COMP 3711 Design and Analysis of Algorithms
Fall 2015 Final Exam

1. **Time Complexity** (16 pts)

   We have two algorithms, A and B. Let $T_A(n)$ and $T_B(n)$ denote the time complexities of algorithm A and B respectively, with respect to the input size n. Below are 8 different cases of time complexities for each algorithm. Complete the last column of the following table with "A", "B", or "U", where:

   - "A" means that algorithm A is faster;
   - "B" means that algorithm B is faster;
   - "U" means that we do not know which algorithm is faster.

Case	$T_A(n)$	$T_B(n)$	Faster
1	$\Theta(n^{2.1})$	$\Theta(n^2 \log n)$	
2	$\Theta(n^2)$	$\Theta(2^{\sqrt{n}/\log n})$	
3	$O(n \log n)$	$O(n^{1.1})$	
4	$\Omega(\log^3 n)$	$O(\log n^6)$	
5	$\Omega(n^2)$	$\Theta(n^{2.31})$	
6	$\Theta(n^2)$	$\Omega(n^{2.5})$	
7	$\Omega(2^n/n^3)$	$O(n^3 2^{\log n})$	
8	$\Theta(\sqrt[3]{\log n})$	$\Theta(\sqrt{\log n}/\log\log n)$	

2. **Shortest path algorithms** (10 pts)

   Select the most efficient algorithms from below to fill in each blank.

   (a) Dijkstra's algorithm;
   (b) Bellman-Ford algorithm;
   (c) Floyd-Warshall algorithm;
   (d) The problem is not well defined.

   (1) Use ______ to find the shortest path from a vertex u to another vertex v in a directed graph with positive edge weights.
   (2) Use ______ to find the shortest path from a vertex u to another vertex v in a directed graph with both positive and negative edge weights, but with no cycles of negative weight.
   (3) Use ______ to find the shortest path from a vertex u to another vertex v in a directed graph with both positive and negative edge weights, and there may be cycles of negative weight.

(4) Use ______ to find the shortest path for all pairs of vertices an directed graph with both positive and negative edge weights, but with no cycles of negative weight.

(5) Use ______ to find the shortest path from a vertex u to another vertex v in an undirected graph with positive edge weights, but with one edge having negative weight.

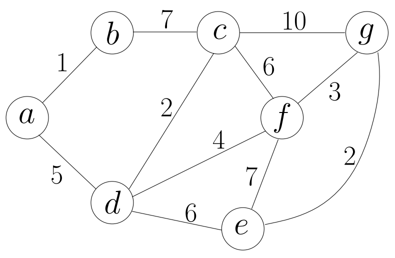
3. **Huffman coding** (6 pts)

Find an optimal Huffman code for the following alphabet with the given frequencies. Note that the optimal Huffman code is not unique; any one of them is acceptable.

a	b	c	d	e	f
5	8	3	9	1	2

4. **Prim's algorithm** (5 pts)

Run Prim's algorithm on the following graph, and show the partial minimum spanning tree after every new edge has been added. The starting vertex is "a".



5. **Minimum subsequence** (15 pts)

Given an array $A[1..n]$ of positive integers, design an $O(n)$-time algorithm to find a subsequence of A with the minimum sum such that, for every three consecutive elements, at least one of them must be selected in the subsequence. An example is give below, in which the optimal subsequence are boxed. Your algorithm should return both the optimal sum and a subsequence achieving the optimal sum.

$$3 \quad \boxed{1} \quad 2 \quad 4 \quad \boxed{0} \quad \boxed{4} \quad 5 \quad 5$$

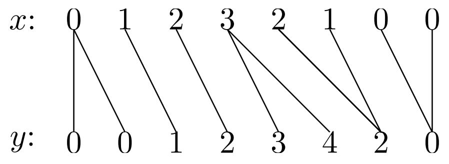
6. **Dynamic time warping** (12 pts)

The *dynamic time warping (DTW)* distance is a measure for the similarity between two temporal sequences that may vary in speed. Suppose we are given two arrays $x[1..n]$ and $y[1..n]$ of integers. To compute the DTW between x and y, we find a matching between x and y such that

(a) (no missing) every $x[i]$ is matched to at least one $y[j]$, and every $y[j]$ is matched to at least one $x[i]$;

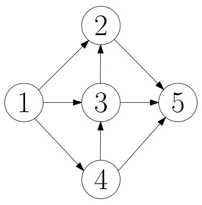
(b) (no crossing) if for $i_1 \le i_2$, $x[i_1]$ matches with $y[j_1]$ and $x[i_2]$ matches with $y[j_2]$, we must have $j_1 \le j_2$.

For a given matching, we compute the sum of $|x[i] - y[j]|$ for all matched pairs $(x[i], y[j])$. Then the DTW is the smallest sum among all valid matchings. The figure below shows an optimal matching between two given sequences, which gives a DTW of 2. Your job is to design an $O(n^2)$-time algorithm to compute the DTW (the matching itself is not needed) between two given sequences.

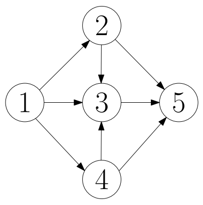


7. **Path finding** (12 pts)

   Recall that a Hamiltonian path in a graph is a path that visits each vertex exactly once. We mentioned in class that this problem in NP-hard for general graphs. However, it turns out this problem can be solved efficiently in a directed, acyclic graph. The figures below show two examples on which such a path exists or not, respectively. Your job is to design an algorithm to find such a path, or report that it doesn't exist. For full credits, your algorithm should run in $O(V + E)$ time.



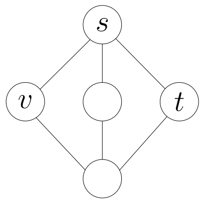
Hamiltonian path: 1, 4, 3, 2, 5



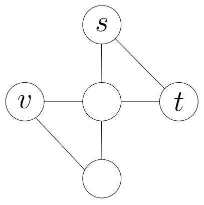
No Hamiltonian path

8. **Path finding, again** (12 pts)

   Given an undirected graph $G = (V, E)$ and three vertices s, t, and v in the graph, design and analyze (the running time of) an algorithm to determine whether there is a simple path from s to t that passes through v. Note that a simple path cannot visit the same vertex twice. The figures below show two examples. For full credits, your algorithm should run in $O(V + E)$ time. Your algorithm only needs to return "yes" or "no"; there is no need to find the actual path. [Hint: Use maximum flow.]



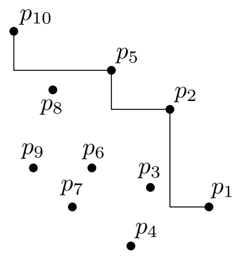
Yes



No

9. **Random skyline** (12 pts)

   Let P be a set of n points in the plane. A point p in P is said to be on the *skyline* if no other point in P is both above and to the right of p. For example, there are 4 points (p_1, p_2, p_5, p_{10}) on the skyline in the figure below. Suppose each point in P is chosen independently and uniformly at random from the unit square $[0, 1] \times [0, 1]$. What is the expected number of points on the skyline of P? [Hint: Consider the points in the decreasing order of x, as labeled in the figure below. You may assume that no two coordinates are the same, which, indeed, happens with probability 0.]



p_{10}
p_5
p_2
p_8
p_9
p_6
p_3
p_7
p_1
p_4