Exercise 1.

a) The runtime of the Algorithm 1 for the shortest path problem equals $\mathcal{O}(n \cdot m)$, where n equals the number of vertices and m equals the number of edges.

Proof. The initialization of the lengths runs in $\mathcal{O}(n)$. For the worst case scenario, for every length update a new shorter path will be revealed, until every vertex is finally evaluated ($\mathcal{O}(n)$). To find the newly revealed shortest path, it takes $\mathcal{O}(m)$ comparisons. Hence, the algorithm will run in $\mathcal{O}(n \cdot m)$ in the worst case and in $\mathcal{O}(\max\{n, m\})$ in best case. □

b) $A <_{\text{asymp}} B <_{\text{asymp}} E <_{\text{asymp}} C <_{\text{asymp}} D$

Proof.

- $A <_{\text{asymp}} B$

$$2^{\sqrt{\log n}} = 2^{\log n^{\frac{1}{2}}} = 2^{\frac{1}{2}\log n} = \sqrt{2}^{\log n}$$

Also, the following holds. Let f be a monotone increasing function, $|f| > 1$ and $c > 1$. Then:

$$\lim_{\infty} \frac{c^f}{f} > 1$$
$$\Rightarrow \lim_{n\to\infty} \frac{\sqrt{2}^{\log n}}{\log n} > 1$$

and $A <_{\text{asymp}} B$

- $B <_{\text{asymp}} E$

$$2^{\sqrt{\log n}} < 2^{\sqrt{\log n \cdot \log n}} = 2^{\log n} = n < 28n$$

- $E <_{\text{asymp}} C$

$$C = \log n^{\log n} = (2^{\log\log n})^{\log n} = (2^{\log n})^{\log\log n} = n^{\log\log n}$$

Therefore, every linear function out of $\mathcal{O}(n)$ will be outgrown by C as soon as $\log\log n > 1$.

- $C <_{\text{asymp}} D$

$$C = \log n^{\log n} = (2^{\log\log n})^{\log n} = (2^{\log n})^{\log\log n} = n^{\log\log n}$$
$$D = (\log\log n)^n = (2^{\log\log\log n})^n = (2^n)^{\log\log\log n}$$

It would suffice to proof that the following holds:

$$\exists n_0 \forall n \geq n_0, c > 1 : 2^{c\cdot n} \geq n^{\log\log n}$$

□

Exercise 2.

a) *Proof.* **Invariant:** $d[v] \geq \delta(s,v)\,\forall v \in V$ at any step.

Initially $d[s] = 0 \geq \delta(s,s)$ and $d[v] = \infty \geq \delta(s,v)\,\forall v \in V$, which means that the invariant holds before the first iteration of the algorithm. In each iteration, the value d for a subset of the vertices is updated according to the following expression: $d[v] = d[u] + w(u,v)$. From this, we can show the following:

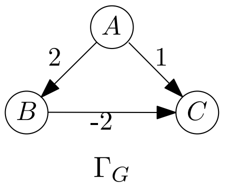
$$d[v] = d[u] + w(u,v) \geq \delta(s,u) + w(u,v) \geq \delta(s,u) + \delta(u,v) \geq \delta(s,v)$$

Therefore, we know that the invariant holds after each step in the algorithm. Since $w : E \to \mathbb{R}^+$, the value d does decrease monotonically whilst the algorithm is running. This means that if $d[v] = \delta(s,v)$ is reached, the value of $d[v]$ does never change again.

Since the value of $d[v]$ cannot change anymore once v is removed from the priority queue, we need to show that when v is removed from PQ, $d[v] = \delta(s,v)$. We use the inductive hypothesis that $d[u] = \delta(s,u)$ for any vertex u that has been removed from the priority queue. As the induction basis we have s being removed as the first element from PQ with $d[s] = 0 = \delta(s,s)$. When any vertex v is removed from the priority queue, it's value $d[v]$ is smaller than d of any of the other vertices in the PQ. If we call the last vertex before v in the shortest path from s to v x, we know that $\delta(s,x) < \delta(s,u)$ because all weights in the graph are positive. This tells us that, x must be a vertex already removed from PQ. During the step of removing x from PQ, $d[v]$ was updated to $d[x] + w(x,v)$ and the induction hypothesis states that when removing x, $d[x] = \delta(s,x)$. Therefore $d[v]$ was updated to $d[x]+w(x,v) = \delta(s,x) + w(x,v) = \delta(s,v)$. □

