Final Exam

Student Information

Student Name: _____

Student ID: _____

- You can bring one sheet of A4 paper (cheat sheet, both sides) for the final exam.
- Time: **9:00 - 11:00am, 16 Dec.**
- This exam is consisted of 12 printed pages.

Problems	Points	Mark
Section 1	20	
Section 2	20	
Section 3	20	
Section 4	20	
Section 5	20	
Full Points	**100**	
Bonus Problem	5	
Total Points	**105**	

Section 1: Single source shortest path, Dijkstra's, Bellman Ford (20p)

Problem 1: (2p) After relaxing edge(s) in a graph, numbers associated with which of the following elements of the graph are likely to change? (choose one)
Note: 'numbers associated' do not refer to the number of vertices or edges.

1. Edges
2. Vertices
3. Edges and vertices
4. Depends on the type of the graph

Problem 2: (4p)

Q1. (2p) Can Djikstra's algorithm compute shortest paths correctly for graphs with positive weight cycles?

Q2. (2p) If you answered yes to the above question, explain in not more than two sentences, how the algorithm prevents itself from going round and round in a positive cycle. If you answered no, explain in not more than two sentences why the algorithm can not compute shortest paths correctly for graphs with positive weight cycles.

Problem 3: (4p)

Q1. (1p) State the optimal substructure property in the context of single source shortest path algorithms. (1 mark)

Q2. (3p) Prove the validity of this property.

Problem 4: (5p) You have built a network of your Facebook friends. Each vertex of this network represents an individual, and each edge denotes the distance in kilometres between the geographical locations of the two individuals at the ends of the edge. You want to find out the degree of separation of this network. Which algorithm will give you the smallest running time? Justify your answer in not more than two sentences. (2 points)

Note: The degree of separation of a graph is the average of the shortest distance between all pairs of vertices.

Problem 5: (5p) In the given graph, find the shortest path from the vertex A to all other vertices. To get credit for this problem you need to mention the algorithm you are using, and show the steps in executing the algorithm. You are free to use any order of relaxation.

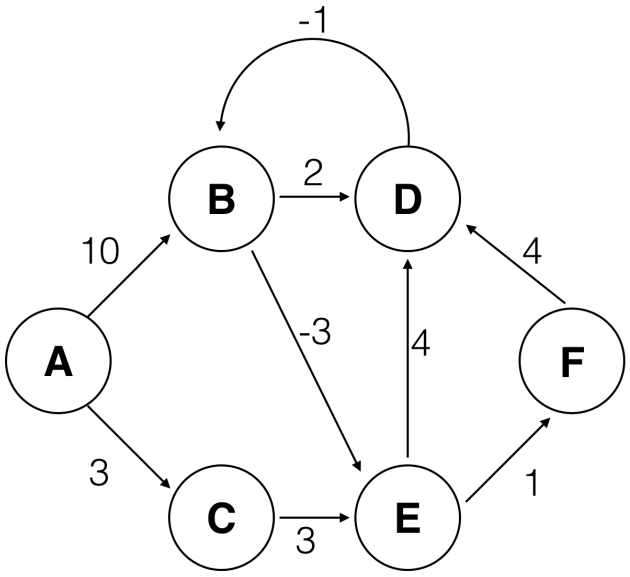

Figure 1: Graph for Problem 5

Section 2: Heap, BST and AVL tree, Hashing (20p)

Problem 1: (6p) See Fig 2 and 3 below and answer **Q1, Q2,** and **Q3**.

