CS3230

INTRO

Matching problem

- Input: preference rankings for all n men and n women
- Output: a stable matching, where no unmatched man and woman both prefer each other to their current partners

Gale-Shapley (1962)

```
Initialize each person to be free.
while (some man is free and hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged, and m' to be free
    else
        w rejects m
}
```

Properties

- `while` loop runs $\leq n^2$ times
- Men propose to women in decreasing order of preference
- A woman stays engaged after the first time she gets engaged. Her sequence of partners keeps getting better according to her preference list

Properties after terminate

- Each man is engaged to a unique woman
- The matching between men and women is stable
- Runs in $O(n^2)$
- For a man m, let *best*(m) be the highest ranked woman on his list who could be his partner in some stable matching. Gale-Shapley returns the matching where each man m is paired with *best*(m).
  - No two men have the same optimal partner
  - All men's wishes are simultaneoulsy optimized
  - Order of proposals does not matter
- In the matching returned by Gale-Shapley, each woman is paired with *worst*(w).

ASYMPTOTIC ANALYSIS

Word-RAM model

- Word is a collection of few bytes
- Word is the basic storage unit of RAM, which can be viewed as huge array of words
- Each input item is stored in binary format
- An arbitrary location of RAM can be accessed in the same time irrespective of the location
- Data and program fully reside in RAM
- Each arithmetic or logical operation involving a **constant number** of words takes **constant number of cycles (steps)** by the CPU

Big-O notation

Upper bound ($\geq$)
We write $f(n) = O(g(n))$ if for some $c > 0$ and $n_0 > 0$,
$$0 \leq \mathbf{f(n)} \leq cg(n)$$
for all $n \geq n_0$.

Lower bound ($\leq$)
We write $f(n) = \Omega(g(n))$ if for some $c > 0$ and $n_0 > 0$,
$$0 \leq cg(n) \leq \mathbf{f(n)}$$
for all $n \geq n_0$.

Tight bound We write $f(n) = \Theta(g(n))$ if for some positive constants c_1, c_2, n_0,
$$0 \leq c_1 g(n) \leq \mathbf{f(n)} \leq c_2 g(n)$$
for all $n \geq n_0$.

Strict upper bound ($>$)
We write $f(n) = o(g(n))$ if for some $c > 0$ and $n_0 > 0$,
$$0 \leq \mathbf{f(n)} < cg(n)$$
for all $n \geq n_0$.

Strict lower bound ($<$)
We write $f(n) = \omega(g(n))$ if for some $c > 0$ and $n_0 > 0$,
$$0 \leq cg(n) < \mathbf{f(n)}$$
for all $n \geq n_0$.

Properties

Set notations

- The notations are really just sets, e.g.
$$O(g(n)) = \{f(n) : \exists c > 0, n_0 > 0 \text{ such that } 0 \leq f(n) \leq c(g(n)) \quad \forall n \geq n_0\}$$
- $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$

Transitivity

$$f(n) = \Theta(g(n)) \wedge g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$$
$$f(n) = O(g(n)) \wedge g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$$
$$f(n) = \Omega(g(n)) \wedge g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$$
$$f(n) = o(g(n)) \wedge g(n) = o(h(n)) \Rightarrow f(n) = o(h(n))$$
$$f(n) = \omega(g(n)) \wedge g(n) = \omega(h(n)) \Rightarrow f(n) = \omega(h(n))$$

Reflexivity

$$f(n) = \Theta(f(n)) \quad f(n) = O(f(n)) \quad f(n) = \Omega(f(n))$$

Symmetry

$$f(n) = \Theta(g(n)) \iff g(n) = \Theta(f(n))$$

Complementarity

$$f(n) = O(g(n)) \iff g(n) = \Omega(f(n))$$
$$f(n) = o(g(n)) \iff g(n) = \omega(f(n))$$

Useful facts

- Degree-k polynomials are $O(n^k)$, $o(n^{k+1})$, and $\omega(n^{k-1})$
- Polynomials dominate logs: $(\log n)^{100} = o(n^{.0001})$
- Exponentials dominate polys: $n^{1000} = o(2^{.001n})$
- $\max(f(n), g(n)) = \Theta(f(n) + g(n))$

