

Question 1:

$$(i) \quad n^{100} < n^{\log(\log n)} < n^{\log n} < n^{\log^2 n} < 2^{\sqrt{n}} < 2^n$$

Grading scheme := Length - Longest correct subsequence
 X 0.5.

$$(ii) \quad T(n) = a \cdot T\left(\frac{n}{b}\right) + n^c$$

$$= a \cdot \left(a \cdot T\left(\frac{n}{b^2}\right) + n^c \right) + n^c$$

$$= a^k \cdot T\left(\frac{n}{b^k}\right) + \left(\sum_{i=0}^{k-1} a^i \right) n^c$$

$$= a^k \cdot T\left(\frac{n}{b^k}\right) + \left(\sum_{i=0}^{k-1} a^i \right) n^c \quad \text{let } k \approx \log_b n$$

$$= a^k \cdot O(1) + \frac{a^k - 1}{a-1} \cdot n^c$$

Grading scheme:

$$\text{Partial marking} = a^k \left[O(1) + \frac{n^c}{a-1} \right] - \frac{n^c}{a-1}$$

upto the evaluator.

$$= a^{\log_b n} \left[O(1) + \frac{n^c}{a-1} \right] - \frac{n^c}{a-1}$$

$$= n^{\log_b a} \left[O(1) + n^c/a-1 \right] - n^c/a-1$$

$$= O\left(n^{\log_b^a + c}\right)$$

Question 2:

- (a) 1, 2, 3, 8, 4, 5, 6, 7
 (b) 1, 2, 4, 5, 3, 6, 7

Grading scheme: Give 0.5 mark for every misplaced letter.

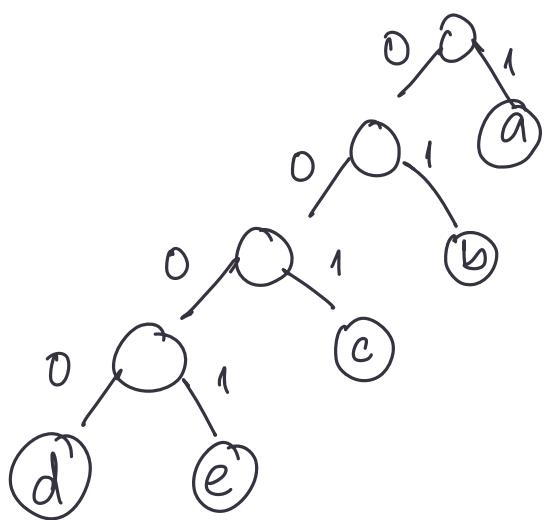
Question 3:

- Compute the DFS tree and place back edges
- In case, if a node without any back edge from its children to its ancestor or itself, it is an articulation point.

Grading scheme: 3 marks for identifying the application of DFS. 3 marks for identifying the back edges. Rest for clear arguments.

Question 4:

- (a) Grading scheme:
 Partial marking is subjective to the answer written.



$$\begin{aligned}
 (b) \quad \text{Average bit length} &= 0.5 \times 1 + 0.25 \times 2 + 0.125 \times 3 \\
 &\quad + 0.0625 \times 4 \times 2 \\
 &= 0.5 + 0.5 + 0.375 + 0.5 \\
 &= 1.875.
 \end{aligned}$$

Grading scheme
 1.5 for average
 bit length computation.

On an average, over a file of 1,000,000 characters,
 the encoding is at most 1,875,000 bits long.

Question 5: Assumptions: tree weights and acyclic.

Pick an ordering of the vertices. (topological sort).

Set $d(v) = \infty$ for all $v \in V \setminus \{s\}$. Put them in Q .
in incr. order.

While Q is not empty:
 $u \leftarrow \text{Extract Min}(Q)$ $\text{pred}(u)$: Previous node to
 u that leads to asc shortest path.

Let e_1, \dots, e_k be incoming edges into u .

Let u_1, \dots, u_k be the start nodes of e_1, \dots, e_k .

If $\text{wt}(\text{pred}(u_i), u_i) > \text{wt}(u_i, u)$ for all $i \in [1, k]$:
 $d(u) = \infty$

Else,

$$d(u) = \min \{ d(u_i) + \text{wt}(u_i, u) \mid \text{wt}(u_i, u) > \text{wt}(\text{pred}(u_i), u_i) \}$$

$\text{pred}(u) = u_{i^*}$ where u_{i^*} attains min above.

Return \bar{d} .

Correctness:

Using ascending instead
of monotonically increasing.

Claim : If there is ^{ascending} _{shortest} path from s to u ,
the algorithm will find it. size excluding source s .

Proof: Proof by induction on [↑] size of the _↑ graph
of G that contains all vertices on any $s \rightarrow u$ path
(path need not be ascending).

Base case: There is one more vertex than s .

The shortest path is trivial. One edge and it is
ascending by default.

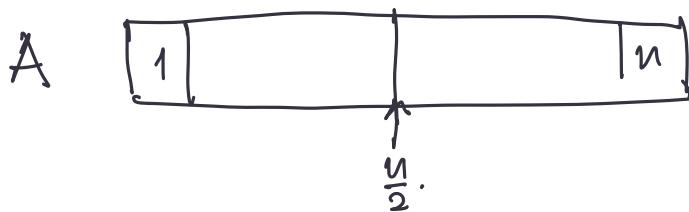
Induction step: Because we considered topological sort,
the vertices from which u has path from must have
been considered before u . Let u_1, \dots, u_k be those nodes.
From induction hypothesis; if there was an ascending
shortest path from $s \rightarrow u_i$ ($i \in [1, k]$), the algorithm
would have found it. Now, the only routes from s to u
must pass via u_1, \dots, u_k . Thus, over the sequence of
edges $(u_i \rightarrow u)$ for $i \in [1, k]$, $d(u)$ could get updated.

$$d(u) = \min \left\{ d(u_i) + \text{weight}(u_i, u) \mid \text{wt}(u_i, u) > \text{wt}(\text{pred}(u_i), u_i) \right\}$$

If there is an ascending path, it should be via one
of u_1, \dots, u_k . Thus, $d(u)$ gets correctly updated.

