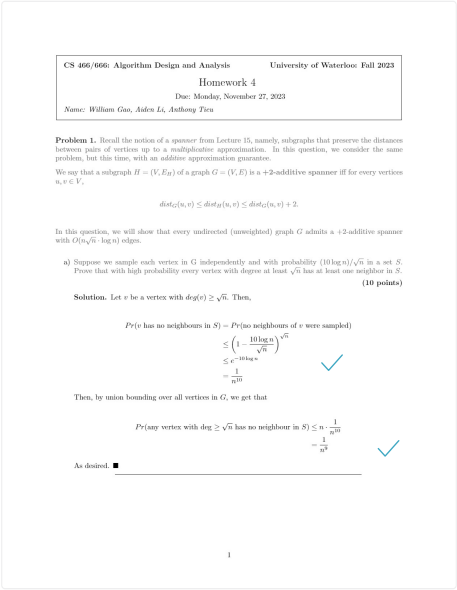
My grades for Homework 4

Q1 25 / 25

Refer to the assignment PDF.



CS 466/666: Algorithm Design and Analysis University of Waterloo: Fall 2023

Homework 4

Due: Monday, November 27, 2023

Name: William Gao, Aiden Li, Anthony Tieu

Problem 1. Recall the notion of a *spanner* from Lecture 15, namely, subgraphs that preserve the distances between pairs of vertices up to a *multiplicative* approximation. In this question, we consider the same problem, but this time, with an additive approximation guarantee.

We say that a subgraph $H = (V, E_H)$ of a graph $G = (V, E)$ is a $+2$-additive spanner iff for every vertices $u, v \in V$,

$$dist_G(u, v) \leq dist_H(u, v) \leq dist_G(u, v) + 2.$$

In this question, we will show that every undirected (unweighted) graph G admits a $+2$-additive spanner with $O(n\sqrt{n} \cdot \log n)$ edges.

a) Suppose we sample each vertex in G independently and with probability $(10 \log n)/\sqrt{n}$ in a set S. Prove that with high probability every vertex with degree at least $\sqrt{n}$ has at least one neighbor in S. **(10 points)**

Solution. Let v be a vertex with $deg(v) \geq \sqrt{n}$. Then,

$$\begin{aligned} Pr(v \text{ has no neighbours in } S) &= Pr(\text{no neighbours of } v \text{ were sampled}) \\ &\leq \left(1 - \frac{10 \log n}{\sqrt{n}}\right)^{\sqrt{n}} \\ &\leq e^{-10 \log n} \\ &= \frac{1}{n^{10}} \end{aligned}$$

Then, by union bounding over all vertices in G, we get that

$$\begin{aligned} Pr(\text{any vertex with deg} \geq \sqrt{n} \text{ has no neighbour in } S) &\leq n \cdot \frac{1}{n^{10}} \\ &= \frac{1}{n^9} \end{aligned}$$

As desired. ■

1

b) Let S be a set as chosen in part (a). Let H be a subgraph of G that contains a BFS tree from every vertex in S, plus the set of all edges on vertices with degree $< \sqrt{n}$. Prove that H is a 3-spanner of G with $O(n\sqrt{n}\log n)$ edges with high probability.

(15 points)

Solution. Let S be chosen as in a) and let H be the subgraph that contains a BFS tree from every vertex in S plus the set of all edges on vertices with degree $< \sqrt{n}$.

Take two vertices $u, v \in V$. Let P be the shortest path from u to v in G. We show there exists a path P' from u to v in H such that $|P| \leq 2 + |P'|$.

Case 1: All edges in P are in H. Then we just pick $P' = P$ and we are done with $|P'| = |P|$.

Case 2: Not all edges in P are in H. Consider following P until the next edge ab is not in H. For notation, we partition P into the path from u to a, P_{ua} and the path from b to v, P_{bv}. So P consists of the path P_{ua}, the edge ab, then the path P_{bv}. Thus, $|P| = |P_{ua}| + 1 + |P_{bv}|$.

Since ab is not an edge in H, $\deg(a) \geq \sqrt{n}$. Then, from part a), we know that a has a neighbour in S, call it c.

