Divide-and-Conquer

1. binary search
2. merge sort
3. powering a number
4. multiplying 2 number
5. matrix multiplication
6. embedding a complete binary tree

Example. how to compute F_n for a fibonnaci sequence

Solution. □

1. **Brute force** $O(n)$

$$F_1 = F_2 = 1$$
$$\text{For } i = 3 \ldots \text{n}$$
$$F_i = F_{i-1} + F_{i-2}$$

2. **Bottom up** $O(n)$

$$F[0] = 1 \tag{1}$$
$$F[1] = 1 \tag{2}$$
$$\text{for } i = 2 \cdots n \tag{3}$$
$$F[i] = F[i-1] + F[i-2] \tag{4}$$

   Idea is that result is stored in array. so dont have to compute any value again.

3. **math formula** $O(\lg n)$

$$F_n = \left\lfloor \frac{\phi^n}{\sqrt{5}} \right\rfloor \quad \phi = \frac{1+\sqrt{5}}{2}$$

   however ϕ is irrational. will cause underflow and overflow errors. Underflow happens when multiplying a number $n < 1$ many number of times, the values after significant digits are dropped in each iteration. At some point, it will be reduced to a very small number $0.00000 \cdots 0001$ whose next iteration will cause underflow error

4. **math formula II** $O(\lg n)$ Prove for every $n \geq 2$

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \left(\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \right)^n$$

powering a matrix same as taking power of number. so $O(\lg n)$

Proof. Proof by induction.
Basis:

$$LHS = \begin{bmatrix} F_3 & F_2 \\ F_2 & F_1 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = RHS$$

Inductive step: Given

$$\begin{bmatrix} F_{n_0+1} & F_{n_0} \\ F_{n_0} & F_{n_0-1} \end{bmatrix} = \left(\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \right)^n$$

Prove holds for $n_0 + 1$, i.e.

$$\begin{bmatrix} F_{n_0+2} & F_{n_0+1} \\ F_{n_0+1} & F_{n_0} \end{bmatrix} = \left(\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \right)^{n+1}$$

$$RHS = \left(\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \right)^n \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} F_{n_0+1} & F_{n_0} \\ F_{n_0} & F_{n_0-1} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} F_{n_0+1} + F_{n_0} & F_{n_0+1} \\ F_{n_0} + F_{n_0-1} & F_{n_0} \end{bmatrix} = LHS$$

No underflow error any more since the procedure includes only addition operation. The complexity is just integer multiplication which we can use divide and conquer to simplify □

Example. Every horse in the world is of the same color

Proof. By induction on the number of horses.
Basis: trivially true
Inductive step: Assume holds for n_0 horses, i.e. any set of n_0 horses has same color. □

Fallacy since cant prove for $n = 2$

Greedy algorithm

Example. Finding function global min with **newton-raphson algorithm** randomly selecting points, and find mins. Not really helpful in finding global min, so being greedy here doesnt really help

Example. Interval scheduling Given a set of requests $R = \{r_1, r_2, \cdots\}$, each one has a starting time. s_{r_i} and ending time t_{r_i}. Goal is to schedule max number of jobs possible. Assumes there is only one resource, hence no two job's interval may overlap, i.e. working on one resource concurrently.

Approach. Greedy approach. Define some simple rule to decide which request to pick next.

1. pick the job that has the smallest duration, i.e. $(t_{r_i} - s_{r_i})$. doesnt work, ex. two longer intervals that doesnt overlap vs a smaller interval that overlaps with both two longer intervals
2. take the job that starts the earliest. doesnt work, take example where there is one long interval vs. several smaller nonoverlapping intervals
3. pick the job that ends earlist. works...