[1]

b) Consider the drawing of the following graph:



Γ_G

Here, the PQ will look as follows:

```
PQ = [(A,0),(C,∞),(B,∞)]
```

`Extract_Min` returns and deletes `(A,0)`. Its neighbours will get updated to PQ':

```
PQ' = [(C,1),(B,2)]
```

Next, C will be extracted since it is the minimum. But no edge from C is left out of PQ'. The result will be:

$$d[A] = 0$$
$$d[B] = 2$$
$$d[C] = 1 \neq 0$$

Therefore not the correct answer.

[1]Loosly following `https://web.engr.oregonstate.edu/~glencora/wiki/uploads/dijkstra-proof.pdf`

Exercise 3.

a)
- `Insert`: Worst case when doubling of array is needed
  1. Allocate memory, assume $\mathcal{O}(1)$, e.g. TLSF[2]
  2. Copy entries $\mathcal{O}(n)$
  3. Insert Element $\mathcal{O}(1)$

  $\Rightarrow \mathcal{O}(n)$
- `Delete`: Worst case when halving of the array is needed
  1. Remove Element $\mathcal{O}(1)$
  2. Allocate memory $\mathcal{O}(1)$
  3. Copy elements $\mathcal{O}(n)$

  $\Rightarrow \mathcal{O}(n)$

b)
- `Insert`

  Proof. Pay three coins per operation. The first coin pays for inserting the element, the remaining two coins get stored in the account. If the array needs to be doubled, at least $\frac{n}{2}$ insertions have happened since the last doubling. This means that at least n coins are in the account that can be used to pay for the doubling of the array. Therefore, the amortized cost of an `Insert` operation is $\mathcal{O}(1)$. □

- `Delete`

  Proof. Pay two coins per operation. The first coin pays for deleting the element, the remaining coin gets stored in the account. If the array needs to be halved, at least n deletions have happened since the last halving. This means that at least n coins are in the account that can be used to pay for the halving of the array. Therefore, the amortized cost of a `Delete` operation is $\mathcal{O}(1)$. □

c) *Proof.* Let m be the current size of the array. The we choose the following potential function Φ:

$$\Phi = \left| n - \left\lfloor \frac{m}{2} \right\rfloor \right|$$

We consider the suboperations `Simple_Insert`, `Double` that make up a `Insert` operation and `Simple_Delete`, `Halve` that make up a `Delete` operation.

- `Simple_Insert`

$$\hat{c}_{\texttt{SI}} = c_{\texttt{SI}} + \Phi(A) - \Phi(A') = \mathcal{O}(1) \begin{cases} +1 & \text{for } n \leq \left\lfloor \frac{m}{2} \right\rfloor \\ -1 & \text{for } n > \left\lfloor \frac{m}{2} \right\rfloor \end{cases} = \mathcal{O}(1)$$

- `Simple_Delete`

$$\hat{c}_{\texttt{SD}} = c_{\texttt{SD}} + \Phi(A) - \Phi(A') = \mathcal{O}(1) \begin{cases} +1 & \text{for } n \geq \left\lfloor \frac{m}{2} \right\rfloor \\ -1 & \text{for } n < \left\lfloor \frac{m}{2} \right\rfloor \end{cases} = \mathcal{O}(1)$$

[2]`https://github.com/mattconte/tlsf`

- `Double` (happens when $n' = m'$)

$$\hat{c}_{\texttt{D}} = c_{\texttt{D}} + \Phi(A) - \Phi(A') = \mathcal{O}(n) + 1 - \left\lceil \frac{n}{2} \right\rceil = \mathcal{O}(1)$$

- `Halve` (happens when $n' = \frac{m'}{4}$)

$$\hat{c}_{\texttt{H}} = c_{\texttt{H}} + \Phi(A) - \Phi(A') = \mathcal{O}(n) + 1 - n = \mathcal{O}(1)$$

Where A denotes the state of the array after and A' the state of the array before the operation.
Therefore, the combined operation `Insert` and `Delete` are also in $\mathcal{O}(1)$. □

d) *Proof.* tbd □

Exercise 4.

a) The height of a Fibonacci heap is in $\mathcal{O}(n)$ in the worst case.