Since $c \in S$, we have a shortest path from c to v, P'_{cv}, using the BFS tree from c. Furthermore, since the path (c, a, P_{bv}) exists in G, $|P'_{cv}| \leq 2 + |P_{bv}|$. So the path from u to a, with the edge ac, together with the path P'_{cv} yields a path of length $|P_{ua}| + 1 + |P'_{cv}| \leq |P_{ua}| + |P_{bv}| + 3 = |P| + 2$.

Now, with high probability (Chernoff), we sample $O\left(\frac{\ln n \log n}{\sqrt{n}} \cdot n\right)$ vertices in S. Since each one contributes a BFS tree, we have $O\left(\frac{\ln n \log n}{\sqrt{n}} \cdot n^2\right) = O(n\sqrt{n}\log n)$ edges from the BFS trees. Furthermore, there are at most $n \cdot \sqrt{n}$ edges contributed by the vertices with degree $< \sqrt{n}$. So the total number of edges in the spanner is $O(n\sqrt{n}\log n)$. ■

with high prob-abil-ity

well done!

Q2 25 / 25

Refer to the assignment PDF.

need to briefly justify that a field of approximately such sizer exists.

Problem 2. Suppose we have two players Alice and Bob, who have strings $x \in \{0,1\}^n$ and $y \in \{0,1\}^n$, respectively. The goal for us is to decide if Alice and Bob have the same string or not, i.e., is $x = y$ or not. The problem is that Alice and Bob cannot talk to each other and they can only toss their own random coins (as in there is no shared source of randomness between them). Design a solution where both Alice and Bob only send a single message of size

$$O(\sqrt{n}) \cdot \log^{O(1)}(n)$$

bits each to us *directly* and based on their messages, we g

Solution. Suppose Alice and Bob have strings $a \in$ $\sum_{i=0}^{n} a_i x^i$ and $B(x) = \sum_{i=0}^{n} b_i x^i$. Let $\mathbb{F}$ be a field of size

Lemma 1. *Let $r \in \mathbb{F}$ be chosen uniformly at random, D*

So to decide whether Alice and Bob have the same strin $B(r)$ to us (total of $O(\log n)$ bits), and we can output Using the Lemma, this outputs the correct answer with

However, Alice and Bob *do not* share a source of rando pick $\sqrt{n}$ *distinct* numbers from $\mathbb{F}$ and send us both the ations of each number. These messages have size $O(\sqrt{n}$

We claim that they pick at least one number in common $A(r) = B(r)$ for at least one $r \in \mathbb{F}$ and output the corre We first prove this claim:

Claim 2. *Let $\mathbb{F}$ be a field of size n. Say Alice and Bob field. Then, with probability at least $1 - \frac{1}{e^2}$, they have pic*

Proof. Say Alice picks $t = \sqrt{n}$ numbers $\{a_1, a_2, ..., a_t\}$ picks b_1, the probability that it does not equal any of a_i the probability that it does not equal any of $a_1, ..., a_t$ is he has $n-1$ numbers to choose from). So the probabil has picked is (call this the "fail" case):

$$\begin{aligned}
Pr(\text{fail}) &= \left(\frac{n-\sqrt{n}}{n}\right)\left(\frac{n-\sqrt{n}-1}{n-1}\right)\cdots\left(\frac{n-2\sqrt{n}+1}{n-\sqrt{n}+1}\right) \\
&= \left(1-\frac{\sqrt{n}}{n}\right)\left(1-\frac{\sqrt{n}}{n-1}\right)\cdots\left(1-\frac{\sqrt{n}}{n-\sqrt{n}+1}\right) \\
&\leq \exp\left(-\frac{\sqrt{n}}{n}\right)\cdot\exp\left(-\frac{\sqrt{n}}{n-1}\right)\cdots\exp\left(-\frac{\sqrt{n}}{n-\sqrt{n}+1}\right) \qquad (\text{since } 1+x \leq e^x) \\
&= \exp\left(-\frac{\sqrt{n}}{n}-\frac{\sqrt{n}}{n-1}\cdots-\frac{\sqrt{n}}{n-\sqrt{n}+1}\right) \\
&= \exp\left(-\sqrt{n}\left(\frac{1}{n}+\frac{1}{n-1}\cdots+\frac{1}{n-\sqrt{n}+1}\right)\right)
\end{aligned}$$