To prove a rule is optimal. Use

1. **stay ahead**
2. **exchange argument**
3. **lowerbound**

□

Solution. □

sort the reqeusts R by end time. Let $A = \emptyset$ While R is not

sort request R by end time
$A = \emptyset$
while R not empty
    $i = R$ first item
    add i to A
    remove i from R and every job overlapping with i
return A

Proof. Let Θ be optimal solution. Want to show $|A| = |\Theta|$, i.e.

$$A = \{i_1, i_2, \cdots i_k\} \qquad |A| = k$$

$$\Theta = \{j_1, j_2, \cdots j_m\} \qquad |\Theta| = m$$

claim: For each pick $s = 1, \cdots, k$

$$t_{i_s} \leq t_{j_s}$$

Proof by induction on s. **basis** $s = 1$. since greedy algo picked the job ending first. so claim holds **inductive hypothesis** Assume result holds for s_0. show that claim holds for $s_0 + 1$. Hence, the next job that greedy algo picks is the one that ends soonest among the remaining non-overlapping requests. And j_{s_0+1} , i.e. the next request the optimal solution

picks, is one such requests. hence the claim holds.
Proof by contradiction Assume $k \neq m$. since Θ optimal i.e. $m \geq k$, we have $m > k$. Apply **claim** to $s = k$, we know $t_{i_k} \leq t_{j_k}$. now greedy algo terminates, but the optimal solution has j_{k+1} because $m > k$. But j_{k+1} is a perfectly valid nonoverlapping interval with

$$t_{i_k} \leq t_{j_k} < t_{j_{k+1}}$$

therefoer, $j_{k+1} \in R$ when i_k and its overlapping request are removed from R. $R \neq \emptyset$ so contradition... □

Example. Weighted interval scheduling One resource and each job has its own weight. goal is to select jobs with most weights

Example. Schedule all interals multiple resources. the goal is to schedule all intervals with as few resources as possible

Example. Huffman coding and data compression Given alphabet $S = \{a, b, c, d, e\}$, encode the text in minimum number of bits possible. As an example, a total of 32 letters requires using 5 bits encoding with fixed length encoding (i.e. every letter encoded by a sequence of 5 bits) vs variable length encoding (i.e. every letter are not encoded with fixed number of bits). Note that both fixed/variable encoding require same numbe of bits. So if text has 8 characters in total, the total number of bits required to encode the text is $8 \times 5 = 40$.
frequency of letters in text let $x \in S$. let f_x be fraction of times x occurs in the text. Let the size of the text be n. then x occurs nf_x times. Let C be the set of corresponding encodings. Let $\gamma : S \to C$. hence the code for x is denoted by $\gamma(x)$. The total length of encoding of the text

$$L = \sum_{x \in S} nf_x \cdot |\gamma(x)|$$

The average number of bits per letter is given by

$$ABL(\gamma) = \sum_{x \in S} f_x \cdot |\gamma(x)|$$

The goal is to come up with encoding γ such that $ABL(\gamma)$ is minimum possible. The idea is to encode higher frequency letters with smaller number of bits (with variable length encoding)

1. **How to get variable length encoding work (i.e. eliminate ambiguity)** Consider $a = 0, b = 1, c = 01$ the problem is that a is part of c so cant distinguish a with c. The solution to this problem is **prefix code** γ for S is an encoding such that for any two letter $x, y \in S$. $\gamma(x)$ is not a prefix of $\gamma(y)$. Now consider a full binary tree with leaves representing a letter. Prefix left subtree with 0 and prefix right subtree with 1. $(a = 00, b = 01, c = 10, d = 110, e = 111)$ The argument is that for any x to

be a prefix of y, x has to be an internal node on the path from root to y. Since no letter is an internal node, hence no two letter are prefix of each other. Also note that the conversion between prefix code and binary tree can be converted to each other.

2. **how to get the most optimal encoding with above approach**

   (a) one solution involves using divide-conquer algo. Let $S = \{x_1, x_2, \cdots, x_n\}$ with weights $W = \{w_1, w_2, \cdots, w_n\}$. Sort S by weights W and divide into approx. two parts, recursively make binary tree. and later combine the two binary trees. the solution is simple but not optimal.