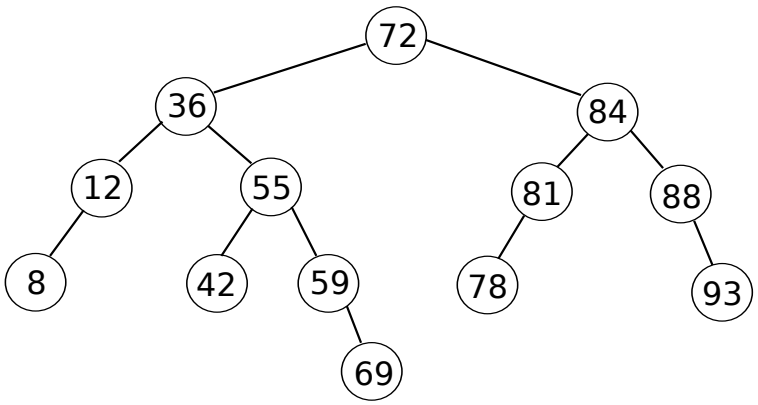

Figure 2: A tree

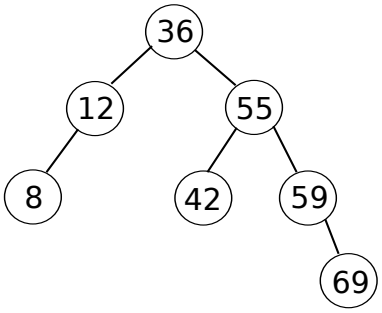

Figure 3: Another tree

Q1. (**2p**) Consider the AVL tree given in Figure 2. Write down for every node the heights of its two subtrees. Note: empty leaves have height -1.

Q2. (**2p**) You need to insert the keys $\{75, 79, 82\}$ in the tree in Figure 2. Write down a possible insertion order of these keys such that you can insert them without any tree rotations in Figure 2.

Q3. (**2p**) Consider the small tree in Figure 3 above. Insert the keys in the following order$(9, 11)$ and perform AVL tree rotations when necessary to maintain the AVL property. Show the resulting tree after each insertion step.

Problem 2: (6p) Suppose you have a hash table with slots $\{0, \ldots, 11\}$. You want to use open addressing hashing. You can choose between these three hash functions for generating a probing sequence. k is the key to be inserted. It is an integer here. i is the probing parameter that starts at $i = 0$.

- $h_1(k, i) = (3k + 5i) mod\, 12$
- $h_2(k, i) = (3k + 36i) mod\, 12$
- $h_3(k, i) = (3k + 3i) mod\, 12$

Using the information above answer **Q1** and **Q2.** below.

Q1. (**3p**) Given a fixed key k, which of the hash functions does not explore the whole hash table space as we go through the probing parameter i?

Q2. (**3p**) Suppose $h_R(k, 0)$ causes a collision in your hash table with some other previously stored key. The normal procedure would be to try $h_R(k, i)$ for various i until you get a free slot. For which of the three hash functions h_R shown above you will never be able to resolve a collision, no matter what value of i you will choose? Why is that so? (It is an example of a failed design.)

Problem 3: (8p) Suppose you have a search tree T that supports the operations `insert(x)` and `extractmin()`. You do NOT need to write a code for `insert(x)` and `extractmin()`. Answer **Q1** to **Q4.**

Q1. (**2p**) Write down an algorithm using pseudocode, that uses these two operations of the tree to implement sorting of n integers. Input is: n integers $(x_1, \ldots, x_n)$. The output should be a python list with these integers sorted. You can assume that, if $bstree$ is an instance of the search tree class, then you can call `bstree.insert(x)` and `x= bstree.extractmin()`

Q2. (**2p**) If `insert(x)` and `extractmin(T)` are both $\mathcal{O}(1)$ (where $\mathcal{O}(\cdot)$ refers here to the number of elements in the tree), what would be a tight $big - O$ running time of this sorting algorithm as a function of the number n of integers to be sorted?

Q3. (**2p**) Why there cannot exist a comparison-based search tree that supports both operations `insert(x)` and `extractmin(T)` in $\mathcal{O}(1)$ running time?

Q4. (**2p**) How about `insert(x)` and `extractmin(T)` in $\mathcal{O}(\log(n))$ running time (again, where $\mathcal{O}(\cdot)$ refers here to the number of elements in the tree)? Can such an algorithm exist?