Exponentials

- For constants $k > 0, a > 1, n^k = o(a^n)$
- Exponentials of different bases differ by an **exponential factor**
- $2^{n+5} = O(2^n)$, but $2^{5n} \neq O(2^n)$

Properties

$$a^{-1} = \frac{1}{a} \qquad a^m a^n = a^{m+n}$$
$$(a^m)^n = a^{mn} \qquad e^x \geq 1 + x$$

Logarithms

- Binary log: $\lg n = \log_2 n$
- Exponentiation: $\lg^k n = (\lg n)^k$
- Natural log: $\ln n = \log_e n$
- Composition: $\lg \lg n = \lg(\lg n)$
- Base of log does not matter in asymptotics:
$$\lg n = \Theta(\ln n) = \Theta(\log_{10} n)$$

Properties

$$a = b^{\log_b a} \qquad \log_b \frac{1}{a} = -\log_b a$$
$$\log_c(ab) = \log_c a + \log_c b \qquad \log_b a = \frac{1}{\log_a b}$$
$$\log_b a^n = n \log_b a$$
$$\log_b a = \frac{\log_c a}{\log_c b} \qquad a^{\log_b c} = c^{\log_b a}$$

Overview

$$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < 2^n < 2^{2n} < n! < n^n$$

Stirling's approximation

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$
$$\log(n!) = \Theta(n \lg n)$$

Arithmetic series

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2} = \Theta(n^2)$$

Geometric series

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1}$$
$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \text{ when } |x|<1$$

Harmonic series

$$\sum_{k=1}^{\infty} \frac{1}{k} = \ln n + O(1)$$

Misc

$\lg(\lg n)! = \Theta(\lg n \lg\lg n)$ by subbing $\lg n$ into Stirling's approx.

$$\sum_{i=1}^{n-2} \lg\lg(n-i) = \Theta(n \lg\lg n)$$
$$\sum_{i=1}^{\lg n-1} \lg\lg \frac{n}{2^i} = \Theta(\lg n \lg\lg n)$$
$$n! > \frac{n}{2}^{\frac{n}{2}}$$

- For $T(n) = 2T(\sqrt{n}) + a$, the recursion tree has height $\lg\lg n$. Visualize by applying lg to each element of the recursion tree
- To compare two functions, consider taking the lg of each and compare that instead. This works since lg is strictly increasing

Limits

Assume $f(n), g(n) > 0$.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f(n) = o(g(n))$$
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \Rightarrow f(n) = O(g(n))$$
$$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \Rightarrow f(n) = \Theta(g(n))$$
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0 \Rightarrow f(n) = \Omega(g(n))$$
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \Rightarrow f(n) = \omega(g(n))$$

Epsilon-delta definition Let $f(x)$ be a function defined on an open interval around x_0, where $f(x_0)$ need not be defined. Then
$$\lim_{x\to x_0} f(x) = L$$
if for every ε there exists $\delta > 0$ such that for all x,
$$0 < |x - x_0| < \delta \implies |f(x) - L| < \varepsilon$$

L'Hopital If $\lim_{x\to\infty} f(x) = \lim_{x\to\infty} g(x) = 0$ or $\pm\infty$,
$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \lim_{x\to\infty} \frac{f'(x)}{g'(x)}$$

Power of e
$$\lim_{n\to\infty} \left(1 + \frac{a}{n}\right)^{bn+c} = e^{ab}$$

CORRECTNESS

Iterative algorithms

Loop invariant

- True at the beginning of an iteration
- Remains true at beginning of next iteration
- If still true at end, then it implies algorithm's correctness

