Question 1.

a).

I). Describe the reduction (c-Fair Sharing Problem $\leq_p$ Knapsack Problem)

Knapsack Problem variables vs c-Fair Sharing Problem variables:

- Values (v_i) in the set (U) = the set of pay offs (p_is) in T
- Weights (w_i) in the set (U) = the set of pay offs (p_is) in T
- Capacity (W) = the half of the sum of all the pay offs ($\frac{1}{2}\sum_{t_i \in T} p_i$)
- Target (V) = The constant value (c) times the sum of all the pay offs ($c * \frac{1}{2}\sum_{t_i \in T} p_i$)
- Constant value (c) = just random constant value which is given as $0 < c \leq 1$

II). The running time of algorithm

- **Reduction time (rearrange the items set with a for-loop in n_2 times):** O(n_2)
- **The running time for Knapsack:** O($f_1(n_1, W, V)$)
- **The final total running time:** O(n_2) + O($f_1(n_1, W, V)$) = O($n_1 + f_1(n_1, W, V)$)

III). The proof of the correctness of the algorithm

If assume the c-Fair Sharing Problem is correct ($c * \frac{1}{2}\sum_{t_i \in T} p_i \leq \sum_{t_i \in T_1} p_i \leq \frac{1}{2}\sum_{t_i \in T} p_i$):

- For the inequality between the disjoint sets and the whole set (T) in c-Fair sharing, "the total value is greater than or equal to target V" will be hold because the sum of the all pay offs (represents as the total values $\sum_{t_i \in T} v_i$) is greater than or equal to the constant value (c) times the half of the sum of the all pay offs (represents as the target V)

$$c * \frac{1}{2}\sum_{t_i \in T} p_i \leq \sum_{t_i \in T_1} p_i \text{ equals to } V \leq \sum_{t_i \in T} v_i$$

- "The weight less than or equal to the capacity W" will be hold because the sum of the pay offs (represents as the weight w_i) is less than or equal to the half of the sum of all pay offs (represents as the capacity W).

$$\sum_{t_i \in T_1} p_i \leq \frac{1}{2}\sum_{t_i \in T} p_i \text{ equals to } w_i \leq W$$

- Hence if yes instance for the c-Fair Sharing, then yes instance for the knapsack

If assume the Knapsack Problem is correct ($V \leq \sum_{t_i \in T} v_i$ and $w_i \leq W$):

- For the inequality between the subset items and capacity W, target V in Knapsack problem, "$c * \frac{1}{2}\sum_{t_i \in T} p_i \leq \sum_{t_i \in T_1} p_i \leq \frac{1}{2}\sum_{t_i \in T} p_i$" will be hold because the total value (represents as sum of the payoffs $\sum_{t_i \in T_1} p_i$) is greater than or equal to the target (represents as c times the half of the sum of the pay offs $c * \frac{1}{2}\sum_{t_i \in T} p_i$); and the weight w_i (represents as sum of the payoffs $\sum_{t_i \in T_1} p_i$) is less than or equal to the capacity W (represents as the half of the sum of all pay offs $\frac{1}{2}\sum_{t_i \in T} p_i$)

$$V \leq \sum_{t_i \in T} v_i \text{ equals to } c * \frac{1}{2}\sum_{t_i \in T} p_i \leq \sum_{t_i \in T_1} p_i.$$

and

$$w_i \leq W \text{ equals to } \sum_{t_i \in T_1} p_i \leq \frac{1}{2}\sum_{t_i \in T} p_i$$

- Hence, if yes instance for the knapsack, then yes instance for the c-Fair Sharing

b).

I). Describe the reduction (Fair-Pairwise Sharing Problem $\leq_p$ c-Fair Sharing Problem)

c-Fair Sharing Problem variables vs Fair-Pairwise Sharing Problem variables

- the set of pay offs (p_is) in T = The set of tasks ($a_i - b_i$)
- the constant value (c) = 1

II). The running time of algorithm

- **Reduction time:**
  (rearrange the pay offs set with a for-loop in n_3 times): O(n_3)
  (rearrange the items set with a for-loop in n_2 times): O(n_2)

- **The running time for Knapsack:** O($f_1(n_1, W, V)$)
- **The final total running time:**
  O(n_2)+ O(n_3) + O($f_1(n_1, W, V)$) = O($n_2 + n_3 + f_1(n_1, W, V)$)

III). The proof of the correctness of the algorithm

$\sum_{(a_i,b_i)\in F_1} a_i + \sum_{(a_i,b_i)\in F_2} b_i = \sum_{(a_i,b_i)\in F_2} a_i + \sum_{(a_i,b_i)\in F_1} b_i$ can be transformed as $\sum_{(a_i,b_i)\in F_1} a_i - \sum_{(a_i,b_i)\in F_1} b_i = \sum_{(a_i,b_i)\in F_2} a_i - \sum_{(a_i,b_i)\in F_2} b_i$ since we put the same disjoint sets in each same side.

