(CS1.301) Algorithm Analysis and Design (Monsoon 2023)

End of Semester Examination

Alloted time: 180 minutes **Total marks: 115**

Please read the instructions VERY carefully.

- **There are a total of 12 questions with varying credit, printed on pages 2 till 6.**
- **Discussions amongst the students are not allowed. No electronic devices (including smart watches) nor notes/books of any kind are allowed.**
- **Any dishonesty shall be penalized heavily.**
- **Any theorem/lemma/claim/fact that was proved in the class can be used without proof in the exam, only by explicitly writing its statement, and a clear remark that it was chosen from the class notes.**
- **Questions have been framed to be disambiguous, and queries may not be answered during the examination. In case you find any ambuguity, please mention that in your answer scripts and work with it. Answers got by misreading of questions may not be given credit.**
- **Be clear in your arguments. Partial marking is available for every question but vague arguments shall not be given any credit.**
- **You do not have to write a pseudocode unless explicitly asked.**
- **Analysis of running times, and proofs of correctness need to be done unless explicitly asked not to.**
- **Questions start on the next page.**
- **There is a hint for a question 9 at the end of page 6.**

Question 1 [2+4 marks]

(i) For a very large $n \geqslant n_0$, write the following asymptotic terms in an increasing order of Big-O notation.

(a) $n^{\log(\log n)}$

(b) $n^{(\log n)^2}$

(c) $n^{\log\log\log(n)}$

(d) n^{100}

(ii) For n that is large enough, and constants a, b, c, compute the asymptotic growth of the following function.

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + n^{1/c}.$$

That is, what is the value of N in terms of n, a, b, c such that $T(n)$ can be written as $O(N)$.

Question 2 [2+4+4 marks]

1. Compute third primitive root of unity.

2. Construct the DFT and inverse DFT matrices of order 3×3.

3. Compute the DFT of the vector $(1, 1, 1)$ (without simplifying the expressions).

Question 3 [8 marks]

A vertex u in a connected (undirected) graph $G = (V, E)$ is called an articulation point if removal of u from the vertex set, and removal of edges incident on the vertex u disconnects the graph. Give an algorithm to find articulation point(s) in a given graph.

Question 4 [3+4+4 marks]

A graph (V, E) is bipartite if the vertices V can be partitioned into two subsets L and R, such that every edge has one vertex in L and the other in R.

1. Prove that every tree is a bipartite graph.

2. Prove that a graph G is bipartite if and only if every cycle in G has an even number of edges.

3. Describe and analyze an efficient algorithm that determines whether a given undirected graph is bipartite.

Question 5

[8 marks]

Algorithm 1: OneDimensionalPeak(Array $A[1, n]$)

```
1  if length(A) ⩽ 3 then
2  |  return a peak here by brute-force.
3  end
4  middle ← ⌈length(A)/2⌉;
5  if A[middle − 1] ⩽ A[middle] and A[middle] ⩾ A[middle + 1] then
6  |  return middle;
7  else
8  |  if A[middle − 1] ⩾ A[middle] then
9  |  |  return OneDimensionalPeak(Array A[1, middle − 1]);
10 |  end
11 |  return OneDimensionalPeak(Array A[middle + 1, n]);
12 end
```

Given an array A with n elements, we would like to find an index i of a peak element $A[i]$ where $A[i] \geqslant A[i-1]$ and $A[i] \geqslant A[i+1]$. For elements on the boundaries of the array, the element only needs to be greater than or equal to its lone neighbor to be considered a peak.
Argue the correctness of Algorithm 1 for finding a peak, and compute its running time. In particular, argue that every given array will contain a peak. Here the notation $A[i, j]$ corresponds to the sublist of elements starting from index i until the index j.

