HOMEWORK 6

Name: Tianyu Gu Cooperator: AngLi

Question 1:

a) This proof will follow two steps:

- Show that BP *(2 backpack partition)* is in NP. i.e. $\mathrm{BP} \in \mathrm{NP}$.
- Pick the SS *(subset sum)* problem, which is in NP-complete, and show that: $\mathrm{SS} \le_{\mathrm{p}} \mathrm{BP}$.

i. ***Show that $BP \in NP$:***

This is trivial to show, that, given a 2 partitions of items, we can simple examine whether they are equal or not, in polynomial time. Thus, $\mathrm{BP} \in \mathrm{NP}$.

ii. ***Show that $SS \le_p BP$:***

- **First, we need to show that a SS problem can be reduced to a BP problem:**

  The SS problem is: Given a set $\mathrm{S} = \{\mathrm{s}_1, \mathrm{s}_2, \dots \mathrm{s}_\mathrm{n}\}$ with the weight of each item s_i equals to w_i. Determine if we could find a set P ⊆ S. Such that the total weight of P is exact a certain number t.

  First, suppose the total weight is denoted as W, then we can safely assume that $\mathrm{t} \le \mathrm{W}/2$, since it must be true for either t or W – t. we could add an item s_m with weight $\mathrm{W} - 2\mathrm{t}$ to the set S, let's denote the new set S'. Thus the SS problem for set S and weight t is equal to BP problem with set S'.

- **Secondly, show that $\boldsymbol{SS(S, t)}$ is true iff $\mathbf{BP}(\boldsymbol{s}')$ is true:**

  - If:

    If $\mathrm{SS(S, t)}$ is true, that means we could find a set P ⊆ S with weight W – t. since $\mathrm{P} \subseteq \mathrm{S}'$ is also true, thus the $\mathrm{BP(s')}$ is also true. Note that the total weight of S' is $2\mathrm{W} - 2\mathrm{t}$.

  - Only if:

    If $\mathrm{BP(s')}$ is true, then one of the partition set P in S' must have the added item s_m. Then we could extract this item form P, and the new set P' will have total weight t. Meanwhile, the P' is also a subset of S, thus $SS(S,t)$ is true.

b) For the 2 backpack partition problem, a dynamic programming algorithm could work:

First, let's assume the total weight **w** is even number, since the answer is apparently false for odd weight.

For a set $S = \{s_1, s_2, \dots s_i\}$, if we could find a subset of total weight j, then we say $D(i,j)$ is true. Therefore, we could know that, the $D(i,j)$ is true only if either of the $D(i-1,j)$ or $D(i-1, j-w_i)$ is true. Otherwise it is false.

That is:

$$D(i,j) = D(i-1,j) \text{ or } D(i-1,j-w_i)$$

$D(i-1,j)$ means we could put the w_i in the other set.

$D(i-1,j-w_i)$ means we could put the w_i in the set S.

Therefore, we could construct the table $D(i,j)$ until we found the result for $D(\mathbf{w}/2, n)$.

This algorithm give a running time of $O(\mathbf{w} * n)$, therefore, if w can be written into a polynomial of n, then this problem can be solved in polynomial time.

However, when **w** equals to exponential of n, such as 2^n, then running time will become $O(2^n * n)$, which is not a polynomial time solution.

Question 2:

a) This proof will follow two steps:

- Show that UU *(using up all the refrigerator magnets)* is in NP. i.e. $\mathrm{UU} \in \mathrm{NP}$.
- Pick the 3DM *(3D perfect matching)* problem, which is in NP-complete, and show that: $3\mathrm{DM} \le_{\mathrm{p}} \mathrm{UU}$.

i. Show that $\boldsymbol{UU \in NP}$ ***:***

Given a set S of all the chosen strings, we could count all the numbers of each character and determine if they match the numbers of collecting symbols, i.e. numbers of each letters. Apparently this can be done in polynomial time, Thus, $\mathrm{UU} \in \mathrm{NP}$.

ii. Show that $\boldsymbol{UU \le_p 3DM}$***:***

- **First we need to show that a 3DM problem can be reduced to a UU problem:**

  The perfect 3D matching problem is: given three sets $\mathrm{X}, \mathrm{Y}, \mathrm{Z}$ s.t. $|\mathrm{x}| = |\mathrm{y}| = |\mathrm{z}| = \mathrm{n}$, and given a set $\mathrm{T} \subseteq \mathrm{X} \times \mathrm{Y} \times \mathrm{Z}$ of ordered triples, determine whether there exist a set of n triples in T so that each elements of $\mathrm{X} \cup \mathrm{Y} \cup \mathrm{Z}$ is contained in exactly one of these triples.

