Advanced information, computation, communication I
EPFL - Fall semester 2020-2021

# Week 6 — solutions

October 22, 2020

**Exercise 1.**

```
function f1() {
  x=0
  i=1
  while (i ≤ n) {
    x=x+1
    i=x+x
  }
  a=x
}
```

```
function f2() {
  y=0
  j=1
  while (j ≤ n) {
    y=y+1
    j=y*y
  }
  b=y
}
```

*After execution of the two program fragments* f1 *and* f2*, it is the case that*

✓ $a \approx \frac{n}{2}$, $b \approx \sqrt{n}$.

◯ $a \approx n$, $b \approx \log_2(n)$.

◯ $a \approx \frac{n}{2}$, $b \approx \log_2(n)$.

◯ $a \approx n$, $b \approx \sqrt{n}$.

For the first program fragment, $x$ acts as a variable that counts the number of times the "while" is executed, where the while terminates as soon as twice the counter is $> n$, i.e., as soon as $2x > n$. The final $x$ is therefore about $\frac{n}{2}$.

For the second program fragment the argument is identical: $y$ acts as a variable that counts the number of times the "while" is executed, where the while terminates as soon as the square of the counter $y$ is $> n$, i.e., as soon as $y^2 > n$. The final $y$ is therefore about $\sqrt{n}$.

**Exercise 2.**

1. *Use Bubble Sort, Selection Sort and Insertion Sort to sort the following sequence:*

```
9, 12,-43, 20, -2, 3, 7, 28, 19.
```

(a) **Bubble Sort**:
We have a sequence of length 9. The procedure passes through the entire sequence and compares each pair of consecutive numbers $(x_i, x_{i+1})$ that are swapped if $x_i > x_{i+1}$ for $0 \leq i < 9$. Hence, the resulting sequences in the end of each passing through are the following:

```
 9, -43, 12, -2, 3, 7, 20, 19, 28.
-43, 9, -2, 3, 7, 12, 19, 20, 28.
-43, -2, 3, 7, 9, 12, 19, 20, 28.
-43, -2, 3, 7, 9, 12, 19, 20, 28.
-43, -2, 3, 7, 9, 12, 19, 20, 28.
-43, -2, 3, 7, 9, 12, 19, 20, 28.
-43, -2, 3, 7, 9, 12, 19, 20, 28.
-43, -2, 3, 7, 9, 12, 19, 20, 28.
```

It is possible to modify bubble sort to keep track of the number of swaps it performs, and in this version of the algorithm we can skip the last four passes.

(b) **Selection Sort**:

We have a sequence of length 9. We start with $i = 0$, i.e. $min = 1$. Then the minimum of $a_1, \ldots, a_8$ is $a_2 = -43 =: a_{min}$. Since $a_0 > a_{min}$ we swap these two and get

```
-43, 12, 9, 20,-2, 3, 7, 28, 19.
```

Now we go to $i = 1$, i.e. $min = 2$. Then the minimum of $a_2, \ldots, a_8$ is $a_4 = -2 =: a_{min}$. Since $a_1 > a_{min}$ we swap these two and get

```
-43, -2, 9, 20, 12, 3, 7, 28, 19.
```

Now we go to $i = 2$, i.e. $min = 3$. Then the minimum of $a_3, \ldots, a_8$ is $a_5 = 3 =: a_{min}$. Since $a_2 > a_{min}$ we swap these two and get

```
-43, -2, 3, 20, 12, 9, 7, 28, 19.
```

Now we go to $i = 3$, i.e. $min = 4$. Then the minimum of $a_4, \ldots, a_8$ is $a_6 = 7 =: a_{min}$. Since $a_3 > a_{min}$ we swap these two and get

```
-43, -2, 3, 7, 12, 9, 20, 28, 19.
```

Now we go to $i = 4$, i.e. $min = 5$. Then the minimum of $a_5, \ldots, a_8$ is $a_5 = 9 =: a_{min}$. Since $a_4 > a_{min}$ we swap these two and get

```
-43, -2, 3, 7, 9, 12, 20, 28, 19.
```

Now we go to $i = 5$, i.e. $min = 6$. Then the minimum of $a_6, \ldots, a_8$ is $a_5 = 12 =: a_{min}$. Since $a_5 = a_{min}$ we do not swap and stay with

```
-43, -2, 3, 7, 9, 12, 20, 28, 19.
```