Showing correctness

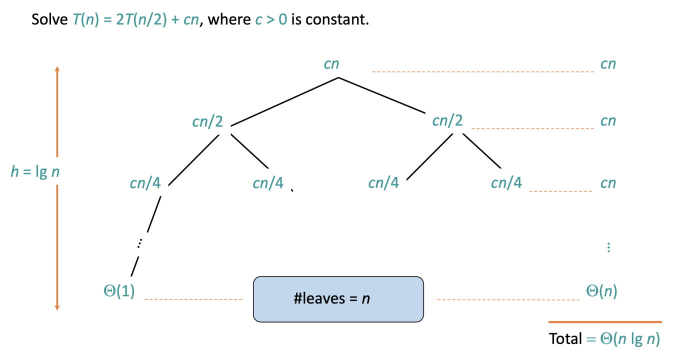
- Initialization: Invariant is true before first iteration
- Maintenance: If invariant is true before an iteration, it remains true before the next iteration
- Termination: When the algorithm terminates, the invariant provides a useful property for showing correctness

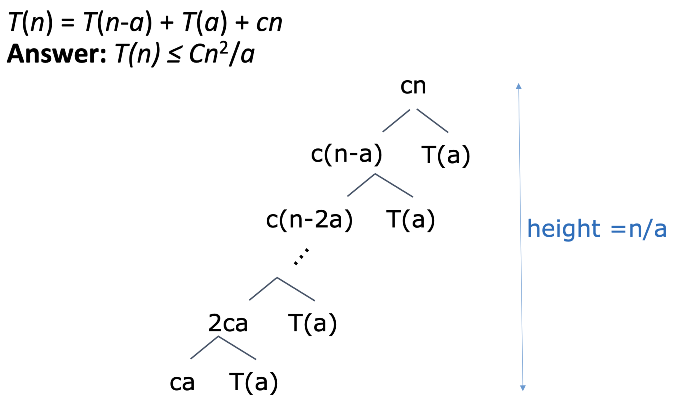
Recursive algorithms

Express time complexity in terms of recurrence:
$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Recursion tree

Draw tree of problem being recursively broken down, where the value at each node is the cost for that node. Sum up all costs to get the overall time complexity.


Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.


T(n) = T(n-a) + T(a) + cn
Answer: ***T(n) ≤ Cn²/a***

Master method

Comparing $f(n)$ with $n^{\log_b(a)}$ as polynomials, for $a \geq 1, b > 1, k \geq 0, \epsilon > 0$ and f is asymptotically positive,
$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$
$$= \begin{cases} \Theta(n^{\log_b(a)}) & \text{if } f(n) = O(n^{\log_b(a)-\epsilon}) \\ \Theta(n^{\log_b(a)} \log^{k+1} n) & \text{if } f(n) = \Theta(n^{\log_b(a)} \log^k n) \\ \Theta(f(n)) & \text{if } f(n) = \Omega(n^{\log_b(a)+\epsilon}) \end{cases}$$
For case 3, $f(n)$ must satisfy the regularity condition that $af(n/b) \leq cf(n)$ for some $c < 1$, which guarantees that the sum of subproblems is smaller than $f(n)$. But this is typically satisified anyway.

Substitution method

- Guess the form of the solution
- Verify by induction

Exponentiation

- Divide: Trivial
- Conquer: Recursively compute $f(\lfloor n/2 \rfloor, m)$
- Combine:
  - $f(n,m) = f(\lfloor n/2 \rfloor, m)^2 (\text{mod } m)$ if n is even;
  - $f(n,m) = f(1,m) \cdot f(\lfloor n/2 \rfloor, m)^2 (\text{mod } m)$ if n is odd
- $T(n) = T(n/2) + \Theta(1)$, so by master theorem, we have $T(n) = \Theta(\lg n)$

Induction

- Verify base case
- Show that if the statement is true for the kth case (regular induction) or all cases up to the kth case (strong induction) , then it is also true for the $(k+1)$th case
- Conclude that the statement is true for all cases

RANDOMIZED ALGOS

Probability

Axioms

Let S be a set (the full sample space). A function P is called probability if it assigns values to each subset of S, subject to:

- $\forall A \subset S, 0 \leq P(A) \leq 1$

- $P(S) = 1$
- If $A_1, A_2, \cdots$ are disjoint subsets of S, then $P(A_1 + A_2 + \cdots) = P(A_1) + P(A_2) + \cdots$

