Names missing!

14/20

Ex. 1

a)

2/2

Suppose we have jobs of time $\{5, 4, 3, 3, 3\}$ and two machines. Then the modified algorithm will assign the jobs to the two machines resulting in the following job distribution:

$$m_1 : [5, 3]$$

$$m_2 : [4, 3, 3]$$

This results in a makespan of 10, while the optimal job distribution is:

$$m_1 : [5, 4]$$

$$m_2 : [3, 3, 3]$$

which results in a makespan of 9. ✓

b)

1/3

That's not any schedule!

We start by distributing the longest m jobs to the machines. Since we sorted the jobs, each machine has a load of a least $t(m+1)$, so after assigning the $m+1$ job to any machine results in a makespan $\geq 2 \cdot t(m+1)$. ✓

c)

3/3

From b) we have that for any schedule f:

$$\frac{1}{2} \cdot makespan(f) \not\leq t(m+1)$$ ✓

From the lecture we also know that for any schedule f:

$$makespan(f) \geq \frac{T}{m}$$ ✓

We can adapt the proof from Lemma 2 for our purposes. Let j_{max} be the last job that gets assigned to the machine that has the highest load under the output schedule f_A from our algorithm. Let $T = \sum_i t(i)$. We then know that:

$$makespan(f_A) \leq t(j_{max}) + \frac{T}{m}$$ ✓

If $j_{max} \geq m+1$ we get:

$$makespan(f_A) \leq t(j_{max}) + \frac{T}{m} \leq \frac{1}{2} makespan(f_{optim}) + makespan(f_{optim}) = \frac{3}{2} makespan(f_{optim})$$ ✓

If $j_{max} < m+1$ we have that the machine with the highest load has only one job assigned, in which case our schedule is optimal since the job has to be assigned to a machine.

✓

Ex. 2

2/4

1. T_3: because it is the largest set ✓
   - $T_l = \{2, 3, 6, 9\}$
2. T_1: the largest set with the smallest index ✗
   - $T_l = \{1, 2, 3, 6, 9\}$
3. T_2: the largest set with the smallest index ✗
   - $T_l = \{1, 2, 3, 4, 6, 9\}$
4. T_4: the largest set with the smallest index (✓)
   - $T_l = \{1, 2, 3, 4, 6, 7, 8, 9\}$
5. T_7: the largest set with the smallest index (✓)
   - $T_l = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

You didn't update the sets after picking T_3 (and after T_1)

Ex. 3

a)

4/4

Input: unweighted graph $G = (V, E)$ Output: matching M

- $M = \emptyset$
- $N = \emptyset$
- for edge $e = (v, w)$ in E do: ✓
  - if $v \notin N$ and $w \notin N$ do:
    - $M = M \cup \{e\}$
    - $N = N \cup \{v, w\}$
- return M

The algorithm iterates over all edges. It checks whether the endpoints of the current edge were already encountered. If so, we discard the edge. Else we add it to the matching set M and the endpoints to the set of already seen nodes N. After iterating over all nodes, we return the matching set.

b)

2/2

We iterate over all edges so we get $\mathcal{O}(|E|)$. ✓ For each edge we have to check whether v, w already are in N which we can do in constant time $\mathcal{O}(1)$ with a hash map. In total this gives us a runtime complexity of $\mathcal{O}(|E|)$. ✓

ok

Initializing hash-map in $\Theta(|E|)$

c)

0/2

By definition, our algorithm returns a so-called maximal matching, i.e. a matching that is not a subset of any other matching. ✓ Let A and B be two maximal matchings. We then have $|A| \leq 2|B|$. ##### **Proof:** Each edge in $B \backslash A$ can be adjacent to at most 2 edges in $A \backslash B$, else A would not be a valid matching. ✓ Further, each edge in $A \backslash B$ is adjacent to an edge in $B \backslash A$. ✓ If there was an edge in $B \backslash A$ that did not fulfill this, A would not be maximal since we could add the

edge from $B \backslash A$ get a superset of A. Combining this, we get

$$|A \backslash B| \leq 2|B \backslash A| \quad \checkmark$$

That's not yet proof for this exercise!

$\frac{1}{2}$-approx. not finished,
valid matching proof missing