Now we go to $i = 6$, i.e. $min = 7$. Then the minimum of $a_7, \ldots, a_8$ is $a_8 = 19 =: a_{min}$. Since $a_6 > a_{min}$ we swap these two and get

```
-43, -2, 3, 7, 9, 12, 19, 28, 20.
```

Now we go to $i = 7$, i.e. $min = 8$. Since $a_7 > a_{min}$ we swap these two and get

```
-43, -2, 3, 7, 9, 12, 19, 20, 28.
```

(c) **Insertion Sort**:

We have a sequence of length 9. We start with $j = 1, i = 0$. The lowest $i$ for which $a_1 \leq a_i$ is $i = 1$ (i.e. this is where the while-loop stops). We set $m = a_1 = 12$. The for-loop is empty and we set $a_i = a_1 := m = 12$. Hence the sequence remains

```
9, 12, -43, 20, -2, 3, 7, 28, 19.
```

Now we go to $j = 2$. The lowest $i$ for which $a_2 \leq a_i$ is $i = 0$ (i.e. this is where the while-loop stops). We set $m = a_2 = -43$. The for-loop assigns $a_2 := a_1$ and $a_1 := a_0$. Then we set $a_i = a_0 := m = -43$. Hence the new sequence is

```
-43, 9, 12, 20, -2, 3, 7, 28, 19.
```

Now we go to $j = 3$. The lowest $i$ for which $a_3 \leq a_i$ is $i = 3$ (i.e. this is where the while-loop stops). We set $m = a_3 = 20$. The for-loop is empty. Then we set $a_i = a_3 := m = 20$. Hence the sequence remains

```
-43, 9, 12, 20, -2, 3, 7, 28, 19.
```

Now we go to $j = 4$. The lowest $i$ for which $a_4 \leq a_i$ is $i = 1$ (i.e. this is where the while-loop stops). We set $m = a_4 = -2$. The for-loop assigns $a_4 := a_3, a_3 := a_2$ and $a_2 := a_1$. Then we set $a_i = a_1 := m = -2$. Hence the new sequence is

```
-43, -2, 9, 12, 20, 3, 7, 28, 19.
```

Now we go to $j = 5$. The lowest $i$ for which $a_5 \leq a_i$ is $i = 2$ (i.e. this is where the while-loop stops). We set $m = a_5 = 3$. The for-loop assigns $a_5 := a_4, a_4 := a_3, a_3 := a_2$. Then we set $a_i = a_2 := m = 3$. Hence the new sequence is

```
-43, -2, 3, 9, 12, 20, 7, 28, 19.
```

Now we go to $j = 6$. The lowest $i$ for which $a_6 \leq a_i$ is $i = 3$ (i.e. this is where the while-loop stops). We set $m = a_6 = 7$. The for-loop assigns $a_6 := a_5, a_5 := a_4, a_4 := a_3$. Then we set $a_i = a_3 := m = 7$. Hence the new sequence is

```
-43, -2, 3, 7, 9, 12, 20, 28, 19.
```

Now we go to $j = 7$. The lowest $i$ for which $a_7 \leq a_i$ is $i = 7$ (i.e. this is where the while-loop stops). We set $m = a_7 = 28$. The for-loop is empty. Then we set $a_i = a_7 := m = 28$. Hence the sequence remains

```
-43, -2, 3, 7, 9, 12, 20, 28, 19.
```

Now we go to $j = 8$. The lowest $i$ for which $a_8 \leq a_i$ is $i = 6$ (i.e. this is where the while-loop stops). We set $m = a_8 = 19$. The for-loop assigns $a_8 := a_7, a_7 := a_6$. Then we set $a_i = a_6 := m = 19$. Hence the final sequence is

```
-43, -2, 3, 7, 9, 12, 19, 20, 28.
```

2. *How many comparisons are done in each of the algorithms?*

   (a) **Bubble sort**:
   The basic version of the algorithm performs $\frac{n(n-1)}{2} = 36$ comparisons. In the optimized version of the algorithm, number of comparisons can be reduced to 26.