Section 3: Graph, DFS, BFS (20p)

Problem 1: (8p) Consider the graph $G = (V, E)$ in Fig. 4 below. Assume that we need to examine vertices in $adj(v), v \in V$, in **alphabetical** order. Let node A be the starting point of the graph.

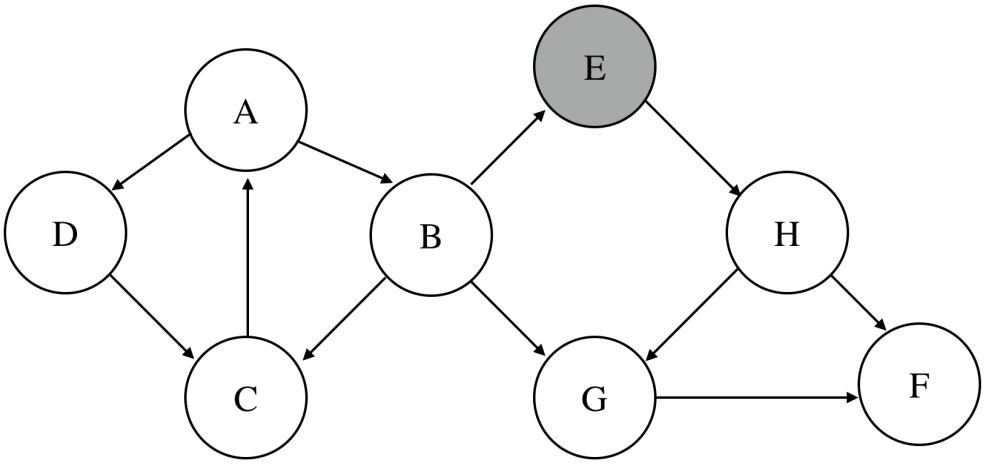

Figure 4: Graph G

Q1. **(2p)** Run DFS on G starting from node A until you have explored the entire graph and write down both time stamps (entrance and exit) on each node in Fig. 4.

Q2. **(3p)** Based on your DFS result in **Q1**, indicate whether each edge in Fig. 4 is a tree, forward, cross, or back edge.

Q3. **(3p)** Which of the two algorithms: BFS or DFS, will visit the **least** number of nodes before reaching the goal node E (from A) in the graph? Write down all the nodes that you need to visit to reach E using both algorithms in your answer.
Note: Please follow the **alphabetical** order when examining $adj(v)$ of any node v in G.

Problem 2: (6p) For **Q1** to **Q3** below, choose which of the three graph representations: **adjacency list**, **adjacency matrix**, or **implicit representation** is most suitable (least running time and least space complexity). **Justify** your reason.

Q1. (2p) Unweighted, directed, and very dense graph, with the degree of each vertex being $O(n)$ (i.e. the number of edges incident on each vertex is $O(n)$).

Q2. (2p) A heap with n elements.

Q3. (2p) Weighted, undirected, and the degree of each vertex is $O(1)$ (i.e. the number of edges incident on each vertex is $O(1)$).

Problem 3: (6p) The *eccentricity* $\epsilon(u)$ of a vertex u in a **connected**, **undirected**, and **unweighted** graph $G = (V, E)$ is the maximum shortest distance from u to any other vertex in the graph. In other words, if $\delta(u, v)$ is the shortest path from u to v, then $\epsilon(u) = \max_{v \in V} \delta(u, v)$. Recall that an undirected graph is connected when there is a path between every pair of its vertices.

Describe clearly in not more than 5 sentences, an $O(V + E)$ algorithm to find the eccentricity of a node u in G, where V is the total number of vertices and E is the total number of edges.

Note: You can describe your algorithm using pseudocode and plain english (we will not consider programming syntax for grading) and show clearly why the running time of your algorithm is $O(V + E)$.

