ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Fall Semester 2020 Prof. R. Wattenhofer

Computational Thinking
Fall Semester 2020

Monday, 15th of February 2021, 9:00-11:00

Do not open before the official start of the exam!

The exam lasts for 120 minutes and comprises 120 points. There is one block of questions for each lecture chapter.

You can answer in German or English, or mix German and English.

Unless explicitly stated, you do **not** have to justify your answers. However, writing down your thoughts (math, text or annotated sketches) will help us to understand your approach. This will allow us to award points even if your solution is wrong. Please make sure your writing is readable.

Some questions ask you to fill in answers in a template. In case you made a mess, and want a fresh start, you find fill-in replacements at the end of the exam.

Please write your name and student number on every extra sheet. Please write your name and student number in the following fields on this cover sheet.

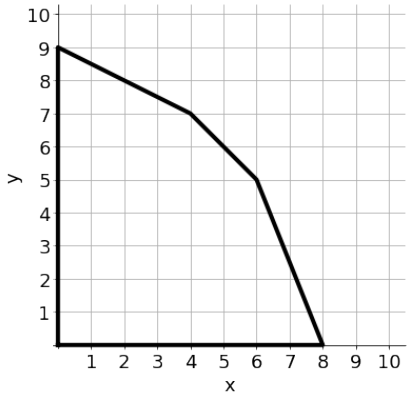
Family Name	**First Name**	**Student Number**

Task	**Achieved Points**	**Maximum Points**
1 - Algorithms		19
2 - Complexity		18
3 - Cryptography		18
4 - Data and Storage		17
5 - Machine Learning		16
6 - Neural Networks		14
7 - Computability		18
Total		**120**

1 Algorithms (19 points)

Multiple Choice

a) `[2 points]` Which solution is optimal for a linear programming problem where we try to maximize $2x + 2y$ given the following feasible region enclosed within the black lines?



- ○ (0, 9)
- ○ (6, 5)
- ○ (7, 7)
- ○ None of the above.

Optimal Change

An intergalactic supermarket chain wants you to program a cashier robot. You have to ensure that the robot gives clients the correct amount of change in their currency. The clients want to receive as few coins as possible. Given some foreign currency set of coins C (e.g. $C = [1, 5, 2]$), $|C| = n$ and a total value of change v (e.g. $v = 11$) you need to determine the **minimum number of coins** to match the correct value v. You have an infinite supply of copies of each of the coins in the coin set C (in our example the minimum number of change coins is 3). Assume that coins C and total value v are positive integers and that there always exists a coin in C with value of 1. The supermarket chain provided you with this algorithm:

```
1  def change(C, v):
2      C.sort(reverse=True)     # descending order
3      n = len(C)
4      coins = 0
5      for i in range(n):
6          while v >= C[i]:
7              v = v - C[i]
8              coins = coins + 1
9      return coins
```

a) `[3 points]` What is the time complexity of `change(C, v)`, as a function of v and n?

b) [5 points] The `change(C, v)` algorithm above does not necessarily return the coin set with the smallest number of coins. Find an example where it fails.

c) [9 points] Your friend started writing a correct dynamic programming solution for this problem. Can you help finish it? Fill in the blanks in Lines 6-11 below.

Hint: *It may help to sketch the dynamic programming table before filling in the blanks, for example, with* $C = [1, 5, 2]$ *and* $v = 11$.

```
1  def optimalChange(C, v):
2      n = len(C)
3      table = [v for i in range(v + 1)]
4      table[0] = 0
5      for i in range(1, v + 1):
6          for j in range(                                    ):
7              if                                             :
8                  sub_result = table[                ]
9                  if                                         :
10                     table[                ] =
11     return table[                ]
```

MC (2 points)

a) (6, 5). The line from (4, 7) to (6, 5) forms a Pareto optimal frontier - all of the values on it are optimal.