   (b) **Selection sort**:
   The algorithm always performs $n - 1$ comparisons at the first round and does one less per round, hence $\sum_{i=1}^{9-1} i = 36$ comparisons.

   (c) **Insertion sort**:
   The algorithm needs 20 comparisons for the given list of values.

3. *How many swaps are done in each of the algorithms?*

   (a) **Bubble sort**:
   The algorithm performs a swap each time an element is bigger than its neighbor, hence, in our case, $5 + 5 + 3 = 13$ swaps.

   (b) **Selection sort**:
   The algorithm performs a swap each time a lower element is found per round, hence, in our case, 7 swaps.

   (c) **Insertion sort**:
   The algorithm does not swap but inserts the lowest element at the end of the sorted sublist, which requires, in our case, $3 + 4 + 4 + 4 + 3 = 18$ insertions.

4. *What is the approximate overall cost of the two algorithms for an input sequence of length $n + 1$?*

| Algorithm | Comp. | Swaps |
|---|---|---|
| **Bubble sort** | $n^2$ | $n^2$ |
| **Selection sort** | $\frac{(n+2)(n+1)}{2}$ | $n$ |
| **Insertion sort** | $\frac{(n+2)(n+1)}{2}$ | $n^2$ |

**Exercise 3.** *Charlotte, Giulia and Patrick are starting university next year. They have applied to EPFL, ETHZ and USI, and their preferences are listed as follows*

| ***Student*** | *Most preferable* | $\longrightarrow$ | *least preferable* |
|---|---|---|---|
| *Charlotte* | *USI* | *ETHZ* | *EPFL* |
| *Giulia* | *EPFL* | *USI* | *ETHZ* |
| *Patrick* | *ETHZ* | *EPFL* | *USI* |

*The universities, on the other hand, have their own lists of preferred students*

| ***University*** | *Most preferable* | $\longrightarrow$ | *least preferable* |
|---|---|---|---|
| *EPFL* | *Giulia* | *Charlotte* | *Patrick* |
| *ETHZ* | *Giulia* | *Patrick* | *Charlotte* |
| *USI* | *Patrick* | *Charlotte* | *Giulia* |

*In how many ways can we match students with universities, so that there is no pair of a student and a university who both prefer each other more than the university/student they have been matched up with?*

Giulia and EPFL are each other's preferred choice. That means that they have to be paired up, otherwise the matching will not be stable. Continuing, Patrick's first choice is ETHZ, and ETHZ cannot choose Giulia because she's already taken by EPFL, so it has Patrick as it's current preferred choice. Being each others preferred choice we have to pair them up, because again, doing differently leads to an unstable matching. In the end we are left with Charlotte who will be going to USI. The final matching is given by

| **Student** | **University** |
|---|---|
| Giulia | EPFL |
| Charlotte | USI |
| Patrick | ETHZ |

This matching is indeed unique because every step of the construction was forced by the preceding steps, so there is only one stable matching.

**Exercise 4.** *The stable maximum matching problem is a generalisation of the previous exercise in the following way: we are given two equally sized sets $A$ and $B$ and to each element of the sets is assigned an ordering of preferences for the elements of the other set. A matching between two sets is a pairing between the elements of the sets, i.e., a bijective function from $A$ to $B$. A matching is called unstable if there exist two elements $a \in A$, $b \in B$ that are not paired and both of them prefer each other to the element they have been paired with. A matching that is not unstable is called stable.*

*Is a stable maximum matching unique? Either prove that every stable maximum matching is unique, or disprove it with a counterexample.*

In general, stable matchings are not unique, and we will show this by giving a counterexample. Consider the simplest case, with two students and two universities and the following preferences:

| **Student** | Most preferable | least preferable |
|---|---|---|
| Giulia | ETHZ | EPFL |
| Charlotte | EPFL | ETHZ |

| **University** | Most preferable | least preferable |
|---|---|---|
| EPFL | Giulia | Charlotte |
| ETHZ | Charlotte | Giulia |

It is not hard to check that both of the following are stable matchings:

| **Student** | **University** |
|---|---|
| Giulia | EPFL |
| Charlotte | ETHZ |

| **Student** | **University** |
|---|---|
| Giulia | ETHZ |
| Charlotte | EPFL |

In fact, in each pair in each possible pairing one of the sides is paired with its most preferable choice so the pairing cannot be unstable.