Now examine,

3

$$\frac{1}{n-\sqrt{n}+1}+\ldots+\frac{1}{n-1}+\frac{1}{n}=\left(\frac{1}{1}+\frac{1}{2}+\ldots+\frac{1}{n}\right)-\left(\frac{1}{1}+\frac{1}{2}+\ldots+\frac{1}{n-\sqrt{n}}\right)$$
$$\approx \ln(n)-\ln(n-\sqrt{n}) \quad \text{(see lecture 13)}$$
$$=\ln\left(\frac{n}{n-\sqrt{n}}\right) \quad (1)$$

So,

$$\begin{aligned}
Pr(\text{fail}) &= \exp\left(-\sqrt{n}\left(\frac{1}{n}+\frac{1}{n-1}+\frac{1}{n-\sqrt{n}+1}\right)\right)\\
&= \exp\left(-\sqrt{n}\cdot\ln\left(\frac{n}{n-\sqrt{n}}\right)\right) && \text{(from (1))}\\
&= \left(\frac{n}{n-\sqrt{n}}\right)^{-\sqrt{n}}\\
&= \left(\frac{n-\sqrt{n}}{n}\right)^{\sqrt{n}}\\
&= \left(1-\frac{1}{\sqrt{n}}\right)^{\sqrt{n}}\\
&\leq \frac{1}{e} && \text{(since } (1+\tfrac{x}{n})^n \leq e^x\text{)}
\end{aligned}$$

So then, $Pr(\text{success}) \geq 1-\frac{1}{e}$ □

Then, by simply repeating this $k = 18\ln(2n)$ times and returning the majority answer, we can output the correct answer with probability at least $1-\frac{1}{e^{\ln(2n)}}$:

Proof. Let $X_1,...,X_k$ be k indicator random variables where $X_i = \begin{cases}1 & \text{if the ith run returns the correct answer}\\ 0 & \text{otherwise}\end{cases}$

Let $X=\sum_{i=1}^{k} X_i$. We know $E[X] \geq \frac{2}{3}\cdot k$ since each success $(X_i=1)$ occurs with probability at least $\frac{2}{3}$. Furthermore, for the majority to be wrong, we need $X \leq \frac{k}{2}$.

$$\begin{aligned}
Pr(\text{majority is wrong}) &\leq Pr\left(|X-E[X]| \geq \frac{k}{6}\right.\\
&\leq 2\cdot\exp\left(-\frac{2k^2}{36k}\right)\\
&= 2\cdot\exp(-\ln(2n))\\
&= \frac{1}{n}
\end{aligned}$$

So the probability that the algorithm outputs the right answer is

Finally, we are now sending messages of size $18\ln(2n)\cdot O(\sqrt{n}\cdot\log$ desired.

4

instead of using majority, it is easier to realize that if (one iteration of) the algorithm returns "not equal" then the two strings must not be equal, so that the overall algorithm can return "not equal" iff one of the iterations returns "not equal". The success probability will be $1-(1/e)^k$.

Q3 50 / 50

Refer to the assignment PDF.

Problem 3. Consider the following linear program for the set cover problem of sets $S_1, \ldots, S_m$ from the universe $[n]$, which we studied in Lecture 10:

$$\begin{aligned} \min_{x \in \mathbb{R}^m} & \sum_{i=1}^{m} x_i \\ \text{subject to} & \sum_{S_i \ni e} x_i \geqslant 1 && \forall e \in [n] \\ & x_i \geqslant 0 && \forall i \in [m] \end{aligned}$$

We use the MWU technique to design a $(1+\epsilon)$-approximation algorithm for this LP for a given $\epsilon > 0$.

a) For every element $e \in [n]$, maintain a weight w_e and let $W := \sum_{e \in [n]} w_e$. Consider this oracle LP:

$$\begin{aligned} \min_{x \in \mathbb{R}^m} & \sum_{i=1}^{m} x_i \\ \text{subject to} & \sum_{e \in [n]} w_e \cdot \sum_{S_i \ni e} x_i \geqslant W \\ & x_i \geqslant 0 && \forall i \in [m] \end{aligned}$$

Design an algorithm for finding the optimum solution to this LP and prove that the value of this solution is always upper bounded by that of the original LP for set cover.

(10 points)

Solution. We wish to prove that the value of the optimum solution to this oracle LP is always upper bounded by that of the original LP for set cover.