Optimal change (2 points)

a) $O(v + n \log n)$

b) For $C = [1, 6, 10]$ and $v = 12$ this algorithm would use 3 coins $[10, 1, 1]$ while the optimal solution is 2 coins $[6, 6]$.

c) Completed DP solution:

```
def optimalChange(C, v):
    n = len(C)
    table = [v for i in range(v + 1)]
    table[0] = 0
    for i in range(1, v + 1):
        for j in range(n):
            if (C[j] <= i):
                sub_result = table[i - C[j]]
                if (sub_result + 1 < table[i]):
                    table[i] = sub_result + 1
    return table[v]
```

2 Complexity (18 points)

Multiple Choice

a) [4 points] Consider our 2-approximation algorithm for the Vertex Cover problem (Algorithm 2.49, here reprinted for you).

```
def VertexCover_Greedy(G):
    S = ∅
    while E ≠ ∅:
        select an arbitrary edge (u, v) ∈ E
        S = S ∪ {u, v}
        remove all edges from E that are adjacent to u or v
    return S
```

Recall that a graph is a *clique* if any two nodes are connected by an edge. In which of the following graphs does the algorithm always find the optimal solution?

- ○ If the graph is a clique on 2 nodes.
- ○ If the graph is a clique on 3 nodes.
- ○ If the graph is a clique on 4 nodes.
- ○ None of the above is correct.

b) [4 points] Recall our Algorithm 2.54 for the Bin Packing problem:

```
def FirstFit(items, B):
    for each item in items:
      place item in the first bin where it still fits
      if item does not fit into any bin:
        open a new bin, and insert item into the new bin
```

Consider a small modification of this algorithm: we now check the bins in reverse order, first trying to fit the item into the last bin, then into the second last, and so on. When is this a 2-approximation algorithm?

- ○ In any case.
- ○ Only if all items have the same size.
- ○ Only if all items have size at most $\frac{B}{2}$.
- ○ Never.

Reductions

c) `[10 points]` AllOrNothingSAT is a SAT variant, where in every clause, there is either no negated variable or there are only negated variables. Prove that AllOrNothingSAT is NP-hard.

Hint: *Use 3-SAT.*

Solution

a) If the graph is a clique on 3 nodes. For a clique on 3 nodes, the algorithm returns a vertex cover of 2 nodes, which is indeed the optimum in this case.

b) In any case. Going from the other direction makes no difference for the algorithm.

c) We show that AllOrNothingSAT is NP-hard through a reduction from 3SAT: Given a 3SAT formula we transform it to an input of AllOrNothingSAT as follows. Clauses of the form $(\neg x_i \vee x_j \vee x_k)$ are replaced by $(\neg x_i \vee \neg y) \wedge (x_j \vee x_k \vee y)$, where y is a new variable (that is different for each clause being replaced). The latter is satisfiable if and only if the former is. Similarly, clauses of the form $(\neg x_i \vee \neg x_j \vee x_k)$ are replaced by $(\neg x_i \vee \neg x_j \vee \neg y) \wedge (x_k \vee y)$. Clauses of the form $(x_i \vee x_j \vee x_k)$ and $(\neg x_i \vee \neg x_j \vee \neg x_k)$ stay unchanged.

3 Cryptography (18 points)

Multiple Choice

a) [2 points] The function $h(x) = x$ is, for all x:

- ○ Strong collision-resistant, but not one-way.
- ○ One-way, but not strong collision-resistant.
- ○ Strong collision-resistant and one-way.
- ○ Neither strong collision-resistant nor one-way.

b) [2 points] Alice receives the following signed message from Bob: "My computer was hacked, do no longer trust my public key". Which of the following is true.

- ○ Alice can continue to communicate with Bob using his public key.
- ○ Alice should no longer trust messages signed with Bob's private key.
- ○ Alice can trust a new public key from Bob when Bob signs it with his private key.
- ○ All above answers are wrong.

