COMP 3711 Design and Analysis of Algorithms
Spring 2015 Final Exam

1. **Data structures** (12 pts)

   For each of the following tasks, decide what should be the most efficient data structure, and say what the cost is for each operation to be supported (you don't need to describe how to do it). The data structure should occupy linear space.

   1.1 Data: A collection of n (IP address, URL) pairs. Each IP address is an integer, and URL is a string.

   Operations to be supported:

   (a) Given an IP address, find its corresponding URL, or report that it doesn't exist.

   (b) Given a new (IP address, URL) pair, insert it into the data structure.

   1.2 Data: A collection of n (IP address, URL) pairs. Each IP address is an integer, and URL is a string.

   Operations to be supported:

   (a) Given an IP address, find its corresponding URL; if it doesn't exist, find the closest IP address and its corresponding URL.

   (b) Given a new (IP address, URL) pair, insert it into the data structure.

   1.3 Data: A collection of n IP addresses. Each IP address is an integer.

   Operations to be supported:

   (a) Find the smallest IP address.

   (b) Delete the smallest IP address from the data structure.

   (c) Given a new IP address, insert it into the data structure.

   1.4 Data: A collection of n IP addresses. Each IP address is an integer.

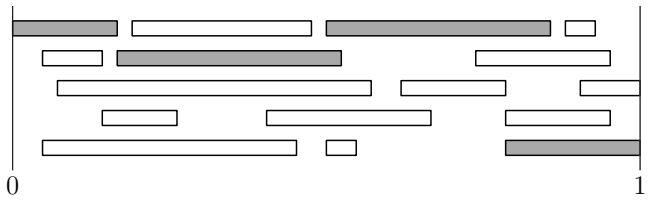
   Operations to be supported:

   (a) Given two IP addresses x and y, check if they are connected.

   (b) A network link is built between two given IP addresses x and y. Reflect this change in your data structure.

2. **Interval covering** (15 pts)

   This problem is similar to the homework question but it is different. Suppose your company wants to build cell phone base stations to cover a highway. However, not every location is allowed to build, and the government has designated a certain number of locations. Depending on how far the location is from the highway, its coverage on the highway also varies. Thus, this problem can be generally modeled as follows. The highway can be thought of as the real line from 0 to 1, and each location covers a certain interval $[x, y] \subseteq [0, 1]$ if a base station is built there. You are given a total of n such intervals and are guaranteed that all of them together cover $[0, 1]$ (otherwise there is no solution to the problem). Your job is to design an algorithm that finds the minimum number of intervals that together cover $[0, 1]$. The figure belows shows an example where an optimal solution consists of the 4 shaded intervals (the optimal solution may not be unique). The intervals are given as two arrays $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ where the i-th interval is $[x_i, y_i]$. If you use a greedy algorithm, you must prove that it is correct.



3. **Shortest symmetric supersequence** (13 pts)

   Recall that one of the homework questions is to find the longest symmetric subsequence of a given string. Here your job is to design an algorithm to find the shortest symmetric *supersequence*. For example, for the string `ACBAC`, the shortest symmetric supersequence is `CACBCAC` or `ACBABCA`, both of which have length 7. Your algorithm only needs to output the optimal length, not the actual symmetric supersequence.

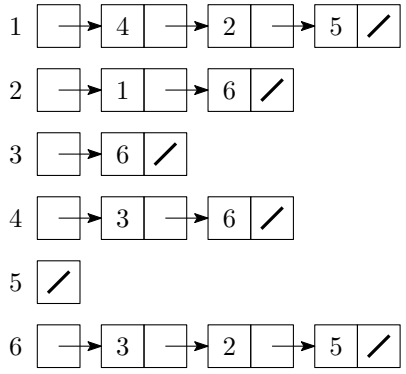
4. **BFS and DFS** (15 pts)

   The figure below shows the adjacency list representation of a directed graph.

   (a) (5 pts) Draw the graph represented.

   (b) (5 pts) Draw the BFS-tree of the graph, using vertex 1 as the source vertex.

   (c) (5 pts) Draw the DFS-tree of the graph, using vertex 1 as the source vertex.



5. **Minimum spanning tree and the cut lemma** (15 pts)

   Let G be a connected undirected graph with distinct weights on the edges, and let T be the MST of G. Recall that the cut lemma states: For *any* cut $(S, V - S)$ of the graph, the minimum-weight edge e crossing the cut must belong to T. Prove or disprove (i.e., give a counter example) the following "reverse cut lemmas".