Proof. The proof follows from a similar claim in Lec19. Any feasible solution x to the original LP must satisfy the following equation for each $e \in E$:

$$\sum_{S_i \ni e} x_i \geqslant 1$$

If we multiply each side by $w_e \geqslant 0$, we obtain:

$$w_e \cdot \sum_{S_i \ni e} x_i \geqslant w_e$$

By summing for all edges, we obtain:

$$\sum_{e} w_e \cdot \sum_{S_i \ni e} x_i \geqslant \sum_{e} w_e = W$$

Thus, any feasible solution to the original LP must also be a feasible solution to the oracle LP. As such, the optimum solution to this oracle LP is always upper bounded by that of the original LP for set cover, as desired. □

Algorithm:

For every set S_i ($i \in [m]$), we define the *artificial weight* of the set S_i to be:

$$aw(S_i) := w_{v_1} + w_{v_2} + \ldots w_{v_{|S_i|}}$$

where $e_1, e_2, \ldots, e_{|S_i|}$ are all the edges in the set S_i. We have:

$$\begin{aligned} \sum_{e} w_e \cdot \sum_{S_i \ni e} x_i &= \sum_{S_i \text{ for } e \in [m]} x_i \cdot (w_{v_1} + w_{v_2} + \ldots w_{v_{|S_i|}}) \\ &= \sum_{S_i \text{ for } e \in [m]} aw(S_i) \cdot x_i \end{aligned}$$

Hence, the main constraint of the oracle LP is:

$$\sum_{S_i \text{ for } e \in [m]} aw(S_i) \cdot x_i \geqslant W$$

This means that to obtain the optimal solution for the oracle LP, we simply require.

$$S_{i^*} = \underset{S_i \text{ for } e \in [m]}{\operatorname{argmax}} \; aw(S_i)$$

and then set:

$$x_{i^*} = \frac{W}{aw(S_{i^*})} \text{ and } x_i = 0 \text{ for all other } i \neq i^*$$

This clearly minimizes $\sum_{i=1}^{m} x_i$. "Moving" any value of x_{i^*} anywhere else (i.e., to any set S_i where $aw(S_i) < aw(S_{i^*})$) will violate constraints without improving the objective value.

(a)
10/10

b) Consider the following update rule in the MWU for a given solution $x^{(t)}$ to the oracle LP of part (a) for weights $w^{(t)}$ at iteration $t \geqslant 1$. For any element $e \in [n]$, define $x_e^{(t)} := \sum_{S \ni e} x_S^{(t)}$ and update:

$$w_e^{(t+1)} \leftarrow (1 - \eta \cdot x_e^{(t)}) \cdot w_e^{(t)}$$

for some $\eta > 0$ that you will need to choose later. Prove the following two equations after running the MWU algorithm with the above update rules for T iterations:

$$W^{(T)} \leqslant \exp(-\eta \cdot T + \ln n)$$

$$w_e^{(T)} \geqslant \exp\left(-\eta \cdot \sum_{t=1}^{T} x_e^{(t)} - \eta^2 \cdot \sum_{t=1}^{T} x_e^{(t)^2}\right)$$

Note that you need to pick η properly to be able to prove the above bounds.

(20 points)

Solution. Solution.

Claim 1. For every choice of $T > 1$, we have that at the end of the last iteration $W^{(T+1)} \leqslant \exp(-\eta \cdot T + \ln n)$.

Proof. For every $t \geqslant 1$:

$$\begin{aligned}
W^{(t+1)} &= \sum_{e \in [n]} w_e^{(t+1)} && \text{by definition of } W^{(t+1)} \\
&= \sum_{e \in [n]} (1 - \eta \cdot x_e^{(t)}) \cdot w_e^{(t)} && \text{by the update rule} \\
&= \sum_{e \in [n]} \left(w_e^{(t)}\right) - \eta \cdot \left(\sum_{e \in [n]} w_e^{(t)} \cdot x_e^{(t)}\right) \\
&= W^{(t)} - \eta \cdot \left(\sum_{e \in [n]} w_e^{(t)} \cdot x_e^{(t)}\right) && \text{by definition of } W^{(t)} \\
&\leqslant W^{(t)} - \eta \cdot W^{(t)} && \text{by the feasibility of } x^{(t)} \text{ in the oracle LP} \\
&= (1 - \eta) \cdot W^{(t)}
\end{aligned}$$

We maintain that $w_e^{(1)} = 1$ initially, as such, $W^{(1)} = n$ and after the T-th iteration, we obtain:

$$W^{(T+1)} \leqslant (1 - \eta)^T \cdot n \leqslant \exp(-\eta \cdot T) \cdot n = \exp(-\eta \cdot T + \ln n)$$

where we use $1 - x \leqslant e^{-x}$.