If assume the Fair-Pairwise Sharing Problem is correct ($\sum_{(a_i,b_i)\in F_1} a_i - \sum_{(a_i,b_i)\in F_1} b_i = \sum_{(a_i,b_i)\in F_2} a_i - \sum_{(a_i,b_i)\in F_2} b_i$):

- For the equation between the disjoint sets and pair tasks in Fair-Pairwise Sharing problem, "$c * \frac{1}{2}\sum_{t_i\in T} p_i \leq \sum_{t_i\in T_1} p_i \leq \frac{1}{2}\sum_{t_i\in T} p_i$" will be hold since the difference between sum of a_i and sum of b_i in F_1 (represents as c times the half of the sum of the pay offs $c * \frac{1}{2}\sum_{t_i\in T} p_i$, $c = 1$) equals to the difference between between sum of a_i and sum of b_i in F_2 (represents as the half of the sum of all pay offs $\frac{1}{2}\sum_{t_i\in T} p_i$)
- Hence if yes instance for the Fair-Pairwise sharing instance, then yes instance for the c-Fair sharing instance

If assume the c-Fair Sharing Problem is correct ($c * \frac{1}{2}\sum_{t_i\in T} p_i \leq \sum_{t_i\in T_1} p_i \leq \frac{1}{2}\sum_{t_i\in T} p_i$):

- For the inequality between the disjoint sets and the whole set (T) in c-Fair sharing, "$\sum_{(a_i,b_i)\in F_1} a_i - \sum_{(a_i,b_i)\in F_1} b_i = \sum_{(a_i,b_i)\in F_2} a_i - \sum_{(a_i,b_i)\in F_2} b_i$" will be hold since the constant value (c) times the half of the sum of the pay offs (represents as the difference between sum of a_i and sum of b_i in F_1) equals the sum of the pay offs in the disjoint set (represents as the difference between sum of a_i and sum of b_i in F_2) when constant value (c) is 1, and the sum of the pay offs in the disjoint set (represents as the difference between sum of a_i and sum of b_i in F_1) can also equals to the half of the sum of the pay offs (represents as the difference between sum of a_i and sum of b_i in F_2).

- Hence, if yes instance for the c-Fair sharing, then yes instance for the Fair-Pairwise sharing

Both assumptions final equalities would be:

$$\sum_{(a_i,b_i)\in F_1} a_i - \sum_{(a_i,b_i)\in F_1} b_i = \sum_{(a_i,b_i)\in F_2} a_i - \sum_{(a_i,b_i)\in F_2} b_i$$

equals to

$$c * \frac{1}{2}\sum_{t_i\in T} p_i = \sum_{t_i\in T_1} p_i \ When\ c = 1$$

and

$$\sum_{(a_i,b_i)\in F_1} a_i - \sum_{(a_i,b_i)\in F_1} b_i = \sum_{(a_i,b_i)\in F_2} a_i - \sum_{(a_i,b_i)\in F_2} b_i$$

equals to

$$\sum_{t_i\in T_1} p_i = \frac{1}{2}\sum_{t_i\in T} p_i.$$

c).

I). Describe the reduction (c-Fixed Distance Problem $\leq_p$ Fair-Pairwise Sharing Problem)

Fair-pairwise Sharing Problem variables vs c-Fixed Distance Problem variables

- The n_3 pair tasks (a_i, b_i) in the set F = The n_4 pair tasks with the direction and the distance of the commands (right command $(li, 0)$; left command $(0, -li)$) and the extra $n_4 + 1\ th$ pair task which include double of the target $(2 * c * \sum_{x_i\in X} l_i)$ times constant (c) and the actual total distance with the direction $(\sum_{x_i\in X}(right\ (+)\ or\ left(-)) * l_i)$
- Constant value (c) = just random constant value which is given as $\frac{1}{2} \leq c < 1$

Example case: (left 1, right 1, left 2, right 4, right 3, c = 0.5)

Translate to the pair tasks of the set:

