :*

$$1 \leq \max\left(\frac{OPT}{ALG}, \frac{ALG}{OPT}\right) \leq \rho(n)$$

9 Linear Programming

Definition 9.0.1. *A linear program is made of n decision variables $x_1, \ldots, x_n \in \mathbb{R}$, m linear constraints*

$$\sum_{j=1}^{n} a_{ij} \star b_i$$

where $\star \in \{\leq, \geq, =\}$; and a function we want to maximise.

The set of points $x \in \mathbb{R}^n$ at which a constraint holds with equality is a hyperplane.
Thus, each constraint is satisfied by a closed half-space of $\mathbb{R}^n$, and the set of feasible solution is the intersection of m closed half-spaces, that is, a convex polyhedron P.
A linear program can have no optimal solution :

- if the set of feasible solution is empty
- if for every integer M, there exists a feasible point x such that $c \cdot x \geq M$. In this case the set of feasible solution is unbounded.