Claim 2. For every choice of $T > 1$, we have that at the end of the last iteration $w_e^{(T+1)} \geqslant \exp\left(-\eta \cdot \sum_{t=1}^{T} x_e^{(t)} - \eta^2 \cdot \sum_{t=1}^{T} x_e^{(t)^2}\right)$.

Proof. Let $p_e = \max_{t \geqslant 1} x_e^{(t)}$. We pick $\epsilon \in (0, \frac{1}{2})$ and $\eta \leqslant \frac{\epsilon}{p_e}$. Then we have:

$$\begin{aligned}
w_e^{(T+1)} &= \prod_{t=1}^{T} \left(1 - \eta \cdot x_e^{(t)}\right) && \text{by the update rule since } w_e^{(1)} = 1 \\
&\geqslant \prod_{t=1}^{T} \exp\left(-\eta \cdot x_e^{(t)} - \eta^2 \cdot x_e^{(t)^2}\right)
\end{aligned}$$

We use $1 - y \geqslant e^{-y - y^2}$ for $-y \in [-\frac{1}{2}, 0]$. This holds since the smallest possible value of $0 \geqslant -\eta \cdot x_e^{(t)} \geqslant -\frac{\epsilon}{p_e} \cdot p_e = -\frac{1}{2} \geqslant -\frac{1}{2}$.

From the above equation, we derive:

$$w_e^{(T+1)} \geqslant \exp\left(-\eta \cdot \sum_{t=1}^{T} x_e^{(t)} - \eta^2 \cdot \sum_{t=1}^{T} x_e^{(t)^2}\right)$$

Thus, we have proven the above bounds, as desired. □

(b)
20/20

c) Use the previous two steps to design a polynomial time algorithm based on MWU that outputs a $(1+\epsilon)$-approximation to the set cover LP. Remember to both bound the number of iterations of your MWU algorithm as well as the time that it takes to solve the oracle LP in each iteration.

(20 points)

Solution. Solution. Algorithm:

(a) $\forall e \in [n]$, let $w_e^{(1)} = 1$

(b) For t = 1 to T iterations:

i. let $x^{(t)}$ be the optimal solution to the oracle LP with weights $w^{(t)}$ according to what we've done in 3(a).

ii. $\forall e \in [n]$, update $w_e^{(t+1)} = (1 - \eta \cdot x_e^{(t)}) \cdot w_e^{(t)}$, where $x_e^{(t)} = \sum_{e_i \in S} x_i^{(t)}$. Notice that we're only updating the weight of those elements in the chosen set S_i while keeping all other elements retaining their original weights.

(c) Return the final solution:

$$\bar{x} = \frac{1}{T} \cdot \sum_{t=1}^{T} x^{(t)}$$

We want to prove that:

Lemma: For a proper choice of η and T, the output $\bar{x}$ of our algorithm satisfies the following:

$$(1) \sum_{i \in [m]} \bar{x}_i \leqslant OPT$$

$$(2) \bar{x}_e \geqslant \frac{1}{1+\epsilon}$$

proof of (1): In 3(a), we have proved that the value of the optimal solution to the oracle LP is always upper bounded by that of the original LP for the set cover problem. Thus, the average solution $\bar{x}$ over T iterations is also always upper bounded by the optimal objective value of the original LP.

proof of (2): In 3(b), we've proved **Claim1** and **Claim 2**. Let's bound the number of iterations T that's required to output a $(1+\epsilon)$-approximation to the original set cover LP using these two claims: for any vertex $e \in V$ and $\eta \leqslant \frac{1}{2p_e}$:

$$\exp\left(-\eta \cdot \sum_{t=1}^{T} x_e^{(t)} - \eta^2 \cdot \sum_{t=1}^{T} x_e^{(t)^2}\right) \leqslant w_e^{(T+1)} \leqslant W^{(T+1)} \leqslant \exp(-\eta \cdot T + \ln n)$$