Running time: $O(m+n) + m$ extract min operations.

Question 6:



Ignoring the $\lceil \rceil$ and $\lfloor \rfloor$ for the brevity.

- Compare $A[\frac{n}{2}]$ and $\frac{n}{2}$.
if They are equal , return TRUE.
else if $A[\frac{n}{2}] > \frac{n}{2}$: recurse into $A[1, \frac{n}{2}-1]$.
else: recurse into $A[\frac{n}{2}+1, n]$.

Base case: Array with one element. Easy to check this case.

Correctness: Any argument of why to recurse in either direction may be considered.

4 marks for the algo. 2 marks for arguments for recursion and 2 for runtime analysis.

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + O(1) \\ &= O(\log n). \end{aligned}$$

Question 7: k-way merge

Form pairs of arrays (except one) ^{possibly}.

Run merge operation as in mergesort on each of these pairs.

Again form pairs of these and merge them pairwise. Continue this operation until all arrays merge into a k^n sized array.

$$\begin{aligned} \text{Runtime} &\leq \frac{k}{2} \cdot n + \frac{k}{4} \cdot 2n + \frac{k}{8} \cdot 4n + \dots + \frac{k}{2^{\log k}} \cdot (2^{\log k}) n \\ &\leq \frac{kn \log k}{2} \quad \begin{matrix} 4 \text{ for algo} \\ 3 \text{ for running time} \\ 1 \text{ for correctness.} \end{matrix} \\ &\approx O(n \cdot k \log k). \end{aligned}$$

Correctness can be an extended MergeSort argument.

Question 8:

Print (`Balanced Strings(n)`) where

`Balanced Strings (n):`

$S_n \leftarrow$ empty.

$S_n \leftarrow$ `Balanced Strings(n-1)`

for each string x in S_{n-1} :

$S_n \cdot \text{append } (L x R)$

$S_n \cdot \text{append } (LR x)$

Return S_n .

if cases are not covered, cut marks.

S_n is the collection of all balanced strings of length $2n$.

Mathematically,

$$\text{Balanced Strings}(n) = \{LR \cdot \text{Concatenation} \text{ } \text{Balanced Strings}(n-1)\} \cup$$

$\{ L \bullet \text{Balanced Strings}(n-1) \bullet R \}$

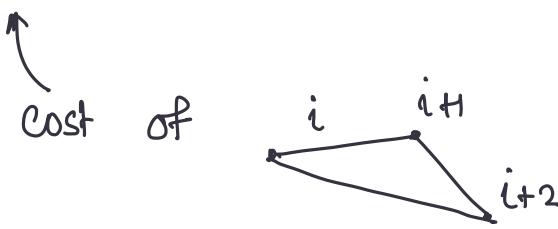
Balanced Strings(1) = $\{ LR \}$.

Question 9:

$A[i, j] = \min \text{ cost of triangulation of polygon spanned by } i, i+1, \dots, j.$

Base cases: $A[i, i+2]$ for all $i \in [n]$.

Partial marking
left to evaluator.



Perimeter of triangle $i \rightarrow k \downarrow j = \text{len}(i, k) + \text{len}(k, j) + l(j, i)$.

$A[i, i+2] = \text{Perimeter}(i, i+1, i+2)$.

$A[i, i+1] = 0$.



For $j > i+2$:

$$A[i, j] = \min_{i < k < j} \{ A[i, k] + A[k, j] + \text{Perimeter}(i, k, j) \}$$

Min cost triangulation is over all possible triangulations between i and k and k and j and the triangle i, j, k for all $i < k < j$.

$A[1, n]$ is the value we are interested in.

Memoization needs at most $n(n-1)$ many cells in the matrix. Each entry can be filled with at most n evaluations and min over that. $\leq O(n^3)$.

~~Better analysis :~~

Exact

$$\sum_{i=1}^{n-2} \sum_{j=i+2}^n (j-i-1) \leftarrow \text{Roughly } O(n^3) ?$$

Question 10: Refer to the notes.

Check for consistency.

Question 11:

If \exists an edge s.t it is not saturated, \exists a value that it could be decreased by. Thus, such edges are not critical.

Suppose an edge is saturated.

→ Say $u \rightarrow v$ is that edge.

Let us check if there is a $u \rightarrow v$ path, that avoids the edge (u, v) , such that this path can carry this flow from decrementing capacity on $u \rightarrow v$.
 in residual graph.

Algo: Residual graph of

1. $\text{res } G \leftarrow$ Final answer from Ford Fulkerson / Edmond Karp.
2. For each edge $u \rightarrow v \notin E(G)$ s.t $u \rightarrow v$ is in $\text{res } G$ and \nexists a u to v path in $\text{res } G$:

(u,v) is a critical edge.

Run Time: $\underbrace{O(V \cdot E^2)}_{\text{Max Flow algo in } 1} + \underbrace{O(V \cdot (V+E))}_{\text{all possible DFS}} \leq O(V \cdot E^2)$

As we consider
Connected graphs
 $E \geq V - 1$.

Question 12:

(1): $\text{MST} \in P$ and $P \subseteq NP \Rightarrow \text{MST} \in NP$.

(2) Input size $\leq \log A + \log k$.

Since $b \leq k$, representation size of b is at most $\log k$.

Assuming division is efficient in bit representation,
we can check if b divides A .

\Rightarrow Efficient verification.

How can you do it
efficiently?