Section 4 : Asymptotic Notation, Complexity, Master theorem, Sorting (20 p)

Problem 1: (7p) Compute the **Θ-complexity** of the following recurrence equations.

Q1. (**3p**) $T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + T(\frac{n}{8}) + n$

Q2. (**4p**) $T(n) = 2T(\frac{n}{2}) + \frac{n}{\lg n}$

Problem 2: (5p) Recall all the sorting algorithms we have learned during class: merge sort, insertion sort, heap sort, counting sort and radix sort. Now we want to sort n dates represented as day-month-year, e.g., 31-12-2015, and all of them are from the 21st century. Answer Q1 and Q2.

Q1. (**3p**) Which algorithm do you think is the most efficient one for this data sorting task? Describe the sorting algorithm you choose.

Q2. (**2p**) Analyze and compute the Θ-complexity of this algorithm.

Problem 3 : (8p) Recall the pseudo code of merge sort, answer the the following questions. Note A represents an array of n elements, and p, r are the starting and ending position of the array, respectively, i.e., $n = r - p + 1$. $A[r]$ denotes the rth element in the array.

```
Merge sort (A, p, r) :
if p < r then
   q = ⌊(p+r)/2⌋
   Merge sort (A, p, q)
   Merge sort (A, q + 1, r)
   Merge(A, p, q, r)
end if
```

Q1. (**2p**) Write down the recurrence formula for merge sort, and solve it to get the complexity.

Q2. (**6p**) Suppose now we have another sorting algorithm called Q-sort, the pseudo code is as follows. Instead of simply setting $q = \lfloor \frac{p+r}{2} \rfloor$, Q-sort uses another function called 'Partition' to determine q. Regarding Q-sort, what are the **best case** and **worst case** performance of this algorithm, and **explain the situation** that will give such best or worst case? Please write down the recurrence formula and calculate the complexity. **Hint:** Try a simple array to help understand the Partition function and the performance is related to it.

```
Q-sort (A, p, r) :
if p < r then
   q = Partition(A, p, r)
   Q-sort (A, p, q)
   Q-sort (A, q + 1, r)
end if
```

```
Partition (A, p, r) :
if p < r then
   x = A[r]
   i = p − 1
   for j = p to r − 1 do
      if A[j] < x then
         i = i + 1
         exchange A[i] with A[j]
      end if
      exchange A[i + 1] with A[r]
   end for
end if
return i + 1
```

Section 5: Dynamic programming & Greedy Algorithm, P & NP (20p)

Problem 1: (10p) In country X, only coins of value \$1, \$5, \$7 are available. You want to make up a total value of \$10 with the minimal number of coins (e.g. using 10 x \$1 coin can make up a total of \$10 but clearly this is not the optimal solution).

Q1. (5p) Please solve the problem in a **Dynamic Programming approach**. Write down the DP equation. Explain your solution. (You can answer with a table, pseudocode, text description, or any format that can clearly present your idea.)

Q2. (5p) Please solve the problem in a **Greedy approach**. Explain your solution. Is your solution optimal? Why?

Problem 2: (10p) Prove that **Independent Set** is NP-Complete. You **MUST NOT** reduce it from 3-SAT. (Independent Set: set of nodes, where all pairs are NOT adjacent to each other)

Bonus Problem

Problem 1: (5p) Five students $s_1, s_2, ..., s_5$ are sitting together. They decide to play a game where a student s_i will whisper a particular phrase to another student s_j. Let $w(s_i, s_j)$ denote the level of distortion of the phrase while being transmitted from s_i to s_j. $w(s_i, s_j) = 0$ denotes the phrase is completely distorted, and $w(s_i, s_j) = 1$ denotes the phrase is completely undistorted. All other values of $w(s_i, s_j)$ lie between 0 and 1. Design an algorithm to detect the path of least distortion between two given students. You can either explain the algorithm you design in natural language (in not more than ten sentences) or express the algorithm in pseudocode.