What will be the range of T if at the end of the T iterations, there is an element e that violates (2) such that

$$\frac{1}{T} \cdot \sum_{t=1}^{T} x_e^{(t)} < \frac{1}{1+\epsilon}$$

Let's assume that:

$$(3) \sum_{t=1}^{T} x_e^{(t)} < \frac{T}{1+\epsilon}$$

So,

$$\begin{aligned}
w_e^{(T+1)} &\geqslant \exp\left(-\eta \cdot \sum_{t=1}^{T} x_e^{(t)} - \eta^2 \cdot \sum_{t=1}^{T} x_e^{(t)^2}\right) \\
&= \exp\left(-(\eta \cdot \sum_{t=1}^{T} x_e^{(t)} + \eta^2 \cdot p_e \cdot \sum_{t=1}^{T} x_e^{(t)^2}\right) \\
&> \exp\left(-(\eta \cdot \frac{T}{1+\epsilon} + \eta^2 \cdot \sum_{t=1}^{T} x_e^{(t)^2}\right) \\
&> \exp\left(-(\eta \cdot \frac{T}{1+\epsilon} + \eta^2 \cdot p_e \cdot \sum_{t=1}^{T} x_e^{(t)}\right) && \text{since we defined } p_e = max_{i:i \ni e} x_e^{(t)} \\
&\geqslant \exp\left(-(\eta \cdot \frac{T}{1+\epsilon} + (\frac{\epsilon}{2p_e}) \cdot \eta \cdot p_e \cdot \sum_{t=1}^{T} x_e^{(t)}\right) && \text{since } \eta \leqslant \frac{\epsilon}{2p_e} \\
&= \exp\left(-(\eta \cdot \frac{T}{1+\epsilon} + \frac{\epsilon}{2} \cdot \eta \cdot \sum_{t=1}^{T} x_e^{(t)}\right) \\
&> \exp\left(-(\eta \cdot \frac{T}{1+\epsilon} + \frac{\epsilon}{2} \cdot \eta \cdot \frac{T}{1+\epsilon}\right) \\
&= \exp\left(-(1+\frac{\epsilon}{2}) \cdot \eta \cdot \frac{T}{1+\epsilon}\right)
\end{aligned}$$

So, we have:

$$\exp\left(-(1+\tfrac{\epsilon}{2}) \cdot \eta \cdot \frac{T}{1+\epsilon}\right) \leqslant w_e^{(T+1)} \leqslant W^{(T+1)} \leqslant \exp(-\eta \cdot T + \ln n)$$

By solving this inequality, we'll get

$$\begin{aligned}
T &\leqslant \frac{2 \cdot ln n \cdot (1+\epsilon)}{\epsilon \cdot \eta} \\
&= \frac{4p \cdot ln n \cdot (1+\epsilon)}{\epsilon^2}
\end{aligned}$$

Thus, after these many iterations, all vertices violated (3) and thus in turn satisfy (2). Therefore, for any $\epsilon \in (0, \frac{1}{2}]$, if we run the algorithm for $T > \frac{8p \ln n(1+\epsilon)}{\epsilon^2}$ iterations for $p \geqslant max_{e \in [n]} max_{I \ni e} x_e^{(t)}$ while parameter $\eta = \frac{\epsilon}{4p}$, then the output solution $\bar{x}$ satisfies (2). Then, to convert $\bar{x}$ to be a feasible solution to the original LP, we can simply scale up $\bar{x}$ by $(1+\epsilon)$, and the corresponding objective value will be a $(1+\epsilon)$- approximation after the scaling.

We know that in the original LP, $OPT \leqslant m$ since we have at most m sets and it's a min problem. Also, since $OPT_{modified\,LP} \leqslant OPT$, we can conclude that $p \leqslant m$. now we can bound T by $O(\frac{m \ln n(1+\epsilon)}{\epsilon^2})$, and therefore, we now have an actual algorithm with known parameters throughout. Each iteration requires us to find the maximum $wt(S_i)$, and each update only changes the weight of $O(m)$ other sets.

So, the total run-time is $O(T \cdot m) = O(\frac{m^2 \ln n(1+\epsilon)}{\epsilon^2})$ if we consider using a Fibonacci heap. (note that we could also use a max-heap and run each iteration of the algorithm in $O(nlogn)$ time)

(c) 20/20

well done!

Q4 0 / 15

Refer to the assignment PDF.

This question wasn't answered

Q5 0 / 25

Refer to the assignment PDF.

This question wasn't answered