

I. GEOMETRIC SERIES

$$\begin{aligned}\sum_{k=0}^{\infty} ar^k &= a \left(\frac{1 - r^{n+1}}{1 - r} \right) \\ &= \frac{a}{1 - r} \forall |r| < 1\end{aligned}$$

II. SMP

It is always true that employers with the same preferences will have the same stable matching.

Proof. Prove inductively, remove the most stable pair, then the remaining employers and employees have the same preferences. Continue until all pairs are removed. \square

III. ASYMPTOTIC

$\log < \text{polynomial} < \text{exponential} < \text{factorial}$

IV. GREEDY ALGORITHMS

Chase only the local optimum, not the global optimum.

A. Greedy stays ahead

If a greedy algorithm always makes a choice that stays ahead (at least as good) of the optimal solution, then the greedy algorithm is optimal.

Inductively show that at each stage, the greedy solution is at least as good as the optimal solution.

V. DIVIDE AND CONQUER

Divide the input into smaller instances (subproblems), solve recursively, then combine to get the solution.

A. Recurrence Relations

1) *Recurrence Tree:* Each level of the tree represents the cost of the work done at that level. The total cost is the sum of the costs at each level. Usually $\log n$ levels with a $f(n)$ cost at each level.

2) *Master Theorem:* Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the non-negative integers by the recurrence relation:

$$T(n) = aT(n/b) + f(n)$$

where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

- 1) If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
- 2) If $f(n) = \Theta(n^{\log_b a} \log^k n)$, then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$.
- 3) If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq kf(n)$ for some constant $k < 1$ and sufficiently large n , then $T(n) = \Theta(f(n))$.

B. Prune and Search

Definition. *Prune and Search* is a problem-solving strategy that divides the search space into two parts, one of which is pruned (discarded) and the other is searched.

To find the k th smallest element in an array, we can use the QuickSelect algorithm. The algorithm is as follows:

- 1) FunctionQuickSelect($A[1 : n], k$)
- 2) If $n = 1$ return $A[1]$
- 3) Choose a random pivot element p from A
- 4) Partition A into $L = \{x \in A : x < p\}$, $E = \{x \in A : x = p\}$, $G = \{x \in A : x > p\}$
- 5) If $k \leq |L|$, return QuickSelect(L, k)
- 6) Else if $k \leq |L| + |E|$, return p
- 7) Else return QuickSelect($G, k - |L| - |E|$)
- 8) End Function