Key Exchange

Suppose we have the following key exchange algorithm: Alice randomly selects k and a from $\{0,1\}^n$ and sends $k_a = k \oplus a$ to Bob. Bob randomly selects b from $\{0,1\}^n$ and sends $k_b = b \oplus k_a$ to Alice. Alice computes and sends to Bob $d = a \oplus k_b$. Bob computes $d \oplus b$ to retrieve the key k.

c) [6 points] Show that the key exchange algorithm is correct meaning that Alice and Bob agree on the same key.

d) `[8 points]` Prove or disprove that can attacker cannot determine the key based on the observable communication.

Solution

MC

a) The function $h(x) = x$ is collision-resistant as each input is mapped to a different output value (itself), but not one-way as the output reveals the input.

b) If f is collision-resistant then Alice cannot find a collision hence another message that maps to the commitment. If f is one-way, then Bob cannot reveal the message that was committed, but he may gain some information on the message (e.g., message length, some bits). Hence one-way functions do not necessarily imply computational hiding commitment schemes.

c) Either the key was compromised and Bob sent that message to warn Alice or a third person sent that message which again implies that either the digital signature scheme was compromised or someone gained access to Bob's private key. In any case, Alice should not trust anything that is signed with Bob's key.

Key Exchange

a) Since $\oplus$ is commutative and $x \oplus x = 0$ we have: $d \oplus b = a \oplus k_b \oplus b = a \oplus b \oplus k_a \oplus b = a \oplus k_a = a \oplus k \oplus a = k$. Hence, the key exchange algorithm is correct.

b) The algorithm is not secure because the public values reveal the private key $k_a \oplus k_b \oplus d = k_a \oplus a = k$.

4 Data and Storage (17 points)

Flight Tickets Database

We consider the PostgreSQL database of an airline. This database contains information about passengers and flights:

boarding_passes(ticket_no, seat_no)
flights(flight_id, flight_no, departure_airport, arrival_airport)
tickets(ticket_no, flight_id, price, passenger_name)

c) [4 points] Describe what the following SQL query returns:

```
SELECT t.*
FROM tickets AS t
LEFT OUTER JOIN boarding_passes AS bp ON t.ticket_no = bp.ticket_no
WHERE bp.ticket_no IS NULL;
```

d) [6 points] Write an SQL query that outputs for each flight the cheapest and the most expensive price of the tickets sold.

e) [7 points] Execute the following SQL query on the given data:

```
SELECT f.flight_id, departure_airport, COUNT(*)
FROM tickets AS t
JOIN flights AS f ON t.flight_id = f.flight_id
WHERE arrival_airport = 'ZRH'
GROUP BY f.flight_id
ORDER BY f.flight_id;
```

ticket_no	flight_id	price	passenger_name
57321	732	230	Matthias Bürger
57402	521	459	Petra Propst
57511	732	180	Lena Bürger
57902	1102	150	Lisa Weber
58140	1029	980	Max Holtzmann
58289	732	250	Michael Probst
58321	1102	160	Felix Maier
58472	1281	80	Matthias Bürger
58781	1394	300	Marie Baumgartner
59293	1102	245	Matthias Bürger

Table 1: Data in tickets table

flight_id	flight_no	departure_airport	arrival_airport
1102	LX3254	HAM	ZRH
1394	LX572	GOT	ZRH
1241	LX982	HAM	PEK
732	LX572	GOT	ZRH
521	LX1575	ZRH	LGW
891	LX572	GOT	ZRH
1281	LX491	GVA	ZRH

Table 2: Data in flights table

f.flight_id	departure_airport	COUNT(*)

MC

a) Probing and chaining are not efficient, all keys are inserted at bucket 9, so lots of collisions. You have to go far through the chain or probing sequence.

b) An INNER JOIN can be replaced with a RIGHT OUTER JOIN with WHERE (column from left table) IS NOT NULL. (A RIGHT OUTER JOIN contains all rows from an INNER JOIN and additionally those where the left table has no corresponding entries. CROSS JOIN forms the Cartesian product, rows with NULL entries as in OUTER JOINs do not arise).