**Exercise 5.** *Let $\{A, B, C, D\}$ be a set of men, and $\{a, b, c, d\}$ a set of women. We want to match up men and women using the Gale–Shapley algorithm in two different ways. The preferences of men and women are given in the following lists, going from most preferable on the left to least preferable on the right.*

| ***Men*** | *1st* | *2nd* | *3rd* | *4th* |
|---|---|---|---|---|
| $A$ | $c$ | $d$ | $b$ | $a$ |
| $B$ | $d$ | $c$ | $a$ | $b$ |
| $C$ | $a$ | $c$ | $b$ | $d$ |
| $D$ | $b$ | $d$ | $a$ | $c$ |

| ***Women*** | *1st* | *2nd* | *3rd* | *4th* |
|---|---|---|---|---|
| $a$ | $D$ | $A$ | $B$ | $C$ |
| $b$ | $C$ | $B$ | $A$ | $D$ |
| $c$ | $C$ | $B$ | $A$ | $D$ |
| $d$ | $D$ | $A$ | $B$ | $C$ |

1. *If the men propose, and women accept/reject, what is the matching after the algorithm terminates?*

   After all men propose to their 1st prefered woman, since they are all different, they accept and the algorithm terminates with the following matching:

   A-c, B-d, C-a, D-b.

2. *If the women propose, and men accept/reject, what is the matching after the algorithm terminates?*

   Firstly a proposes to D, and b proposes to C, and they accept because it is the first proposal.

   a-D, b-C; c,d,A,B unmatched

   Then c proposes to C. Since C prefers c more than b, b gets rejected, and C accepts c instead.

   a-D, c-C; b,d,A,B unmatched

   Now d proposes to D, and again because of preferences of D, a gets rejected and D accepts d instead.

   c-C, d-D; a,b,A,B unmatched

   a and b are now unmatched, so they proposed to their second preference, A and B respectively. Since they are free, A and B accept and the algorithm terminates with:

   a-A, b-B, c-C, d-D

3. *Who is the best possible (stable) valid partner for "a"?*

   First we will show that a matching where a is paired with D is unstable. This is because D prefers d to a, and d will prefer D whoever she is matched with (D is her 1st preference). So a cannot be matched with D. On the other side a can be matched with A, as we have seen from 2. So the best matching for a is A.

   One can also simply argue that the solution found in 2 above is Women-optimal, so no woman can do better in any stable matching than found under 2, so A is the best partner than a can find.

**Exercise 6.**

$$
\begin{aligned}
L_{x_1} &= (y_3, y_1, y_2) & L_{y_1} &= (x_2, x_1, x_3)\\
L_{x_2} &= (y_2, y_3, y_1) & L_{y_2} &= (x_1, x_3, x_2)\\
L_{x_3} &= (y_1, y_2, y_3) & L_{y_3} &= (x_3, x_2, x_1)
\end{aligned}
$$

(*français*) *Soit $L_x$ pour $x \in X = \{x_1, x_2, x_3\}$ la liste de préférence de $x$ donnée ci-dessus et soit $L_y$ pour $y \in Y = \{y_1, y_2, y_3\}$ la liste de préférence de $y$ donnée ci-dessus. Le couplage $\{(x_1, y_1), (x_2, y_3), (x_3, y_2)\}$ est*

(*English*) *Let $L_x$ for $x \in X = \{x_1, x_2, x_3\}$ be the preference list of $x$ as given above and let $L_y$ for $y \in Y = \{y_1, y_2, y_3\}$ be the preference list of $y$ as given above. The matching $\{(x_1, y_1),\ (x_2, y_3),\ (x_3, y_2)\}$ is*

○ $\begin{cases} \textit{instable.} \\ \textit{unstable.} \end{cases}$

○ $\begin{cases} \textit{stable et optimal pour } Y. \\ \textit{stable and } Y\textit{-optimal.} \end{cases}$

○ $\begin{cases} \textit{stable et optimal pour } X. \\ \textit{stable and } X\textit{-optimal.} \end{cases}$

✓ $\begin{cases} \textit{stable, mais n'est pas un couplage stable optimal pour } X \textit{ ou pour } Y. \\ \textit{stable but not a stable matching that is } X\textit{- or } Y\textit{-optimal.} \end{cases}$