  We could reduce the 3D matching as follows:

  For each items in X, Y and Z, consider it as a character (refrigerator magnet), therefore, we will have 3n characters. For each ordered triple in T, we could consider it as string or word that Madison knows. Then we could solve this UU problem, which will equally solve the perfect 3D-matching problem.

- **Secondly, show that** $\mathbf{3DM(X, Y, Z, T)}$ **is true iff** $\mathbf{UU(3n, T)}$ **is true:**

  - If:

    If we could find a set $\mathrm{S} \subseteq \mathrm{T}$, which makes the perfect matching, then we could choose S as the set of spelled out words, and it will exactly used up all the 3n characters (refrigerator magnet).

  - Only if:

    If we could get a set $\mathrm{S} = \{(\mathrm{x_i}, \mathrm{y_l}, \mathrm{z_j})\}$, which exactly used up all the

3n characters, then we could collect all the x_i, y_l, z_j separately to get sets X, Y, Z s.t. $|x| = |y| = |z| = n$. Thus, the S will make the perfect matching.

Therefore, it has been proved that using up all the refrigerator magnets problems is NP-complete.

Question 3:

a) For example $A = \{1, 2,7\}$ and set B = 8. The algorithm in the question will give the set $S = \{1, 2\}$ and the total sum T = 3. However, the optimal set will be $S' = \{1, 7\}$ and the total sum T' = 8. Thus, the sum of the set S, given by the algorithm is smaller than half the total sum of some other feasible subset of A.

b) First, we could sort the set A in decreasing order, let's denote the new set $A' = \{a'_1, a'_2, \dots, a'_n\}$. Then, we could use the similar algorithm as described in part (a).

Pseudo code:

```
get sorted A in decreasing order, $A' = \{a'_1, a'_2, \dots, a'_n\}$
$T = 0$ and $S = \emptyset$
for i = 1 to n:
    if $a'_i \le B$ :
        if $T + a'_i \le B$ :
            $S = S \cup \{a'_i\}$ and $T = T + a'_i$
endfor
return T and S
```

Denote the optimal solution as S' and the sum is T'.

Correctness:

The proof can be divided into two cases:

- If $S = \emptyset$, all the integers in A' is larger than B, then the results S will be $\emptyset$, and T = 0. In this case $T \ge T'/2 = 0$.
- If $S \ne \emptyset$:
    - If S contains all the elements from a certain a'_j to a'_n , i.e. $S = \{a'_j, a'_{j+1}, \dots, a'_n\}$. That means all the elements before a'_j are larger than B , Therefore, they cannot in S'. Thus $T = T' \ge T'/2$.
    - Otherwise, we know that there exist an element a'_i, when S is not empty and $T + a'_i > B$, so that a'_i cannot be added to S. therefore, we have:

      $T + a'_i > B \ge T'$, meanwhile, we know that $a'_i \le T$, since $T \ne \emptyset$.