   (a) Every edge e of T must be the minimum-weight edge crossing *any* cut of G.

   (b) Every edge e of T must be the minimum-weight edge crossing *some* cut of G.

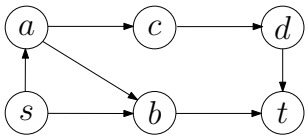
6. **Longest path in a DAG** (15 pts)

   You have solved the longest path problem on a directed acyclic graph when either the vertices or the edges have weights. Now please design an algorithm that solves the most general case where both edges and vertices have weights. More precisely, given the weight $w(v)$ for every $v \in V$ and the weight $w(u, v)$ for every edge $(u, v) \in E$, a source vertex s and a destination t, your algorithm should find the path from s to t whose total weight (sum of all edge weights and vertex weights along the path) is maximized. Your algorithm should output the actual longest path.

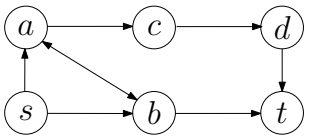
7. **Random routing** (15 pts)

We can model a computer network as a directed graph, where each node is a computer and a directed edge (u, v) means that u can send messages to v. Ideally, messages should be routed along the shortest path, but this information may not be available since no node has a global view of the entire network. Thus, decisions have to be made by each node locally. There are sophisticated routing protocols (which you can learn in COMP 4621); here we just consider one of the simplest routing protocols: RANDOM. In RANDOM, each node simply randomly chooses an outgoing link, and forwards the message along that link, hoping the message can find its way eventually.

Consider the graph G_1 below. Suppose we have a message at the source node s intending to reach destination t. Using RANDOM, s will forward the message to a or b, each with probability $1/2$. If b gets it, it will forward the message to t in the next step, as t is its only outgoing neighbor. If a gets the message, it will choose b or c in the next step, each with probability $1/2$, and so on so forth.



G_1



G_2

(a) (2 pts) On graph G_1, what's the probability that RANDOM indeed chooses the shortest path from s to t to route the message?

(b) (4 pts) On graph G_1, what's the expected length (in terms of the number of edges) of the path chosen by RANDOM?

(c) (9 pts) Answer question (a) and (b) on graph G_2. Note that the only difference is that G_2 has one more directed edge from b to a. [Hint: For question (b), compute the expected contribution of each edge to the path and use linearity of expectation. The key is: how long will the message stay in the loop $a \leftrightarrow b$ in expectation?]

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.

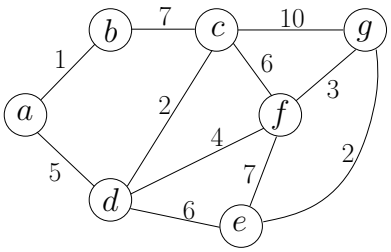
3. **Huffman coding** (6 pts)

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.

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

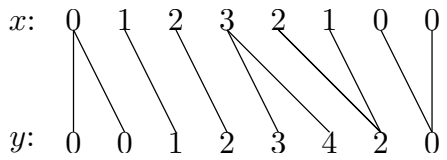
6. **Dynamic time warping** (12 pts)

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 $x[i]$;

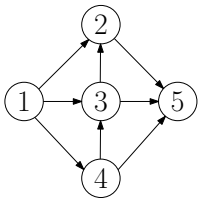
(b) (no crossing) if for $i_1 \leq i_2$, $x[i_1]$ matches with $y[j_1]$ and $x[i_2]$ matches with $y[j_2]$, we must have $j_1 \leq j_2$.

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.

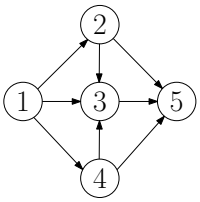


7. **Path finding** (12 pts)

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.



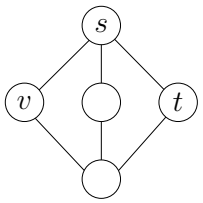
Hamiltonian path: 1, 4, 3, 2, 5



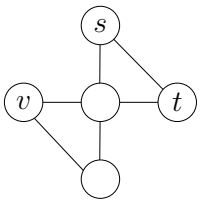
No Hamiltonian path

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

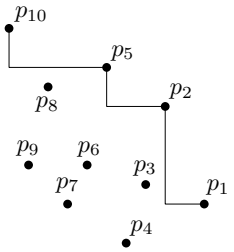


No

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.]