In the following matching: $\{(x_1, y_3),\ (x_2, y_2),\ (x_3, y_1)\}$, all $x \in X$ are matched with their optimal choices and therefore the matching $\{(x_1, y_3),\ (x_2, y_2),\ (x_3, y_1)\}$ is stable and $X$-optimal. Because this matching $\{(x_1, y_3),\ (x_2, y_2),\ (x_3, y_1)\}$ is different from the given matching $\{(x_1, y_1),\ (x_2, y_3),\ (x_3, y_2)\}$, the matching $\{(x_1, y_1),\ (x_2, y_3),\ (x_3, y_2)\}$ is not $X$-optimal, so that the third answer is not correct. (Note that in the matching $\{(x_1, y_3),\ (x_2, y_2),\ (x_3, y_1)\}$ all $y \in Y$ are matched to their least favorite choices.)

In the following matching: $\{(x_1, y_2),\ (x_2, y_1),\ (x_3, y_3)\}$, all $y \in Y$ are matched with their optimal choices and therefore the matching $\{(x_1, y_2),\ (x_2, y_1),\ (x_3, y_3)\}$ is stable and $Y$-optimal. Again, because this matching $\{(x_1, y_2),\ (x_2, y_1),\ (x_3, y_3)\}$ is different from the given matching $\{(x_1, y_1),\ (x_2, y_3),\ (x_3, y_2)\}$, the matching $\{(x_1, y_1),\ (x_2, y_3),\ (x_3, y_2)\}$ is not $Y$-optimal, so that the second answer is not correct. (Note that in the matching $\{(x_1, y_2),\ (x_2, y_1),\ (x_3, y_3)\}$ all $x \in X$ are matched to their least favorite choices.)

Actually, in the matching $\{(x_1, y_1),\ (x_2, y_3),\ (x_3, y_2)\}$, no $x \in X$ or $y \in Y$ is matched with its optimal or with its worst choice. So, in principle, there could be a quite a lot of room to move things around:

**in** $(x_1, y_1)$**:**

$x_1$ would prefer to be matched with $y_3$, but $y_3$ prefers its current $x_2$ to $x_1$. Because there is no other $y \in Y$ that $x_1$ would prefer to $y_1$, the pair $(x_1, y_1)$ is stable from $x_1$'s point of view.

$y_1$ would prefer to be matched with $x_2$, but $x_2$ prefers its current $y_3$ to $y_1$. Because there is no other $x \in X$ that $y_1$ would prefer to $x_1$, the pair $(x_1, y_1)$ is stable from $y_1$'s point of view as well.

It follows that the pair $(x_1, y_1)$ is stable.

**in** $(x_2, y_3)$**:**

$x_2$ would prefer to be matched with $y_2$, but $y_2$ prefers its current $x_3$ to $x_2$. Because there is no other $y \in Y$ that $x_2$ would prefer to $y_3$, the pair $(x_2, y_3)$ is stable from $x_2$'s point of view.

$y_3$ would prefer to be matched with $x_3$, but $x_3$ prefers its current $y_2$ to $y_3$. Because there is no other $x \in X$ that $y_3$ would prefer to $x_2$, the pair $(x_2, y_3)$ is stable from $y_3$'s point of view as well.

It follows that the pair $(x_2, y_3)$ is stable.

**in** $(x_3, y_2)$**:** a similar argumentation can be used again, or we can simply argue that the pair $(x_3, y_2)$ must be stable because its instability would imply instability of another pair which is impossible because all other pairs are stable.

It follows that only the fourth answer is correct.

**Exercise 7.**
**Greedy algorithm**: The total amount n is first compared with the largest denomination. As long as the total amount n is more than (or equal to) the largest denomination, then one such coin is added and the

total amount is decreased by the denomination. Then we repeat it for the second largest denomination, third largest denomination and so on (until we checked all denominations).
Quarter: 25 cent
Dime: 10 cent
Nickel: 5 cent
Penny: 1 cent

**Solution**:
We apply the cashier algorithm in full detail only for the first case as it's pretty similar for other values of $n$.