Results

- $P(\emptyset) = 0$
- For $A \subset S$, let A^C be the complement of A, then $P(A^C) = 1 - P(A)$
- Let $A \subset B \subset S$, then $P(A) \leq P(B)$

Conditional probability

Let A and B be events in a sample space S. If $P(A) \neq 0$, then the conditional probability of B given A is

$$P(B \mid A) = \frac{P(A \cap B)}{P(A)} = \frac{P(A \mid B)P(B)}{P(A)}$$

If we do not have $P(A)$, then we have to calculate it using disjoint events:

$$P(B_k \mid A) = \frac{P(A \mid B_k)P(B_k)}{P(A \mid B_1)P(B_1) + \cdots + P(A \mid B_n)P(B_n)}$$

Independence

Definition:

$$P(A \cap B) = P(A)P(B \mid A) = P(A)P(B)$$

We may use $A \perp\!\!\!\perp B$ to mean A and B are independent. When $A \perp\!\!\!\perp B$,

- $A^C \perp\!\!\!\perp B, A \perp\!\!\!\perp B^C, A^C \perp\!\!\!\perp B^C$
- If $P(A) > 0$, then $P(B \mid A) = P(B)$
- If $P(B) > 0$, then $P(A \mid B) = P(A)$

Joint (Mutual) independence 3 events A, B, C are jointly independent if

- A, B, C pairwise indepdent
- $P(A \cap B \cap C) = P(A)P(B)P(C)$

Expectation

Let X be a discrete random variable taking values in $\mathbb{Z}$. Let x_i denote the possible realizations of X. The expected value of X is

$$E(X) = \sum_{x_i \in \mathbb{Z}} x_i P(X = x_i)$$

Properties

- $E(a) = a$ for real constant a
- $E(aX + b) = aE(X) + b$ for real constants a, b
- If A and B are indepdent, $E(AB) = E(A)E(B)$

Linearity of expectation For random variables X and Y:

$$E(aX + bY) = aE(X) + bE(Y)$$

Indicator Random Variables

Let X be a indicator random variable associated with an event A, that occurs with probability p:

$$X = \begin{cases} 1 & \text{with probability } p \\ 0 & \text{with probability } 1-p \end{cases}$$

Then $E(X) = p$.

Bernoulli trials

- A Bernoulli trial is an experiment that has probability p of success, and probability $q = 1 - p$ of failure
- Let X be the number of Bernoulli trials before the first success
  $$\Pr[X = k] = (1-p)^{k-1} \cdot p$$
  and this is a geometric distribution, with $E[X] = \frac{1}{p}$

Quick sort

- Worst case: $O(n^2)$
- Average case, select 1st element as pivot: $O(n \log n)$
  - Depends on initial input permutation
- Outperforms merge sort, because
  - Less overhead of temporary storage
  - Fewer cache misses
- Worst case expected, with random pivot: $O(n \log n)$
- Randomized quick select: $O(n)$

Randomized analysis

- Select pivots uniformly at random
- Time taken depends on random choice of pivot, rather than randomness of input

Types of randomized algorithms

Randomized Las Vegas algorithms

- Output is always correct
- Running time is a random variable

Randmoized Monte Carlo algorithms

- Output may be incorrect with some small probability
- Running time is deterministic

HASHING

Dictionary

- Maintains (key, value) pairs
- Inserting a key will overwrite the value
- Querying a key will return the value

Variants

- Static: Inserted items are fixed, only care about queries
- Insertion-only: Only insertions and queries
- Dynamic: Insertions, deletions, queries

Implementation

- Static: sorted list - $O(\log n)$ queries
- Dynamic: balanced search tree - $O(\log n)$ operations

Hashing

Desired properties

- Minimize collisions
- Minimize storage space M, aim is $M = O(N)$
- Hash function should be easy to compute. Assume that hash can be computed in constant time