$F = \{(0,-1),(1,0),(0,-2),(4,0),(3,0),(2 * 0.5 * (1+1+2+4+3),(-1+1-2+4+3))\}$

II). The running time of algorithm

- **Reduction time:**
  (rearrange the pair-tasks set with a for-loop in n_4 times): O(n_4)
  (rearrange the pay offs set with a for-loop in n_3 times): O(n_3)
  (rearrange the items set with a for-loop in n_2 times): O(n_2)
- **The running time for Knapsack:** O($f_1(n_1, W, V)$)
- **The final total running time:**
  O(n_4) + O(n_3)+ O(n_2) + O($f_1(n_1, W, V)$) = O($n_4 + n_3 + n_2 + f_1(n_1, W, V)$)

III). The proof of the correctness of the algorithm

$\sum_{(a_i,b_i)\in F_1} a_i + \sum_{(a_i,b_i)\in F_2} b_i = \sum_{(a_i,b_i)\in F_2} a_i + \sum_{(a_i,b_i)\in F_1} b_i$ can be transformed as $\sum_{(a_i,b_i)\in F_1} a_i - \sum_{(a_i,b_i)\in F_1} b_i = \sum_{(a_i,b_i)\in F_2} a_i - \sum_{(a_i,b_i)\in F_2} b_i$ since we put the same disjoint sets in each same side.

If assume c-Fixed Distance problem is correct ($2c * \sum_{x_i\in X} l_i$):

- For the formula $2c * \sum_{x_i\in X} l_i$ in c-Fixed Problem, the claim "$\sum_{(a_i,b_i)\in F_1} a_i - \sum_{(a_i,b_i)\in F_1} b_i = \sum_{(a_i,b_i)\in F_2} a_i - \sum_{(a_i,b_i)\in F_2} b_i$" will be hold because $2c * \sum_{x_i\in X} l_i$ equals the constant value c times the sum of all the distances (represents as($\sum_{(a_i,b_i)\in F_1} a_i - \sum_{(a_i,b_i)\in F_1} b_i) + (\sum_{(a_i,b_i)\in F_2} a_i - \sum_{(a_i,b_i)\in F_2} b_i)$).

  $$(\sum_{(a_i,b_i)\in F_1} a_i - \sum_{(a_i,b_i)\in F_1} b_i) + (\sum_{(a_i,b_i)\in F_2} a_i - \sum_{(a_i,b_i)\in F_2} b_i) = 2c * \sum_{x_i\in X} l_i$$

- Hence, if yes instance for the c-Fixed Distance, then yes instance for the Fair-Pairwise Sharing

If assume Fair-Pairwise Sharing problem is correct ($\sum_{(a_i,b_i)\in F_1} a_i - \sum_{(a_i,b_i)\in F_1} b_i = \sum_{(a_i,b_i)\in F_2} a_i - \sum_{(a_i,b_i)\in F_2} b_i$):

- For the equation between the disjoint sets and pair tasks in Fair-Pairwise Sharing problem, the claim "$2c * \sum_{x_i\in X} l_i$" will be hold because the difference between the sum of a_i and the sum of b_i in F_1 (represents as $c * \sum_{x_i\in X} l_i$) equals to the difference between the sum of a_i and the sum of b_i in F_2 (represents as $c * \sum_{x_i\in X} l_i$). The sum of $a_i - b_i$ in two disjoint sets F_1 and F_2 equals to the double of the one of the disjoint sets (represents as $2c * \sum_{x_i\in X} l_i$).

$$(\sum_{(a_i,b_i)\in F_1} a_i - \sum_{(a_i,b_i)\in F_1} b_i) + (\sum_{(a_i,b_i)\in F_2} a_i - \sum_{(a_i,b_i)\in F_2} b_i) = 2c * \sum_{x_i\in X} l_i$$

- Hence, if yes instance for the Fair-Pairwise Sharing, then yes instance for the c-Fixed Distance

Question 2.

I). Prove c-Fair Sharing decision problem is NP-complete:

Prove c-Fair Sharing Decision Problem is in NP

- **Output of the problem:** True or False
- **Certifier:**
  Given a set T' = { p_1' …. p_n'} and divide the set as two disjoint sets T_1' and T_2'. Compare the sum of the values p_i' of the disjoint set would be greater than or equal to a constant value c' (which $0 < c' \leq 1$) times the half of the sum of all the values ($c' * \frac{1}{2}\sum_{t_i'\in T_1' or T_2'} p_i'$) in T' and less than or equal to the half of the sum of all the values ($\frac{1}{2}\sum_{t_i'\in T'} p_i'$ in T'). If the disjoint sets match the requirements then the algorithm would return true, else it would return false.
  The running time of it would be O(n^2) since T' needs to be divided and rearrange the values lists in two disjoint sets by running through the double for- loop.