1. Given total amount: n = 87
   (a) First pass (quarters) $c_1 = 25$
   We note that $n \geq 25$, thus we add 1 quarter and decrease the total amount by 25.
   $$d_1 = 1$$
   $$n = n_{old} - 25 = 87 - 25 = 62$$
   We note that $n \geq 25$, thus we add 1 quarter and decrease the total amount by 25.
   $$d_1 = 1 + 1 = 2$$
   $$n = n_{old} - 25 = 62 - 25 = 37$$
   We note that $n \geq 25$, thus we add 1 quarter and decrease the total amount by 25.
   $$d_1 = 2 + 1 = 3$$
   $$n = n_{old} - 25 = 37 - 25 = 12$$
   We note n $<$ 25 thus the first pass ends.
   (b) Second pass (dimes) $c_2 = 10$
   We note that $n \geq 10$, thus we add 1 quarter and decrease the total amount by 10.
   $$d_1 = 3$$
   $$d_2 = 1$$
   $$n = n_{old} - 10 = 12 - 10 = 2$$
   We note n $<$ 10 thus the second pass ends.
   (c) Third pass (dimes) $c_3 = 5$
   We note n $<$ 5 thus the third pass ends.
   (d) Fourth pass (pennies) $c_4 = 1$
   We note that $n \geq 1$, thus we add 1 quarter and decrease the total amount by 1.
   $$d_1 = 3$$
   $$d_2 = 1$$
   $$d_4 = 1$$
   $$n = n_{old} - 1 = 2 - 1 = 1$$
   We note that $n \geq 1$, thus we add 1 quarter and decrease the total amount by 1.
   $$d_1 = 3$$

$$d_2 = 1$$

$$d_4 = 2$$

$$n = n_{old} - 1 = 1 - 1 = 0$$

We note n < 1 thus the fourth pass ends.
**Conclusion:** 87 cents is 3 quarters, 1 dime, 0 nickels and 2 pennies.

2. (similarly) 49 cents is 1 quarter, 2 dimes, 0 nickels and 4 pennies.
3. (similarly) 99 cents is 3 quarters, 2 dimes, 0 nickels and 4 pennies.
4. (similarly) 33 cents is 1 quarter, 0 dimes, 1 nickel and 3 pennies.

**Exercise 8.** The only thing that we need to show, is that given a function $f$ and a finite list of input values $a_1, a_2, ..., a_n$ (the domain), all the respective output values of the function (i.e. $f(a_1), f(a_2), ..., f(a_n)$) are unique or not. In other word, for any distinct i and j where $1 \leq i \leq n$ and $1 \leq j \leq n$, we should have $f(a_i) \neq f(a_j)$ to have a one-to-one function.
The following algorithm tries to check if all input pairs have distinct outputs to determine if the function is one-to-one or not.

```
Truth = True
for i = 1 to n
    for j = 1 to n
        if ($a_i \neq a_j$ and $f(a_i) = f(a_j)$)
            Truth = False
return Truth
```

**Exercise 9.** We call the algorithm " improved-bubblesort" and the input is a list of real numbers $a_1, a2, ..., a_n$.
The variable "interchange" will keep track if any interchanges were made during each pass/step. i is the position of an element in the list (we will start from 2 as the first element does not have a preceding term).

$$interchange := 0$$

$$i := 2$$

For every element in the list, we wukk determine if the elements are in increasing order. If they are not, then we interchange the two elements. If an interchange is made, the the variable "interchange" should be put to 0 (and thus we also need to change the value from 0 in each iteration step).

```
interchange = 0
i = 2
while (i < n and interchange= 0)
    interchange = 1
    for j = 1 to n-i
        if $a_j < a_{j-1}$
            temp = $a_j$
            $a_j$ = $a_{j-1}$
            $a_{j-1}$ = temp
            interchange = 0
    i = i+1
return $a_1, a_2, ..., a_n$
```

**Exercise 10.** As the characters of both words should be exactly similar to each other to be anagrams, we can proceed as follows.
First we check if the length of the two strings are equal (as two strings with different length cannot be anagrams).
If they have the same length, then we sort both strings using any sorting algorithm like bubble-sort (the "value" of each character can be defined to be its position in the alphabet).
once two sorted strings are achieved, in order to be anagrams, these two sorted strings should be exactly the same, which can be easily checked by iterating through both strings and checking whether all corresponding characters are equal to each other or not.