COMP 3711 Design and Analysis of Algorithms
Fall 2014 Final Exam

1. **Time Complexity** (8 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 there 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	$\Theta(n^2)$	$\Theta(n^2/\log n)$	
2	$O(n^3)$	$\Omega(2^{\sqrt{n}})$	
3	$\Theta(\log n)$	$O(\log\log n)$	
4	$\Theta(\log^3 n)$	$\Theta(\sqrt[3]{n})$	
5	$O(n^2)$	$O(n^{2.31})$	
6	$\Omega(n^2)$	$O(n^{2.5})$	
7	$\Omega(n^3)$	$O(n^{2.81})$	
8	$\Theta(\sqrt{\log n})$	$\Theta(\log n/\log\log n)$	

2. **Divide and Conquer** (12 pts)

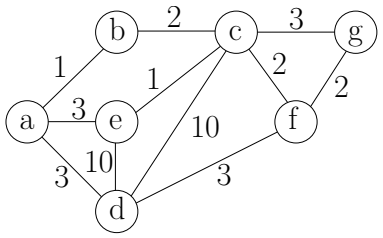
   You are given an array $A[1..n]$ that contains a sequence of 0 followed by a sequence of 1 (e.g., 0001111111). A contains at least one 0 and one 1.

   (a) (5 pts) Design an $O(\log n)$-time algorithm that finds the position k of the last 0, i.e., $A[k] = 0$ and $A[k+1] = 1$.

   (b) (7 pts) Suppose you have been told that k is much smaller than n. Design an $O(\log k)$-time algorithm that finds the position k of the last 0. [Hint: you can re-use the solution of part (a).]

3. **Prim vs. Dijkstra** (16 pts)

   Prim's minimum spanning tree algorithm and Dijkstra's shortest path algorithm are very similar, but with crucial differences. Run both algorithms on the following graph, and show the partial MST / shortest path tree after every new edge has been added. The starting vertex for both algorithms is "a".



4. **Minimum Spanning Tree** (20 pts)

   All MST algorithm studied in class use the notion of "safe" edges, and the MST lemma shows that any safe edge can be added to a partial MST. There is an alternate formulation. Let G be a weighted undirected graph, where the edge weights are distinct. We say that an edge e is *dangerous* if it is the longest edge in ***some*** cycle in G.

   (a) (10 pts) Prove that no MST of G can contain a dangerous edge.

   (b) (10 pts) Given the graph G and an edge e, design and analyze an algorithm to check whether e is dangerous. For full credits, your algorithm should run in $O(V + E)$ time.

5. **Greedy Algorithm** (15 pts)

   In the old days, files were stored on tapes rather than disks. Reading a file from tape isn't like reading a file from disk; first we have to fast-forward past all the other files, and that takes a significant amount of time. Suppose we have a set of n files that we want to store on a tape, where file i has length $L[i]$. Given the array $L[1..n]$, your job is to design an algorithm to find the optimal order to store these files on a tape to minimize the cost. Note that the cost of reading file i is total length of all files stored before it, including file i itself. Your algorithm should run in $O(n \log n)$ time.

   (a) (5 pts) Suppose each file is accessed with equal probability, and you want to minimize the expected cost. For example, if $L[1] = 3, L[2] = 6, L[3] = 2$, you would want to use the order $(3, 1, 2)$. This way, the expected cost is $2/3 + (2+3)/3 + (2+3+6)/3 = 6$, which is optimal. You need to prove the optimality of your algorithm.

   (b) (10 pts) Suppose the files are not accessed uniformly; file i will have probability $p[i]$ to be accessed. Given the array $L[1..n]$ and $p[1..n]$, how would you find an ordering that minimizes the expected cost? For example, if $L[1] = 3, L[2] = 6, L[3] = 2$ and $p[1] = 1/6, p[2] = 1/2, p[3] = 1/3$, then the optimal ordering would be $(3, 2, 1)$, with an expected cost of $2/3 + (2 + 6)/2 + (2 + 6 + 3)/6 = 6.5$. Remember to prove the optimality of your algorithm.

6. **Recurrences and Dynamic Programming** (14 pts)

   Consider the following recursive algorithm RA, which takes as input a positive integer n.