   (b) Another solution involes greedy algo, which works! Let T^* is most optimal tree, suppose u and v are two leaves of T^*. leaf u is labelled with x and leaf v is labelled y with $depth(u) < depth(v)$, then

$$f_x \geq f_y$$

Definition. ***Greedy algo for Huffman encoding***

(a) If S has two letters then encode one with 0 and encode the other with 1

(b) Let y^* and z^* be two least frequent letters, form a new alphabet

$$S' = S \cup \{w\} \setminus \{y^*, z^*\} \qquad with \qquad f(w) = f(y^*) + f(z^*)$$

Recursively construct a prefix code γ' for S' with tree T'. Define a prefix code for S as follows. Starts with T', take the leaf labelled $w \in T'$, add two children below it, labelled x^* and z^*

Proposition. *Most optimal solution is a full binary tree*

Proof. If not full, then can replace parent with child and get a more optimal tree □

Proposition. *Let x, y be least frequent letter, there is some (not all) optimal tree where x and y are siblings*

Proof. Let T be an optimal tree for (S, f), where S is a set of alphabets and $f : S \to \mathbb{R}$ is the frequency of letters. Let x and y be two least frequent letters. If x and y are siblings, then done, otherwise x and y are not siblings, then they are on different levels. Without loss of generality, assume $|\gamma(x)| \geq |\gamma(y)|$, i.e. $f(x) \leq f(y)$. By previous proposition, x has sibling w. Now we derive tree T' by interchanging w and y. We claim that that T' is optimal. Let γ' be encoding of tree T'.

$$\begin{aligned} ABL(T') - ABL(T) &= f(w)|\gamma'(w)| + f(y)|\gamma'(y)| - f(w)|\gamma(w)| - f(y)|\gamma(y)| \\ &= f(w)|\gamma(y)| + f(y)|\gamma(w)| - f(w)|\gamma(w)| - f(y)|\gamma(y)| \\ &= (f(w) - f(y))(|\gamma(y)| - |\gamma(w)|) \end{aligned}$$

Since x and w on same level $|\gamma(w)| = |\gamma(x)|$ hence

$$ABL(T') - ABL(T) \leq 0$$

Since T is the optimal tree, implies

$$ABL(T') - ABL(T) \geq 0$$

Hence T' is also an optimal tree, claim holds. □

Proposition. ***Correctness for Huffman encoding with greedy algo***

Proof.

(a) **basis** $S = \{a, b\}$, assign $\gamma(a) = 0$ and $\gamma(b) = 1$

(b) **inductive step** Assume Huffman encoding gives optimal tree for any alphabet S of size n. Prove it works for alphabet size $n + 1$. Let T be the Huffman tree for S and assume O be an arbitrary optimal tree for S. By algo, replace x^*, y^* with w such that

$$S' = S \cup \{w\} \setminus \{y^*, z^*\} \qquad \text{with} \qquad f(w) = f(y^*) + f(z^*)$$

Since $|S'| = n$, by induction hypothesis, Huffman encoding gives the optimal tree T' for S'.

$$\begin{aligned}
ABL(T) &= \sum_{x \in S} f(x)|\gamma(x)| \\
&= \sum_{x \in S \setminus \{y^*, z^*\}} f(x)|\gamma(x)| + f(y^*)|\gamma(y^*)| + f(z^*)|\gamma(z^*)| \\
&= \sum_{x \in S \setminus \{y^*, z^*\}} f(x)|\gamma(x)| \ |\gamma(x)| + (f(y^*) + f(w^*))(|\gamma(w)| + 1) \\
&= \sum_{x \in S \setminus \{y^*, z^*\}} f(x)|\gamma(x)| + f(w)(|\gamma(w)| + 1) \\
&= \sum_{x \in S \cup \{w\} \setminus \{y^*, z^*\}} f(x)|\gamma(x)| + f(w) \\
&= ABL(T') + f(w) \\
&= ABL(T') + f(y^*) + f(z^*)
\end{aligned}$$

Assume for contradiction that T is not optimal, Since y^* and z^* are least frequent letters, hence by previous proposition, then y^* and z^* are siblings in O. If remove y^* and z^* from O replace them by w with $f(w) = f(y^*) + f(z^*)$ then O' must be optimal for S'. (This is true since assume O' is not optimal, exists O'' that

is optimal, $ABL(O'') + f(y^*) + f(z^*) < ABL(O') + f(y^*) + f(z^*) = ABL(O)$, which contradicts with assumption that O is optimal.) Then

$$\begin{aligned} ABL(O') &= ABL(T') \\ ABL(O') + f(y^*) + f(z^*) &= ABL(T') + f(y^*) + f(z^*) \\ ABL(O) &= ABL(T) \end{aligned}$$

Proved Huffman tree of size $n + 1$ T is optimal

□