Universal hashing

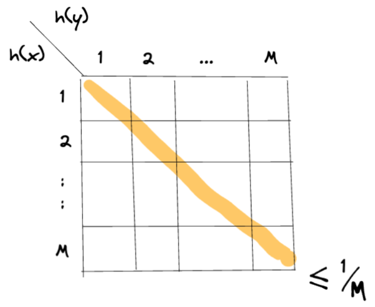
Suppose $\mathcal{H}$ is a set of hash functions mapping U to $[M]$. We say $\mathcal{H}$ is universal if for all $x \neq y$,

$$\frac{|h \in \mathcal{H} : h(x) = h(y)|}{|\mathcal{H}|} \leq \frac{1}{M}$$

or, using probability,

$$\Pr_{h \sim \mathcal{H}}[h(x) = h(y)] \leq \frac{1}{M}$$

- i.e. For any $x \neq y$, if h is chosen uniformly at random from universe $\mathcal{H}$, there's at most $\frac{1}{M}$ probability that $h(x) = h(y)$.
- Note: hash function is **fixed** throughout the lifetime of the dictionary, unless otherwise stated
- Consider the table below. For a hash function h drawn from a universal hash family, the above inequality states that the sum along the highlighted diagonal is $\leq \dfrac{1}{M}$.



Properties

- For any N elements $x_1, x_2, \cdots, x_N$, the expected number of collisions between x_N and the other elements is $\frac{N-1}{M}$ which is less than $\frac{N}{M}$.
  - i.e. the cost of an operation is $O(1)$
- For any sequence of N insertions, deletions and queries, if $M \geq N$, then the expected total cost for a random $h \in \mathcal{H}$ is $O(N)$

Construction

Suppose U is indexed by u-bit strings, and $M = 2^m$. For any binary matrix A with m rows and u columns,

$$h_A(x) = Ax \pmod 2$$

where $x \in U$ is a column vector. Then the hash family

$$\{h_A : A \in \{0,1\}^{m \times u}\}$$

is universal.

Additional storage

- Has overhead of $\Theta(\log N \cdot \log U)$ bits if $M = \Theta(N)$

Uniform hashing

$\mathcal{H}$ is said to be a uniform family of hash functions if for every key $x \in U$ and every hash value $i \in [M]$, it holds that

$$\Pr_{h \leftarrow \mathcal{H}}[h(x) = i] \leq \frac{1}{M}$$

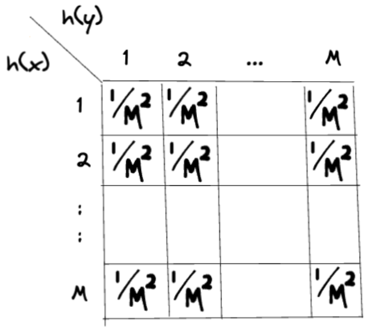
Pairwise independent

$\mathcal{H}$ is pairwise independent, if for any two distinct $x, y \in U$, and for any two hash values i_1, i_2

$$\Pr_{h \sim \mathcal{H}}[h(x) = i_1, h(y) = i_2] = \frac{1}{M^2}$$

- Pariwise independent $\Rightarrow$ Universal, but not the converse.

Consider the table below. For a hash function h drawn from a pairwise independent hash family, the above equality states that the probability in each cell is exactly $\dfrac{1}{M^2}$.



h(x) \ h(y)	1	2	...	M
1	$1/M^2$	$1/M^2$		$1/M^2$
2	$1/M^2$	$1/M^2$		$1/M^2$
:				
:				
M	$1/M^2$	$1/M^2$		$1/M^2$

(Static) perfect hashing

- Want to do all queries in worst-case constant time

Using quadratic space

If $\mathcal{H}$ is universal, and $M = N^2$, then if h is sampled uniformly from $\mathcal{H}$, the expected number of collisions is < 1.