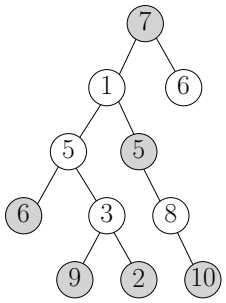
```
RA(n):
    if n = 1 then return 10
    s ← 5
    for i ← 1 to n − 1 do
        s ← s + 3 · RA(i)
    return s
```

(a) (7 pts) Describe the recurrence for the running time $T(n)$ of RA, and use the substitution method (i.e., induction) to prove that $T(n) = O(2^n)$.

(b) (7 pts) Provide the pseudocode of a dynamic programming algorithm that achieves the same result as RA and analyze its running time.

7. **Maximum Independent Set** (15 pts)

In class we learned about the maximum independent set problem on an array, where we want to pick a subset of non-adjacent numbers from an array to maximum their sum. Here we generalize this problem to a binary tree. More precisely, given a binary tree where each node has a number, your job is to design an algorithm to pick a subset of nodes to maximize their sum. Still, adjacent nodes cannot be picked. The figure below shows an example where the grayed nodes form the optimal solution. You will get full credits if your algorithm runs in $O(n)$ time, where n is the number of nodes in the binary tree. Your algorithm just needs to output the sum, not the actual nodes picked.



COMP 3711 Design and Analysis of Algorithms
Fall 2012 Final Exam

1. **Quick-Answer Questions** (15 pts)

   1.1 (6 pts) Please arrange the following functions in asymptotic ascending order (e.g., n, n^2, n^3): (a) n; (b) $\log n$; (c) $\log \log n$; (d) $\log^* n$; (e) $2^{2 \log n}$.

   1.2 (5 pts) Given n integers in the range 0 to n^3, what is the fastest sorting algorithm (asymptotically) to sort them?

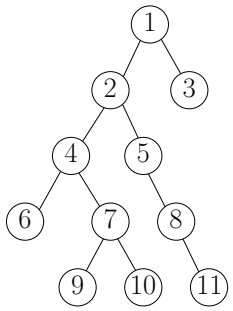
   1.3 (4 pts) Put "$\forall$" and "$\exists$" into the following blankets.

   The precise meaning of "the problem can be solved in $O(n)$ time" is:

   _____ a correct algorithm for the problem, _____ $c > 0$, _____ $n > 0$, such that _____ input of size n, the running time of the algorithm on the input is at most cn.

2. **Longest Path in a Tree** (10 pts)

   Design an algorithm that, given a binary tree, finds the longest path in the tree. For example, in the tree below, the longest path is 9-7-4-2-5-8-11. Your algorithm just needs to output the two endpoints of the path, i.e., 9 and 11 in this example. For full credits, your algorithm should run in $O(n)$ time, where n is the number of nodes in the tree.



3. **Shortest Path on an Chessboard** (10 pts)

   In the game of chess, the Knight can move two squares horizontally and one square vertically, or two squares vertically and one square horizontally. The complete move therefore looks like the letter "L". You are given two squares on an $n \times n$ chessboard. Design an algorithm to determine the shortest sequence of knight moves from one square to the other. For full credits, your algorithm should run in $O(n^2)$ time.

4. **Bottleneck Spanning Tree** (10 pts)

   A *bottleneck spanning tree* T of an undirected, weighted graph G is a spanning tree of G whose largest edge weight is minimum over all spanning trees of G. We say that the value of the bottleneck spanning tree is the weight of the maximum-weight edge in T. Give a linear-time algorithm that, given a graph G and an integer b, determines whether the value of the bottleneck spanning tree is at most b.

5. **Non-adjacent Elements in an Array** (15 pts)

   Given an array of n positive numbers $A[1], A[2], \ldots, A[n]$. Our goal is to pick out a subset of non-adjacent elements whose sum is maximized. For example, if the array is $(1, 8, 6, 3, 6)$, then the elements chosen should be $A[2]$ and $A[5]$, whose sum is 14.

(a) (5 pts) Give an example to show that the following algorithm does not always find a subset of non-adjacent elements whose sum is maximized.

```
Start with S equal to the empty set
While some elements remain in A
  Pick the largest A[i]
  Add A[i] to S
  Mark A[i] and its neighbors as deleted
Endwhile
Return S
```

(b) (10 pts) Give an algorithm based on dynamic programming that finds a subset of non-adjacent elements in A whose sum is maximized. For full credits, the running time should be $O(n)$. All elements in A are positive.