- **The output of the c-Fair Sharing Decision Problem in the certifier will be run in O(n^2). Hence, the c-Fair Sharing Decision Problem is solved in polynomial time.**

- **Therefore, the c-Fair Sharing Decision Problem is in NP**

Prove c-Fair Sharing Decision Problem is NP-Hard

- Since we have proved Fair-Pairwise Sharing $\leq_p$c-Fair Sharing in Question 1, part b
- c-Fair Sharing Decision Problem is NP-Hard

Since c-Fixed Distance Decision problem is NP-complete

- **In transitivity:**
  c-Fixed Distance Decision $\leq_p$ Fair-Pairwise Sharing $\leq_p$c-Fair Sharing

- Hence, c-Fair Sharing problem is NP-complete

II). Prove Fair-Pairwise Scheduling decision problem is NP-complete:

Prove Fair-Pairwise Scheduling decision problem is in NP

- **Output of the problem:** True or False
- **Certifier:**
  Given a set F' with pair tasks {(a_1', b_1')… (a_n', b_n')} and divide the set as two disjoint sets F_1' and F_2'.
  Compare $\sum_{(a_i', b_i') \in F_1}, a_i' - \sum_{(a_i', b_i') \in F_1}, b_i'$ and $\sum_{(a_i', b_i') \in F_2}, a_i' - \sum_{(a_i', b_i') \in F_2}, b_i'$. The algorithm will keep dividing F' and rearranging to two disjoint sets until it finds the two sides comparison are equal and then return true. If the algorithm can not find the equality, then it would return false.
  The running time of it would be O(n^2) since F' needs to be divided and rearrange the pairs lists in two disjoint sets by running through the double for-loop.

- **The output of the Fair-Pairwise sharing problem in the certifier will be run in O(n^2). Hence, the fair-pairwise sharing problem is solved in polynomial time.**

- **Therefore, the Fair-Pairwise Sharing problem is in NP**

Prove Fair-Pairwise sharing problem is NP-Hard

- We have already proved that c-Fixed Distance Decision $\leq_p$ Fair-Pairwise Sharing in Question 1, part c.

- Hence, Fair-Pairwise sharing problem is NP-Hard

Since c-Fixed Distance Decision problem is NP-complete

- **In transitivity:**
  c-Fixed Distance Decision $\leq_p$ Fair-Pairwise Sharing

- Hence, Fair-Pairwise Scheduling decision problem is NP-complete

III). Prove Knapsack decision problem is NP-complete:

Prove Knapsack decision problem is in NP

- **Output of the problem:** True or False
- **Certifier:**
  Given a set U' with pair tasks $\{u_1{}' \ldots u_n'\}$ and extract a subset $U_1{}'$ from U'. Check whether the sum of the values $(\sum_{u_i \in U'} u_i{}')$ of the subset is greater than or equal to a constant value c' (which $0 < c' \leq 1$) times the half of the sum of all values $(c' * \frac{1}{2}\sum_{u_i \in U'} u_i{}')$ in U'; and the each value $(u_i{}')$ in the subset is less than or equal to the half of the sum of all values $(\frac{1}{2}\sum_{u_i \in U'} u_i{}')$. If the subset matches the requirements, then the algorithm would return true, else return false.
  The running time of it would be O(n) since the subset U_1' needs to be extracted and rearrange the values lists from the set U' by running through the a for- loop.

- **The output of the Knapsack decision problem in the certifier will be run in O(n). Hence, the Knapsack decision problem is solved in polynomial time.**

- **Therefore, the Knapsack decision problem is in NP**

Prove Knapsack decision problem is NP-Hard

- Since we have proved c-Fixed Distance Decision $\leq_p$ Knapsack decision problem in Question 1, part a

- Hence, the Knapsack decision problem is NP-hard

Since c-Fixed Distance Decision problem is NP-complete

- **In transitivity:**
  c-Fixed Distance Decision $\leq_p$ Fair-Pairwise Sharing $\leq_p$c-Fair Sharing $\leq_p$ Knapsack

- Hence, Knapsack decision problem is NP-complete