Proof. Consider a chain of increasing elements L. Then, The first element can be interpreted as a root list with one element which has one child. This child has only one child and so on. Every element points to itself in a linked list. Therefore, L can be interpreted as a valid Fibonacci Heap where the height is in $\mathcal{O}(|L|)$. □

b) We first define a generalized number sequence based on the fibonacci numbers:

$$F_n^{(k)} = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } 1 \leq n < k \\ F_{n-1}^{(k)} + F_{n-k}^{(k)} & \text{otherwise} \end{cases}, \quad k \in \mathbb{N} \setminus \{0,1\}$$

The fibbonaci numbers F_n are a special case of these number sequences with $F_n = F_n^{(2)}$. Based on these generalized number sequences we can now generalize the bounding of $d_{\max}$ as seen in the lecture.

Lemma 1. $F_{n+k}^{(k)} \geq \alpha^n \, \forall k \in \mathbb{N} \setminus \{0,1\}$, *where* $\alpha^k = \alpha^{k-1} = 1$.

Proof. Induction base $n = 0$:

$$F_{n-k}^{(k)} = F_k^{(k)} = 1 \geq \alpha^0$$

Induction hypothesis: $F_{j+k}^{(k)} \geq \alpha^j \, \forall 0 \leq j \leq n+k-1$.
Induction step $n+k-1 \to n = k$

$$F_{n+k}^{(k)} = F_{n+k-1}^{(k)} + F_n^{(k)} \geq \alpha^{n-1} + \alpha^{n-k} = \alpha^{n-k}(\alpha^{k-1} + 1) = \alpha^n$$

□

Lemma 2. *Let x be a node of degree i and let $z_1, \ldots, z_i$ be the children of x in the order they were linked to x, then*

$$\text{degree}(z_j) \geq \begin{cases} 0 & \textit{if } 1 \leq j < k \\ j - k & \textit{if } j \geq k \end{cases}$$

Proof. Since the degree of a node is always ≥ 0, the lemma is true for $1 \leq j \leq k$. When x and z_j are linked, they must have the same degree, because only trees of the same degree are linked. At that time the degree of x and therefore also of z_j is $j-1$. Since being linked to x, z_j can have lost at most $k-1$ children, otherwise it would have been cut. Therefore, $\text{degree}(z_j) \geq j - k$. □

Let $s_i^{(k)}$ be the minimum size of a subtree rooted at a vertex of degree i fulfilling lemma 2, then

$$s_i^{(k)} = 1 + \min(i, k-1) + \sum_{j=0}^{i-k} s_j = \begin{cases} 1 + i & \text{if } 0 \leq i < k \\ s_{i-1} + s_{i-k} & \text{if } i \geq k \end{cases} = F_{i+k}^{(k)}$$

Lemma 3. *Let $H^{(k)}$ be a Fibonacci heap with n nodes where nodes are marked when $k-1$ children were removed from the node. Let $d_{max}(H^{(k)})$ be the maximum degree of any node in $H^{(k)}$. Then, $d_{max}(H) \in \mathcal{O}(\log n)$.*

Proof. For any node x in $H^{(k)}$ with degree i it holds that $\text{size}(x) \geq \alpha^i$ where $\alpha^k = \alpha^{k-1} + 1$:

$$\text{size}(x) \geq s_i^{(k)} = F_{i+k}^{(k)} \geq \alpha^i$$

Since $\text{size}(x) \leq n$, we know that $i \leq \log_\alpha n$ and since we defined x to be an arbitrary node of $H^{(k)}$

$$d_{\max} \leq \log_\alpha n = \mathcal{O}(\log n)$$

□