6. **Longest Balanced Subsequence** (15 pts)

A string of parentheses is said to be balanced if the left- and right-parentheses in the string can be paired off properly. For example, the strings (()) and ()() are both balanced, while the string ((()))(is not. Given a string S of length n consisting of parentheses, design an algorithm to find the longest subsequence of S that is balanced. For this problem, you will gain full marks as long as your algorithm is correct and runs in polynomial time.

7. **Greedy Algorithm** (15 + 10 pts)

A teacher assigns a homework with n questions at the beginning of the first day of class. Each homework question carries a certain number of marks, if submitted on or before a specified deadline; otherwise it earns no marks. Each homework question takes exactly one day to complete. Your task is to find a schedule to finish the homework questions so as to maximize marks earned.

For instance, suppose there are seven questions with deadlines and marks as follows:

Problem	1	2	3	4	5	6	7
Deadline	1	1	3	3	2	2	6
Marks	6	7	2	1	4	5	1

Then the maximum number of marks one can obtain is 15, which can be achieved by completing the problems in the order of 2, 6, 3, 1, 7, 5, 4. Note that there are also other schedules that obtain the same marks.

For this problem, you will gain full marks as long as your algorithm is correct and runs in polynomial time. You get 10 bonus marks if your algorithm runs in time $O(n \log n)$.

8. **Huffman Coding** (10 pts)

What is the optimal Huffman code for the following set of frequencies, based on the first 8 Fibonacci numbers?

a	b	c	d	e	f	g	h
1	1	2	3	5	8	13	21

Can you generalize your answer to find the optimal code when the frequencies are the first n Fibonacci numbers? Recall that the Fibonacci numbers are defined inductively as $f_1 = f_2 = 1, f_n = f_{n-2} + f_{n-1}$ for $n \geq 3$. You need to justify your answer.

COMP 3711 Design and Analysis of Algorithms
Fall 2011 Final Exam

1. **Quick-Answer Questions** ($5 \times 4 = 20$ pts)

   1.1 Please arrange the following functions in asymptotic ascending order (e.g., n, n^2, n^3): (a) n; (b) $n \log n$; (c) $n \log^* n$; (d) $\frac{n}{\log n}$; (e) $n^{1.0001}$.

   1.2 What is the minimum number of nodes in an AVL tree of height 4? Note that the AVL tree of height 1 has 1 node; the AVL tree of height 2 has 2 nodes at the minimum.

   1.3 Radix sort may take $O(n)$ time to sort n integers. This breaks the $\Omega(n \log n)$ sorting lower bound we have proved. Is this a contradiction? Why?

   1.4 Let G be a weighted, connected, undirected graph, where all the weights are distinct. Dijkstra's algorithm can be used to find all the shortest paths from a given source vertex. These paths form a tree, called the *shortest path tree*. Is it always the same as the minimum spanning tree of G? If yes, give a proof; otherwise give a counter example.

2. **Majority** (10 pts)

   Assume you have an array $A[1..n]$ of n elements. A *majority element* of A is any element occurring more than $n/2$ times (e.g., if $n = 8$, then a majority element should occur at least 5 times). Your task is to design an algorithm that finds a majority element, or reports that so such element exists.

   (a) This problem can be solved by simply sorting the array in $O(n \log n)$ time. But to have more fun (or torture?) we have decided that you are not allowed to order the elements, but can only test them for equality. More precisely, the only way you can access the elements is to check whether two elements are equal or not, so you cannot sort the elements. Can you still design an $O(n \log n)$-time algorithm for this problem? [Hint: Divide-and-conquer.]

   (b) You will get 10 bonus points if you can design an $O(n)$-time algorithm. Of course, you are still only allowed to use equality tests on the elements. (Warning: This is very difficult — do not spend too much time on it unless you have finished all other questions.)

3. **Topological Sort** (8 pts)

   We have shown in class how to do topological sort by modifying the BFS algorithm. We can also design a topological sort algorithm based on DFS. Please complete the following DFS-based topological sort algorithm.

Algorithm 1: Main algorithm

```
foreach u ∈ V do
    color[u]= WHITE;
end
foreach u ∈ V do
    if color[u]=WHITE and in-degree(u) = 0 then
        DFS-visit(u);
    end
end
```

Algorithm 2: DFS-visit(u)

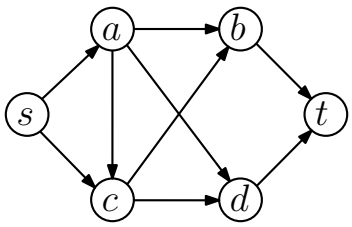
4. **Minimum Spanning Tree** (10 pts)

   Let $G = (V, E)$ be a weighted, undirected, connected graph, whose weights are integers from $\{1, \ldots, k\}$. Design an algorithm to compute the minimum spanning tree of G in $O(|E| \log k)$ time. [Hint: Modify Prim's algorithm.]