Ticket database

a) This query finds all tickets for which no boarding pass was issued.

b)
```
SELECT flight_id, MIN(price), MAX(price)
FROM tickets
GROUP BY flight_id;
```

c) Results in the following table:

f.flight_id	departure_airport	COUNT(*)
732	GOT	3
1102	HAM	3
1281	GVA	1
1394	GOT	1

5 Machine Learning (16 points)

Multiple Choice

a) [4 points] Suppose you have n distinct points in $\mathbb{R}^2$ belonging to two classes, i.e. $(\boldsymbol{x}, y) \in \mathbb{R}^2 \times \{-1, 1\}$. What is the maximum number of splits a decision tree might need to separate the two classes?

- ○ n
- ○ $n/2$
- ○ $\log n$
- ○ $n - 1$

Differentiable Lasso

Bob has some high-dimensional data about potato farming. He wants to fit a linear model to it. In particular he wants to learn which variables are particularly important for determining the yield. He reads about Ridge and Lasso regularization and decides he wants to incorporate Lasso regularization into his model, since it is better at setting the weights of less important features identically to zero, i.e. feature selection.

b) [4 points] Why is Lasso better than Ridge at setting weights to zero?

Bob wants to use gradient descent, but he learns that Lasso is not differentiable. Rather than learn all about subgradients, he opts for using a differentiable version of Lasso, giving the loss function below:

$$L(\boldsymbol{w}, D) = \frac{1}{n} \sum_{(\boldsymbol{x},y)\in D} (y - \boldsymbol{w}^T \boldsymbol{x})^2 + \lambda \sum_{i=0}^{d-1} \sqrt{w_i^2 + c}$$

where $c > 0$ is some small constant. This way Bob can use gradient descent.

c) [4 points] Find the partial derivative of the regularization term for weight w_j.

Hint: *Chain Rule* $(f(g(x)))' = f'(g(x))g'(x)$

Hint: *Power Rule* $(x^\alpha)' = \alpha x^{\alpha-1}$.

d) [4 points] Bob uses gradient descent to optimize the weights. What difference do you expect in the final weights compared to using Lasso regularization?

Solution

a) $n - 1$. E.g. consider n colinear points with alternating labels.

b) As a weight goes towards zero the derivative of the corresponding ridge regularization term also goes to zero, whereas the derivative of the corresponding lasso regularization term is constant 1 (or -1).

OR

The weights in the ridge regularization term are squared so small weights contribute even less to the loss, whereas lasso does not square them. Therefore Lasso punishes small weights more.

c) We differentiate the regularization term (let's call it R) with respect to w_j:

$$\begin{aligned}\frac{\partial R}{\partial w_j} &= \lambda \frac{\partial}{\partial w_j}(w_j^2 + c)^{1/2} \\ &= \lambda(\frac{1}{2})(2w_j)(w_j^2 + c)^{-1/2} \\ &= \lambda w_j (w_j^2 + c)^{-1/2}\end{aligned}$$

d) The partial derivatives for the regularization term go to zero as the corresponding weights approach zero, so changes in small weights have very little impact on the total loss. Therefore weights are unlikely to end up being exactly zero. Instead we expect to end up with many small non-zero weights, unlike regular Lasso.

6 Neural Networks (14 points)

Median Network

We build a neural network consisting of 3 Layers: (1) an input layer taking in 3 integer values x_1, x_2, x_3, (2) a hidden layer with 6 nodes, and (3) an output layer with 3 nodes. We use a ReLU activation function on the hidden nodes. As in the lecture, all layers include an additional constant 1, i.e., input $\boldsymbol{x} = (1, x_1, x_2, x_3)$.

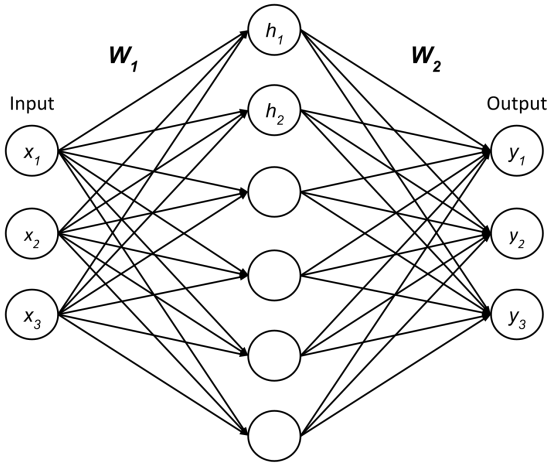


Figure 1: Illustration of the Neural network.

