**Question 1.**

**a).**

**I). Describe the reduction (c-Fair Sharing Problem Knapsack Problem)**

**Knapsack Problem variables c-Fair Sharing Problem variables:**

* Values () in the set () = the set of pay offs (s) in
* Weights () in the set () = the set of pay offs (s) in
* Capacity () = the half of the sum of all the pay offs ( )
* Target () = The constant value times the sum of all the pay offs ()
* Constant value (c) = just random constant value which is given as

**II). The running time of algorithm**

* **Reduction time (rearrange the items set with a for-loop in times):** O()
* **The running time for Knapsack:** O()
* **The final total running time:** O() + O() = O()

**III). The proof of the correctness of the algorithm**

**If assume the c-Fair Sharing Problem is correct ( ):**

* For the inequality between the disjoint sets and the whole set () in c-Fair sharing, “the total value is greater than or equal to target ” will be hold because the sum of the all pay offs (represents as the total values ) is greater than or equal to the constant value () times the half of the sum of the all pay offs (represents as the target )

*equals to*

* “The weight less than or equal to the capacity ” will be hold because the sum of the pay offs (represents as the weight ) is less than or equal to the half of the sum of all pay offs (represents as the capacity ).

*equals to*

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

**If assume the Knapsack Problem is correct ():**

* For the inequality between the subset items and capacity , target in Knapsack problem, “” will be hold because the total value (represents as sum of the payoffs ) is greater than or equal to the target ( times the half of the sum of the pay offs ); and the weight (represents as sum of the payoffs ) is less than or equal to the capacity (represents as the half of the sum of all pay offs )

.

**and**

*equals to*

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

**b).**

**I). Describe the reduction (Fair-Pairwise Sharing Problem c-Fair Sharing Problem)**

**c-Fair Sharing Problem variables Fair-Pairwise Sharing Problem variables**

* the set of pay offs (s) in = The set of tasks ()
* the constant value () = 1

**II). The running time of algorithm**

* **Reduction time:**

**(rearrange the pay offs set with a for-loop in times):** O()

**(rearrange the items set with a for-loop in times):** O()

* **The running time for Knapsack:** O()
* **The final total running time:**

O()+ O() + O() = O()

**III). The proof of the correctness of the algorithm**

= can be transformed as = since we put the same disjoint sets in each same side.

**If assume the Fair-Pairwise Sharing Problem is correct ( = )**:

* For the equation between the disjoint sets and pair tasks in Fair-Pairwise Sharing problem, “” will be hold since the difference between sum of and sum of in (represents as times the half of the sum of the pay offs ) equals to the difference between between sum of and sum of in (represents as the half of the sum of all pay offs )
* 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 ()**:

* For the inequality between the disjoint sets and the whole set () in c-Fair sharing, “ = ” will be hold since the constant value times the half of the sum of the pay offs (represents as the difference between sum of and sum of in ) equals the sum of the pay offs in the disjoint set (represents as the difference between sum of and sum of in ) when constant value () is 1, and the sum of the pay offs in the disjoint set (represents as the difference between sum of and sum of in ) can also equals to the half of the sum of the pay offs (represents as the difference between sum of and sum of in ).
* Hence, if yes instance for the c-Fair sharing, then yes instance for the Fair-Pairwise sharing

**Both assumptions final equalities would be:**

=

equals to

**and**

=

equals to

.

**c).**

**I). Describe the reduction (c-Fixed Distance Problem Fair-Pairwise Sharing Problem)**

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

* The pair tasks () in the set = The pair tasks with the direction and the distance of the commands ( right command ; left command ) and the extra pair task which include double of the target () times constant () and the actual total distance with the direction ()
* Constant value (c) = just random constant value which is given as

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

Translate to the pair tasks of the set:

**II). The running time of algorithm**

* **Reduction time:**

**(rearrange the pair-tasks set with a for-loop in times):** O()

**(rearrange the pay offs set with a for-loop in times):** O()

**(rearrange the items set with a for-loop in times):** O()

* **The running time for Knapsack:** O()
* **The final total running time:**

O O+ O + O() = O()

**III). The proof of the correctness of the algorithm**

= can be transformed as = since we put the same disjoint sets in each same side.

**If assume c-Fixed Distance problem is correct ():**

* For the formula in c-Fixed Problem, the claim “ = ” will be hold because equals the constant value times the sum of all the distances (represents as( ).

(

* 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 ( = ):**

* For the equation between the disjoint sets and pair tasks in Fair-Pairwise Sharing problem, the claim “” will be hold because the difference between the sum of and the sum of in (represents as ) equals to the difference between the sum of and the sum of in (represents as ). The sum of in two disjoint sets equals to the double of the one of the disjoint sets (represents as ).

(

* 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 = {} and divide the set as two disjoint sets and . Compare the sum of the values of the disjoint set would be greater than or equal to a constant value (which ) times the half of the sum of all the values in and less than or equal to the half of the sum of all the values in . 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 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 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 Fair-Pairwise Sharing 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 with pair tasks {()… ()} and divide the set as two disjoint sets and .

Compare and . The algorithm will keep dividing 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 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 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 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 with pair tasks {… } and extract a subset from . Check whether the sum of the values of the subset is greater than or equal to a constant value (which ) times the half of the sum of all values

in ; and the each value () in the subset is less than or equal to the half of the sum of all values . 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 needs to be extracted and rearrange the values lists from the set 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 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 Fair-Pairwise Sharing c-Fair Sharing Knapsack

* Hence, Knapsack decision problem is NP-complete