Question 1:

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

Grading scheme := Length - longest correct subsequence $\times 0.5$.

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

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

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

Let $k = \log_b n$

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

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

Grading scheme:

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

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

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

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

Question 2:

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

(b) 1, 2, 4, 5, 3, 6, 7

Grading scheme: Cut 0.5 mark for every misplaced letter.

Question 3:

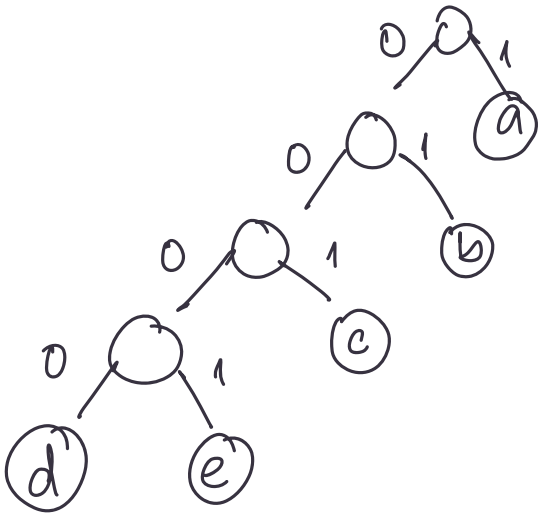
- Compute the DFS tree and place back edges
- In case, ∃ a node without any back edge from its children to its ancestor or it self, it is an articulation point.

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.



(b) Average bit length: $0.5 \times 1 + 0.25 \times 2 + 0.125 \times 3 + 0.0625 \times 4 \times 2$

$= 0.5 + 0.5 + 0.375 + 0.5$

$= 1.875.$

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: +ve 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:

pred(u): Previous node to u that leads to asc shortest path.

$u \leftarrow$ ExtractMin(Q)

Let $e_1, \ldots, e_k$ be incoming edges into u.

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

If $wt(pred(u_i), u_i) > wt(u_i, u)$ for all $i \in [1,k]$:

$d(u) = \infty$

Else,

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

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

Return $\bar{d}$.

Using ascending instead of monotonically increasing.

Correctness:

Claim: If there is an ascending shortest path from s to u, the algorithm will find it.

Proof: Proof by induction on size (excluding source s) of the subgraph of G that contains all vertices on any $s \rightsquigarrow u$ 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.

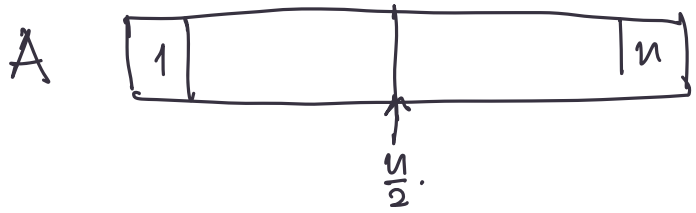
Induction step: Because we considered topological sort, the vertices from which u has path from must have been considered before u. Let $u_1, \ldots, u_k$ be those nodes. From induction hypothesis; if there was an ascending shortest path from $s \rightsquigarrow u_i$ ($i \in [1,k]$), the algorithm would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

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

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

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

Question 6:



Ignoring the ⌈ ⌉ and ⌊ ⌋ 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.

$$T(n) = T\left(\frac{n}{2}\right) + O(1)$$

$$= O(\log n).$$

Question 7: k-way merge

Form pairs of arrays (possibly except one).

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

Again form pairs of these and merge them pairwise.

Continue this operation until all arrays merge into a kn sized array.

Runtime: $\leq \frac{k}{2} \cdot n + \frac{k}{4} \cdot 2n + \frac{k}{8} \cdot 4n + \ldots + \frac{k}{2^{\log k}} \cdot (2^{\log k - 1})\, n$

$\leq \frac{kn}{2} \log k$

$\simeq O(n \cdot k \log k)$.

4 for algo
3 for running time
1 for correctness.

Correctness can be an extended MergeSort argument.

Question 8:

Print (Balanced Strings (n)) where

Balanced Strings (n):
    // if n=1 return {LR}
    $S_n \leftarrow$ empty.
    $S_{n-1} \leftarrow$ Balanced Strings (n-1)
    For each string x in S_{n-1}:
        S_n.append (L x R)
        S_n.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,

Balanced Strings (n) = {LR • Balanced Strings (n-1)} ∪

(• : concatenation)

$\{L \cdot \text{BalancedStrings}(n-1) \cdot R\}$

BalancedStrings(1) = $\{LR\}$.

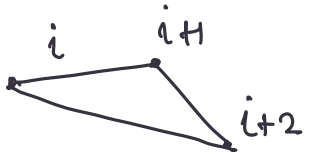
Question 9:

$A[i,j]$ = min cost of triangulation of polygon spanned by $i, i+1, \ldots, j$.

Partial marking left to evaluator.

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

↑ cost of



Perimeter of triangle $i \to k \to j \to i$ = $\text{len}(i,k) + \text{len}(k,j) + \ell(j,i)$.

$A[i,i+2]$ = 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 $\sum_{i=1}^{n-2} \sum_{j=i+2}^{n} (j-i-1)$ ← Roughly $O(n^3)$?

Question 10: Refer to the notes.

Check for consistency.

Question 11:

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

Suppose an edge is saturated.

↳ Say $u \to v$ is that edge.

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

Algo:

1. resG ← Final answer from Ford Fulkerson / Edmond Karp. Residual graph of
2. For each edge $u \to v \notin E(G)$ s.t $u \to v$ is in resG and ∄ a u to v path in resG:

(u,v) is a critical edge.

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

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

Question 12:

(1): MST $\in P$ and $P \subseteq NP \Rightarrow$ 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. ← How can you do it efficiently?

$\Rightarrow$ Efficient verification.