Explain your reasoning and show calculations/steps.

If you have any questions, visit the Algkon forum (https://algkon.idi.ntnu.no/forum/) or contact the responsible TA by email.

Task 1

We fulfill exactly three clauses for all of the four possible assignments. Ergo, you cannot fulfill all clauses, and any assignment is optimal.

Task 2

The idea is the same as for the case of the simple bipartite cut. For each vertex i, we let it be random which partition it ends up in, with uniform probability.

$$\mathrm{P}(i \in \mathrm{V}_l) = \frac{1}{k} \quad 1 \leqslant l \leqslant k \tag{1}$$

The probability that an edge (i, j) is included in the cut is then the same as the probability that the endpoints are in different partitions.

$$\mathrm{P}(i \in \mathrm{V}_l, j \in \mathrm{V}_m, l \neq m) = 1 - \mathrm{P}(i \in \mathrm{V}_l, j \in \mathrm{V}_m, l = m) = 1 - k \cdot \frac{1}{k} \cdot \frac{1}{k} = \frac{k-1}{k} \tag{2}$$

We then have an expected total weight of

$$\sum_{(i,j)\in \mathrm{E}} \frac{k-1}{k} w_{ij} \geqslant \frac{k-1}{k} \mathrm{OPT}, \tag{3}$$

as even an optimal solution cannot expect to include more than all the edges.

Task 3

Again, we can try the simple idea of letting it be random which partition each vertex ends up in. We let there be an equal probability $= \frac{1}{2}$ that a vertex ends up in U or W. The probability that an arc is included in our cut is then equal to the probability that the start point is in U and the end point is in W.

For an arc (i, j) we then have the following

$$\mathrm{P}(i \in \mathrm{U} \cap j \in \mathrm{W}) = \mathrm{P}(i \in \mathrm{U}) \cdot \mathrm{P}(j \in \mathrm{W}) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}. \tag{4}$$

The rest of the proof follows the same pattern as in task 2.

Task 4

The idea is simple and follows the technique in the syllabus, we assign vertices one by one and keep the expected value up. We start with an arbitrary vertex and put it in one of the partitions.

This will not change the expected value, it must be in one of the partitions after all. We then move on to another vertex, and calculate the expected value of the cut conditional on the vertex being placed in a given partition. We then place the vertex in the partition that gives the highest conditional expected value.

This is based on the fact that if we have an expected value of a random event, then there must be at least one outcome that is worth as much or more than that value. We will therefore always manage to keep to at least as high a value as the random algorithm.

Task 5

One possibility for an algorithm with a strict performance guarantee of 2 is to buy a one-time ticket the first $\lfloor p_{365}/p_1 \rfloor$ times, and then buy an annual ticket the next time. To show that this algorithm has a strict 2 performance guarantee, we need to show that we never spend more money than an optimal solution. Since we never buy single tickets for more than the same price as an annual pass, we can never end up spending more money than $2p_{365}$. Thus, we are within $2 \cdot \text{OPT}$ if it pays to buy an annual pass. If it is not worthwhile to buy an annual pass, the price of single tickets for all visits will not reach p_{365} and we therefore spend exactly the same amount of money as an optimal solution. Thus, more money is never used than twice what an optimal solution would have done.

Task 6

Let SingleReplacePaging be the replacement algorithm and let ArbitraryReplacePaging be the original algorithm (the one that can replace more than one page in each iteration). Let C be the original cache, C′ be the cache controlled by ArbitraryReplacePaging and I the input. If we leave SingleReplacePaging as below, overall it will not replace more pages than ArbitraryReplacePaging would.

```
SingleReplacePaging(C, I)
1  C′ = C
2  for i = 1 to n
3       C′ = ArbitraryReplacePaging(I_i, C′)
4       if I_i ∉ C
5            place I_i in C on the same space as I_i is in C′
```

To see that this is true, note that all replacements of pages occur on line 5. These replacements place I_i in C, in the same place that I_i is in C′. So, it updates C based on what C′ would look like. That is, if it updates a space in C , then the space must have been updated in C′ since the last time it was updated in C . Thus, no more updates can occur to pages in C than in C′.

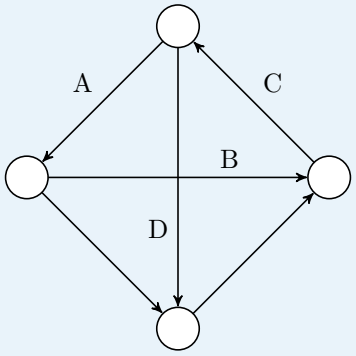
Task 7

Local is not competitive. We can see this by letting the cache have a size of k and let there be a total of $k + 1$ different pages in memory. Let the cache start by containing $(p_1, p_2, \ldots, p_k)$. If the requested pages are $(p_{k+1}, p_1, p_{k+1}, p_1, \ldots)$ then in each iteration we will have to replace a page in the cache, since in each iteration either p_1 or p_{k+1} is thrown out. An optimal solution contains only one replacement. Thus, the algorithm cannot be competitive.

Task 8

FWF works by fetching pages into the cache without removing other pages as long as there is space. If there is no more space in the cache, on the next fetch it removes all the pages that are in the cache. This is exactly how a *marking algorithm* works, as we do not replace any marked pages (pages that have been requested since the last time all pages were marked) until all pages are marked.

Task 9



In the tournament $G = (V, E)$, above, $F = \{A, D\}$ is a feedback arc set. (If we remove only A or D, then we still have at least one cycle in the graph). The graph $G = (V, E \setminus F)$ has no cycles, but $G \circledast F$ has one cycle. As the theorem in the syllabus mentions, there must be a minimal feedback arc set that has the property that $G \circledast F$ is acyclic, and $F = \{C\}$ is one possible example for G.

The point here is that the implication only goes one way. If $G \circledast F$ is acyclic then F is a feedback arc set, but an arbitrary feedback arc set F does not necessarily have the property that $G \circledast F$ is acyclic.

Task 10

We start by giving a limit on the number of unique lines we have to consider. Each line intersects any number of points, but two of them must be outermost. The number of combinations $\binom{n}{2} = O(n^2)$. We can thus evaluate all the lines in polynomial time.
We use the following reduction rules:

1. If a line intersects $k + 1$ points, we include it and reduce k by one.

   If this line is not included, we will have to include $k + 1$ other lines to cover the points. This follows from the fact that this line is the only one that can cover more than one of these points at the same time.

2. If no lines intersect $\frac{n}{k}$ points, we return that we have a no instance.

   If all lines intersect $\frac{n}{k} - x$ points, $x \geqslant 1$, we have $k \cdot (\frac{n}{k} - x) = n - kx < n$

When we can no longer use these rules, we know that each line intersects at most k points (if not we could use rule 1). Since the number of possible lines is $O(n^2)$, the number of points must also be $O(k^2)$.