The output of the network should indicate which of the inputs x_1, x_2, x_3 is between the other two, i.e., which input is the median. All weights in the network can be summarized with the two weight matrices $\boldsymbol{W_1}$ (size 6×4) and $\boldsymbol{W_2}$ (3×7). As in the lecture, the first column accesses the additional hidden offset 1. We already know some of the weights from a perfectly trained network:

$$\boldsymbol{W_1} = \begin{pmatrix} 0 & 1 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & \square & \square & \square \\ 0 & \square & \square & \square \\ 0 & \square & \square & \square \\ 0 & \square & \square & \square \end{pmatrix} \qquad \boldsymbol{W_2} = \begin{pmatrix} \square & \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square & \square \end{pmatrix}$$

a) `[11 points]` Fill in the missing values in $\boldsymbol{W_1}$ and $\boldsymbol{W_2}$, such that the output indicates the index of the median value. You must name your output activation function, and explain how your output encodes the median index. While it is possible that the output is perfectly one-hot encoding the median index, you also get many points if you have a different solution.

b) `[3 points]` Why is it usually not a good idea to have ReLU as output activation function for a classification problem?

- ○ ReLUs cannot output negative values.
- ○ ReLUs can lead to exploding gradients.
- ○ ReLUs have a gradient of 0 on negative inputs.
- ○ ReLUs are continuous for positive inputs.

a) To fill the weights in $\boldsymbol{W_1}$ we can rely on symmetry:

$$\boldsymbol{W_1} = \begin{pmatrix} 0 & 1 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{pmatrix}$$

For $\boldsymbol{W_2}$ we can get an exact one-hot encoding of the median position by using a ReLU on the output layer and the following weight matrix:

$$\boldsymbol{W_2} = \begin{pmatrix} 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & -1 & -1 & -1 & -1 \end{pmatrix}$$

This solution relies on the fact that $|b-c| - |a-b| - |a-c| = 0$ if a is the median and $|b-c| - |a-b| - |a-c| < 0$ if a is not the median. Since the input only consists of integer values we can further see that $|b-c| - |a-b| - |a-c| <= -1$ if a is not the median. Therefore, adding a bias/w_0 term of 1 will result in an output of 1 at the location of the median and a 0 at the location of the other inputs, as everything ≤ 0 will be mapped to 0 by the ReLU.

Alternative solution: An alternative that gives an approximation to the one-hot vector on the output is to rely on the fact that one can get the highest activation at the location of the median. With this, we can use a softmax to approximate the one-hot vector. For this solution, any $\boldsymbol{W_2}$ of the form

$$\boldsymbol{W_2} = \begin{pmatrix} w_0 & w_1 & w_1 & w_1 & w_1 & w_2 & w_2 \\ w_0 & w_1 & w_1 & w_2 & w_2 & w_1 & w_1 \\ w_0 & w_2 & w_2 & w_1 & w_1 & w_1 & w_1 \end{pmatrix}$$

that fulfills the constraints $w_1 < 0$ and $0 < w_2 < -w_1$ is fine (no constraint on w_0).

b) The correct answer is "ReLUs have a gradient of 0 on negative inputs" as dead neurons lead to a loss of training signal which can lead the network to end up in a bad local optima.

7 Computability (18 points)

Multiple Choice

a) [3 points] You want to create a set of tiles S such that a tiling of the plane is not possible with S. What is the smallest number of symbols (e.g., letters or numbers) that you need to use for your tiles?