      Therefore, $2T \ge T + a'_i > T'$, $T \ge T'/2$

Running time:

The running time is $\mathrm{O}(\mathrm{nlog}^{\mathrm{n}} + \mathrm{n}) = \mathrm{O}(\mathrm{nlog}^{\mathrm{n}})$.

c) Comparisons and discussion:

- I chose Java to implement the two problem solving algorithms and the data generation function.
- In the program, since the datasets are generated randomly, I use variable **m** to control the number of trials for the two algorithms and take the average time as result.
- I used the variable **n** to represent the number of items in the package, and **w** to represent the maximum weight of each item, therefore, the half weight W will be $\mathrm{W} = \frac{1}{2}\mathbf{n} * \mathbf{w} = \mathrm{O}(\mathbf{n} * \mathbf{w})$
- When the total weight is odd number, I use $\frac{\mathrm{W}}{2} - 1$ to be the half weight, since the $\frac{\mathrm{W}}{2}$ is not integer and the result is meaningless in this discussion.

i. Comparison:

Theoretically, the approximate greedy algorithm runs in $\mathrm{O}(\mathrm{nlog}^{\mathrm{n}})$ and the dynamic programming runs in$\mathrm{O}(\mathrm{W} * \mathrm{n})$. Below is the table for different n and w I tried, the number of trials is equal to 100.

n/w	100	1000	10000
10	***dp***: 0.09	***dp***: 0.3	***dp***: 1.46
	Approx:0.007	***Approx***:0.0074	***Approx***:0.009
	Time percentage: 7.7%	***Time percentage: 2.46%***	***Time percentage: 0.61%***
	Error rate: ***65%***	***Error rate***: ***26%***	***Error rate***:***2%***
100	***dp***: 1.63	***dp***: 10.51	***dp***: 93.2
	Approx:0.05	***Approx***:0.036	***Approx***:0.0389
	Time percentage: 3%	***Time percentage: 0.34%***	***Time percentage: 0.041%***
	Error rate: ***18%***	***Error rate***: ***83%***	***Error rate***: ***98%***
1000	***dp***: 96.95	***dp***: 967.1	***Heap out of the memory.***
	Approx:0.136	***Approx***: 0.143	
	Time percentage: 0.14%	***Time percentage: 0.0147%***	
	Error rate: ***0%***	***Error rate***: ***18%***	

Time in millisecond

ii. Discussion:

1) Running time analysis:

- ***Approximate greedy algorithm:***

  The results basically meet the expectations:

  - When n remains the same, the running time is irrelevant to the amount of w.
  - When n is increased by 10 times, the running time is basically proportional to the $O(nlog^{n})$, however, when n is too small, the results tend to be biased.

- ***Dynamic programming algorithm:***

  The results basically meet the expectations:

  - When w is increased by 10 times, the total sum of the weight is also increased by 10 times. As results, the running time is also increased the same times. Therefore, the running time is linear to the w, which is as described above: $O(W * n) = O(n^2\mathbf{w})$.

- When n is increased 10 times, the total running time is increased nearly 100 times, which is proportional to the n^2, which is the same as expected.

Therefore, the percentage of time the approximate algorithm to the dynamic programming is $\frac{\mathrm{nlong}^n}{n^2 w} = \frac{\log^n}{nw}$, which is reversely linear promotional to n and w, this has also been examined by the results table. In summary, the dynamic programming method is a pseudo polynomial algorithm, which means when w is exponential to the n, the running time will be exponential too, In contrast, the approximate greedy algorithm will always be $O(n\log^n)$, thus the difference will be significant when n and w become larger. Meanwhile, the space complexity is also a severe issue, since the dynamic programming method need store a $n * w/2$ table, which may cause out of memory.

2) Error rate analysis:

The error rate varies significantly for different n and w. the total average of error rate is: ***26.5%***, which is the half of ***50%***, which is the result of pure guessing. This is actually a little worse than my expectation, since this approximate algorithm will never give a true answer, if the real answer if false. Therefore, I would expect the error rate will be little smaller than 25%. However, this may due to the unbalanced numbers of true and false trials, so this can be considered as acceptable.

It is notable that, when n/w is considerable large or small, the approximate algorithm will give a better results, or even perfect. This is due to the fact that:

- When n/w becomes extremely large, the answer of the problem is tending to be true. Meanwhile, with more items in the package, the divisions of achievable weight become smaller, which means the accuracy of approximate algorithm is also increased.

- When n/w becomes extremely small, the answer of the problem is tending to be false, meanwhile, the approximate algorithm is 100% accurate when the real answer is false, therefore, the accuracy will increase too.

The code can be found as attachment.