Question 6

[12 marks]

Suppose we are given a set U of objects labeled $p_1, p_2, \ldots, p_n$. For each pair p_i and p_j, we have a numerical distance $d(p_i, p_j)$. We futher have the property that for all $1 \leqslant i \leqslant n$, $d(p_i, p_i) = 0$ and for all $1 \leqslant i \neq j \leqslant n$, $d(p_j, p_i) = d(p_i, p_j) > 0$.
For a given parameter k as input, k-clustering of U is a partition of U into k nonempty sets $C_1, C_2, \ldots, C_k$. Spacing of a k-clustering is defined to be the minimum distance between any pair of points lying in different clusters. That is,

$$\text{Spacing}(C_1, C_2, \ldots, C_k) = \min_{1 \leqslant u \neq v \leqslant k} \{\min\{d(p, p') \mid p \in C_u \text{ and } p' \in C_v\}\}.$$

Given that we want points in different clusters to be far apart from one another, a natural goal is to seek the k-clustering with the maximum possible spacing. In other words, we want to find the partition of U into k non-empty sets that maximizes the following expression.

$$\max_{U = C_1 \sqcup C_2 \sqcup \ldots \sqcup C_k} \{\text{Spacing}(C_1, C_2, \ldots, C_k)\}$$

The question now becomes the following – how can we efficiently find the one that has maximum spacing?

Question 7 [8 marks]

Consider an undirected (positively) weighted graph $G = (V, E)$ with a MST T and a shortest path $\pi(s, t)$ between two vertices $s, t \in V$. Will T still be an MST and $\pi(s, t)$ be a shortest path if

1. Weight of each edge is multiplied by a fixed constant $c > 0$.
2. Weight of each edge is incremented by a fixed constant $c > 0$.

Question 8 [8 marks]

Given a sequence $a_1, \ldots, a_n$, we say that two indices $i < j$ form an inversion if $a_i > a_j$, that is, if the two elements a_i and a_j are "out of order". We seek to determine the number of inversions in the sequence $a_1, \ldots, a_n$.

The basic idea is to divide the list into the two pieces $a_1, \ldots, a_m$ and $a_{m+1}, \ldots, a_n$. We first count the number of inversions in each of these two halves separately. Then we count the number of inversions (a_i, a_j), where the two numbers belong to different halves. We would like to do this merging part in $O(n)$ time.

Please read through the following pseudo-code (with explanation for each step) and fill in the missing snippets (in Algorithm 3) so that merging can be done in $O(n)$ time. Also construct the recursive relation for time complexity and state the correct running complexity. Argue why your answer is correct.

Algorithm 2: Sort-and-Count(L)

```
1 if the list L has one element then
2     return "There are no inversions";
3 A ← first ⌈n/2⌉ elements of L;
4 B ← the remaining ⌊n/2⌋ elements of L;
5 (r_A, A) = Sort-and-Count(A);
6 (r_B, B) = Sort-and-Count(B);
7 (r, L) = Merge-and-Count(A, B);
8 return r = r_A + r_B + r, and the sorted list L;
```

Algorithm 3: Merge-and-Count(A,B)

```
1 Maintain a Current pointer into each list, initialized to point to the front elements;
2 Maintain a variable Count for the number of inversions, initialized to 0;
3 while both the lists A and B are nonempty do
4     TODO: FILL THE MISSING CODE
5 return Count and the merged list;
```

Question 9 [12 marks]

A polygon is convex if every line that does not contain any edge of the polygon intersects the polygon in at most two points.

You are given a convex polygon P on n fixed vertices in the plane (specified by their x and y coordinates). A triangulation of P is a collection of $n - 3$ diagonals of P such that no two

diagonals intersect (except possibly at their endpoints). Notice that a triangulation splits the polygon's interior into $n-2$ disjoint triangles (see Figure 1 for a visualization).
The cost of a triangulation is the sum of the perimeters of the triangles in it. Give an efficient algorithm for finding a triangulation of minimum cost.

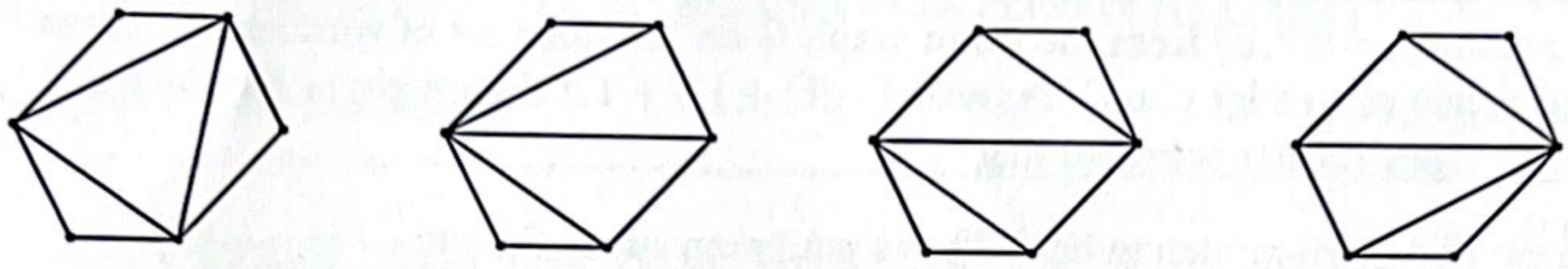

Figure 1: Some of the many different ways to triangulate a convex polygon on 6 vertices. Note that the second and the last figures are distinct.