- ○ 1 symbol.
- ○ 2 symbols.
- ○ 3 symbols.
- ○ 4 symbols.

b) [3 points] Consider the following variant of the PCP problem: in Bottom-Heavy PCP, we know that on each domino, the word on the top side is at least one character shorter than the word on the bottom side. How hard is the Bottom-Heavy PCP problem?

- ○ Solvable in polynomial time.
- ○ Decidable but NP-hard.
- ○ Undecidable.
- ○ None of the above.

Turing Machines

c) [7 points] Consider the Turing Machine (TM) described by the transition tables below. We interpret the tape of the TM as an integer $x \geq 1$ in binary format, with the least significant bit (LSB) first, starting at cell 0. The machine consists of two states s_0 and s_1, plus a halting state s_h.

Starting state: s_0

Transitions from state s_0

Read		Write	Pointer	Next state
0	$\longrightarrow$	1	right	s_1
1	$\longrightarrow$	1	right	s_0
$\perp$	$\longrightarrow$	1	stay	s_h

Transitions from state s_1

Read		Write	Pointer	Next state
0	$\longrightarrow$	0	right	s_1
1	$\longrightarrow$	0	right	s_0
$\perp$	$\longrightarrow$	0	stay	s_h

What function does this TM compute?

$f(x) =$ [ ]

d) [5 points] Now consider a different setting: we want to compute the function $f(x) = 4x$, but now the input and the output is in a different representation, starting with the most significant bit (MSB) at cell 0.

We create a new TM for this task, again with two states s_0 and s_1, plus a halting state s_h. Fill out the missing values in the transition tables.

Hint: We have crossed out some cells in the transition tables with an X to show that the machine can never encounter these configurations.

Starting state: s_0

Transitions from state s_0

Read		Write	Pointer	Next state
0	$\longrightarrow$	0	right	$\square$
1	$\longrightarrow$	$\square$	right	s_0
$\perp$	$\longrightarrow$	0	$\square$	s_1

Transitions from state s_1

Read		Write	Pointer	Next state
0 or 1	$\longrightarrow$	X	X	X
$\perp$	$\longrightarrow$	$\square$	stay	s_h

Solution

a) With 2 symbols, we can already build a single tile that has e.g. different symbols on the top and bottom side.

b) Solvable in polynomial time. The answer is always No, the bottom string is always longer.

c) $f(x) = 2x + 1$.

d) The correct table values are as follows:

Transitions from state s_0

Read		Write	Pointer	Next state
0	$\longrightarrow$	0	right	s_0
1	$\longrightarrow$	1	right	s_0
$\perp$	$\longrightarrow$	0	right	s_1

Transitions from state s_1

Read		Write	Pointer	Next state
0 or 1	$\longrightarrow$	X	X	X
$\perp$	$\longrightarrow$	0	stay	s_h

Replacements

Here, you can find replacement templates for the fill in tasks, in case you messed up. If you use these, please **clearly** indicate which one we should consider.

```
def optimalChange(C, v):
    n = len(C)
    table = [v for i in range(v + 1)]
    table[0] = 0
    for i in range(1, v + 1):
        for j in range(                                    ):
            if                                             :
                sub_result = table[                ]
                if                                         :
                    table[                ] =
    return table[                ]
```

f.flight_id	departure_airport	COUNT(*)

$$\boldsymbol{W_1} = \begin{pmatrix} 0 & 1 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & \square & \square & \square \\ 0 & \square & \square & \square \\ 0 & \square & \square & \square \\ 0 & \square & \square & \square \end{pmatrix} \qquad \boldsymbol{W_2} = \begin{pmatrix} \square & \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square & \square \end{pmatrix}$$

Transitions from state s_0

Read		Write	Pointer	Next state
0	$\longrightarrow$	0	right	$\square$
1	$\longrightarrow$	$\square$	right	s_0
$\perp$	$\longrightarrow$	0	$\square$	s_1

Transitions from state s_1

Read		Write	Pointer	Next state
0 or 1	$\longrightarrow$	X	X	X
$\perp$	$\longrightarrow$	$\square$	stay	s_h

This page is intentionally blank