- For $i \neq j$, let $A_{ij} = 1$ if $h(x_i) = h(x_j)$, 0 otherwise
- By universality, $E[A_{ij}] = \Pr[A_{ij} = 1] \leq \frac{1}{M} = \frac{1}{N^2}$
- $E[\text{collisions}] = \sum_{i \neq j} E[A_{ij}] \leq \binom{n}{2} \frac{1}{N^2} < 1$

2-level scheme

- Choose $h : U \mapsto [N]$ from a universal hash family
- Let L_k be the number of x_i for which $h(x_i) = k$
- Choose $h_1, \cdots, h_N$ second-level functions $h_k : [N] \mapsto [L_k^2]$ for each hash value k, such that there are no collisions among the L_k elements mapped to k by h
  - Such a hash function exists because of the quadratic space example earlier
- Space used is $O(N)$ (for first level) and $E[\sum_k L_k^2] = O(N)$ (for second level)

Proof

- For $1 \leq i, j \leq N$, define $A_{ij} = 1$ if $h(x_i) = h(x_j)$ and $A_{ij} = 0$ otherwise
- Note that $\sum_k L_k^2 = \sum_{i,j} A_{ij}$

$$E[\sum_{ij} A_i j] = \sum_i E[A_{ii}] + \sum_{i \neq j} E[A_{ij}]$$

$$\leq N \cdot 1 + N(N-1) \cdot \frac{1}{N}$$

$$< 2N$$

AMORTIZED ANALYSIS

Guarantees the average performance of each op in the worst case

Aggregate method

- Count total cost and divide by number of operations

Queues

- n INSERT and EMPTY operations
- Notice EMPTY is a sequence of DELETES, and DELETES $\leq$ INSERTS
- If there are k INSERTs, then sum of cost of all EMPTYs is $\leq k$
- Total cost $\leq k + k = 2k \leq 2n$ since $k \leq n$. Amortized cost is $O(1)$

Accounting method

- Charge ith operation a fictitious amoritzed cost $c(i)$, that satisfies
$$\sum_{i=1}^{n} t(i) \leq \sum_{i=1}^{n} c(i) \quad \forall n$$
where $t(i)$ is the true cost of the ith operation
- Usually $c(i) > t(i)$, with the extra amount paid stored as credit for future, rare, expensive operations
- Analysis should ensure that there's always enough credit to pay for ture cost
- Always **identify the expensive operation**, which you try to do "free of cost" using stored credit

Queues

- For INSERT, set amortized cost to 2 (true cost is 1)
- For EMPTY, set amortized cost to 0 (true cost is size of queue)
- Whenever an element is inserted, we pay an extra 1. This extra 1 can be used as credit to pay for later deletions
- Total cost is at most 2 $\times$ number of INSERTS $\leq 2n$

Binary increment

- Charge 2 for each $0 \to 1$; Charge 0 for each $1 \to 0$
- Starting from 0, actual cost for n increments is $O(n)$

Potential method

Motivation

- Accounting method tries to eyepower the required amortized cost
- Potential method tries to find some metric that decreases a lot on expensive operations
- Similar idea

Idea

- Define potential function ϕ, where $\phi(i)$ denotes potential at the end of the ith operation
- Must have $\phi(i) \geq 0$ for all i
- Amortized cost of ith operation, $c(i)$ is defined as
$$c(i) = t(i) + \phi(i) - \phi(i-1)$$
- Amortized cost of nth operations
$$\sum_i c(i) = \sum_i t(i) + (\phi(n) - \phi(0)) \geq \sum_i t(i) - \phi(0)$$
- Select suitable ϕ so that for the **costly** operation, $\Delta\phi_i$ is **negative to an extent** that it nullifies or reduces the effect of the actual cost

Binary increment

Working

- Use $\phi(i)$ = number of 1s in the counter after ith increment
- Let L_i be the length of the longest suffix with all 1s
- True cost of ith increment is $1 + L_i$
- $\Delta\phi_i = -L_i + 1$
- Sum of actual cost and $\Delta\phi_i$ is 2, so amortized cost of ith increment is 2

Results

- Starting from 0, actual cost for n increments is $O(n)$
- Starting from t ones, actual cost for n increments is $O(n + t)$

MISC

Inversion

Given an array of integers, the pair (i, j) is called an inversion if $i < j$ and $A[i] > A[j]$