Question 10 [12 marks]

For the following network, with edge capacities as shown in Figure 2, find the maximum flow from s to t along with the associated minimum cut.

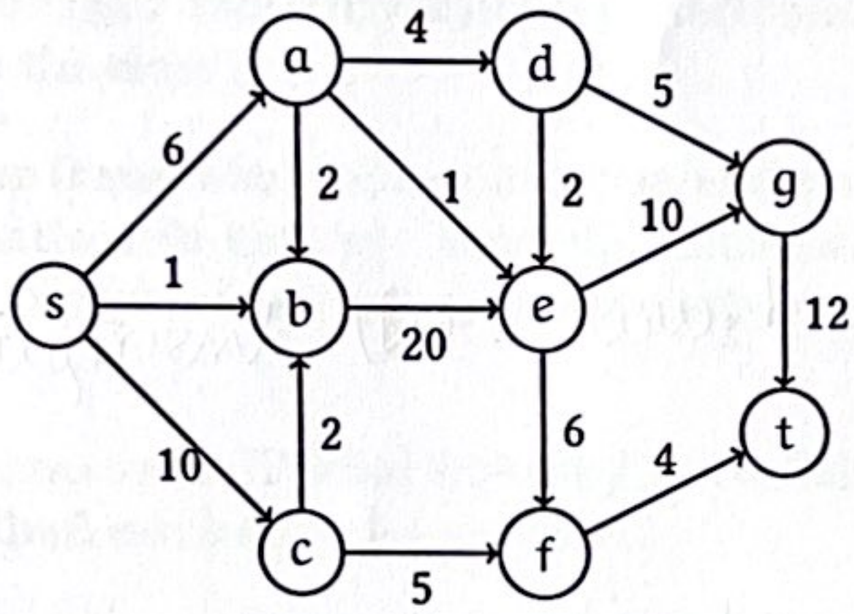

Figure 2: Figure for question 10

Please work this problem out by first starting with a feasible flow and then listing out all the intermediate steps of computing the residual graphs and augmentations until the maximum flow is found.

Question 11 [8 marks]

Suppose you are given a directed graph $G = (V, E)$, with a positive integer capacity c_e on each edge e, a source $s \in V$, and a sink $t \in V$. You are also given a maximum s-t flow in G, defined by a flow value f_e on each edge e. The flow f is acyclic: There is no cycle in G on which all edges carry positive flow. The flow f is also integer-valued.
Now suppose we pick a specific edge $e' \in E$ and increase its capacity by 1 unit. Show how to find a maximum flow in the resulting capacitated graph in time $O(m + n)$, where m is the number of edges in G and n is the number of nodes.

Question 12 [12 marks]

A given flow network $G = (V, E)$ may have more than one minimum (s, t)-cut. Let us define the best minimum (s, t)-cut to be any minimum cut (S, T) with the smallest number of edges crossing from S to T. You are asked to design an algorithm to find the best minimum (s, t)-cut when the capacities are integers. Towards that a colleague tells you that it is suffient to construct a new graph $G' = (V, E)$ from the given graph G on the same set of vertices and edges exept that the weight of an edge e in G' is $(wt(e) \cdot (|E| + 1)) + 1$ if the weight of the same edge in G was $wt(e)$. They further point out that

- a cut that is not minimum in G is not a minimum cut in G', and
- the minimum cut in G' gives the best minimum cut for G.

Your task is to analyse if your colleague was correct. Please provide detailed mathematical arguments.

Hints

- Hint for question 9: Label the vertices of P by $\{1, 2, \ldots, n\}$, starting from an arbitrary vertex and walking clockwise. For $1 \leqslant i < j \leqslant n$, let the subproblem $A(i, j)$ denote the minimum cost triangulation of the polygon spanned by vertices $i, i + 1, \ldots, j$.