5. **Counting Paths** (12 pts)

   Let $G = (V, E)$ be a connected, directed, acyclic graph (DAG). Let s, t be two vertices from G. Please design an algorithm that counts the number of different paths from s to t in G. For full credits, your algorithm should run in $O(|E|)$.

   For example, the graph below has 6 different paths from s to t, as follows. (Your algorithm just needs to output the number 6, not the actual paths.)



1. $s \to a \to b \to t$;
2. $s \to a \to c \to b \to t$;
3. $s \to a \to c \to d \to t$;
4. $s \to a \to d \to t$;
5. $s \to c \to b \to t$;
6. $s \to c \to d \to t$;

6. **Consulting Firm** (20 pts)

   Suppose you're running a lightweight consulting firm with a few friends. Each month, you can either run your business from an office in Hong Kong or from an office in Beijing. Since your clients travel a lot, the demand varies from month to month, and the profits are different if your office is in different cities. In month i, let the profit in Hong Kong be H_i, and the profit in Beijing be B_i. But every time you switch cities, you incur a fixed moving cost M. Given the H_i and B_i values for a sequence of n months, we would like to find the optimal plan for the locations where you run your business for these n months that maximizes your net profit (i.e., total profit minus the moving costs). Suppose you start your business in Hong Kong in month 1.

   Example: Suppose $n = 4, M = 10$, and the profits are given by the following table:

	Month 1	Month 2	Month 3	Month 4
HK	80	90	30	20
BJ	20	30	80	70

   Then the optimal plan would be the sequence [HK, HK, BJ, BJ], with a net profit of $80 + 90 - 10 + 80 + 70 = 310$.

   (a) (4 pts) The following greedy strategy looks rather reasonable: If in the next month, the profit in the other city is M more than the profit of the current city, then we switch. More precisely, if we are currently in HK in month i, and $B_{i+1} - H_{i+1} > M$, we move to Beijing for month $i + 1$; if we are currently in BJ in month i, and

$H_{i+1} - B_{i+1} > M$, we move to Hong Kong for month $i+1$. This greedy algorithm, however, is not always correct. Please give an example in which this algorithm returns a non-optimal plan.

(b) (10 pts) Design and analyze an algorithm that always finds the optimal plan. For full credits, your algorithm should run in $O(n)$ time. Your algorithm should return both the maximum net profit and the optimal plan that achieves this profit.

(c) (6 pts) Now consider the case that your business may incur a loss (i.e., negative profit) for certain months due to bad economy. Further suppose that you have the freedom of choosing a starting month and a finishing month. Please design an algorithm that finds the best time to start and finish your business given the profit table and M. Your algorithm just needs to output the optimal net profit, not the actual plan or the starting/finishing month. Your starting city is still Hong Kong. You may assume that the profit is positive for at least one month in HK (otherwise you may not want to start the business at all).

Example: Suppose $n = 5, M = 10$, and the profits are given by the following table:

	Month 1	Month 2	Month 3	Month 4	Month 5
HK	-10	30	-30	40	-5
BJ	20	30	-5	10	5

Then the optimal plan is to start in month 2 and finish in month 4, with the sequence [HK, BJ, HK]. The net profit is $30 - 10 - 5 - 10 + 40 = 45$.

For full credits, your algorithm should run in time $O(n)$. Note that if $M = \infty$, then you wouldn't want to move at all, and this problem degenerates to the *maximum contiguous subarray problem*. In this case, your algorithm will actually beat the $O(n \log n)$-time algorithm given in class!

7. **Greedy Algorithm** (10 pts)

Given two sequences $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_n)$, design an algorithm that determines if X is a subsequence of Y. You can assume $m \leq n$. For example, if $X =$ `BCBA` and $Y =$ `ABCBDAB`, then the answer is "yes". But if $X =$ `BDBA`, then the answer is "no". For full credits your algorithm should run in $O(n)$ time. Remember to argue for the correctness of your algorithm unless it is obvious.

8. **Huffman Coding** (10 pts)

Please design a Huffman code for the following alphabet of 7 letters:

Character	Number of occurrences
a	34
b	22
c	20
d	34
e	100
f	8
g	12