Assignment 4 --- Ziming Cui，zxc701

1. [24 = 2+14+4+4] Suppose that in an instance of the coins-in-a-line game the coins have the following values in the line:

$$\{5, 1, 7, 5, 10, 9, 7, 8\}$$

a. What is the maximum that the first player, Alice, can win, assuming that the second player, Bob, plays optimally?

The maximum is 30. I will show the maximum value in a Matrix. You can see it in b question.

b. Using the algorithm from class, show the matrix with correct values for each matrix element, when appropriate, for this example. The matrix is an $n \times n$ matrix. The pseudo-code is also available in the presentation for this problem. Read part c of this problem to help with this.

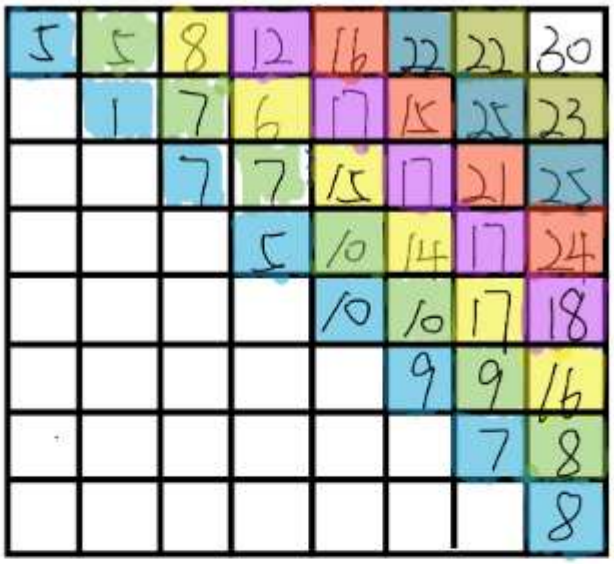
Algo: Coins (C)
Input: list C of integer n≥ 0 coins
Output: maximum value that Alice, the first player, can obtain from C
assuming both players play perfectly

```
M ← new n x n Matrix
for i← to n
  M[i, i] ← C(i)   // only one coin, pick it (base case)

for i ← 1 to (n-1) do // two coins to choose from (base case)
     M[i, i+1] ← max { C(i), C(i+1) }

for Length ← 2 to (n-1) do
  for i ← 1 to (n – Length)
    j ← i + Length
    M[i, j] ← max {
                C(i) + { min { M[i+2, j], m[i+1, j-1] } ,
                C(j) + { min { M[i+1, j-1], M[i, j-2] } }
return M[1,n]
```

M[i,j]



$M[1,3] = \max\{5 + \min\{M[3,3], M[2,2]\},$
$7 + \min\{M[2,2], M[1,1]\}\}$

$= 8$

$M[1,4] = \max\{5 + \min\{M[3,4], M[2,3]\},$
$5 + \min\{M[2,3], M[1,2]\}\}$

$= 12$

$M[1,5] = \max\{5 + \min\{M[3,5], M[2,4]\},$
$10 + \min\{M[2,4], M[1,3]\}\} = 16$

$M[1,6] = \max\{5 + \min\{M[2,5], M[3,6]\},$
$9 + \min\{M[2,5], M[1,4]\} = 22$

$M[1,7] = 22 \quad M[1,8] = 30$

c. Aside from the two base cases, there is a pattern for determining the value stored in the e ntry M[i,j] in the matrix. What is this pattern? That is, rewrite the following wrong pattern for M[i, j].

M[i,j] = max { C(i) + min {3 cells down from M[i,j], 1 cell up from M[i,j] }, C(j) + min{1 cell diagonal down left from M[i,j], 2 cells right} }

My Answer:

$$M_{i,j} = \max\left\{\min\{M_{i+1,j-1}, M_{i+2,j}\} + C[i], \min\{M_{i,j-2}, M_{i+1,j-1}\} + C[j]\right\}.$$

In addition, for i = 1, 2, . . . , n − 1, we have the initial conditions

d. List in order the coins that Alice picks.

My Answer:

Alice = {8, 5, 10, 7}

Bob = {7, 9, 5, 1}

2. [20 = 12+4+4] This is a 0-1 Knapsack Problem. Let S = {item1, item2, item3, item4, item5} be a collection of objects with benefit-weight values item1:($100, 20 lbs), item2:($20,10 lbs), item3:($30,5 lbs), item4:($160, 40 lbs), item5:($90, 30 lbs). This is the order that must be considered when answer questions on this problem. Notice, this is the example given in class except that collection of objects are ordered differently. You must keep the ordering provided in this example. You must show work for part a to receive points for any part of this question.

a. What is the maximum value obtained with a knapsack that can hold 60 lbs? You must show your work by presenting an the matrix with values according to the algorithm. You can give W in increments of 5. That is your matrix will look something like this. Put the correct values in this matrix.

a:

	0	5	10	15	20	25	30	35	40	45	50	55	60
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	100	100	100	100	100	100	100	100	100
2	0	0	20	20	100	100	120	120	120	120	120	120	120
3	0	30	30	50	100	130	130	150	150	150	150	150	150
4	0	30	30	50	100	130	130	150	160	190	190	210	260
5	0	30	30	50	100	130	130	150	160	190	190	220	260

$S = \{(100, 20)_1, (20, 10)_2, (30, 5)_3, (160, 40)_4, (90, 30)_5\}$

$W = 60.$

∴ Maximum Value = 260 $

b. What items are selected?

b:

	0	5	10	15	20	25	30	35	40	45	50	55	60
0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1)	0	0	0	0	100	100	100	100	100	100	100	100	100
2	0	0	20	20	100	100	120	120	120	120	120	120	120
3	0	30	30	50	100	130	130	150	150	150	150	150	150
(4)	0	30	30	50	100	130	130	150	160	190	190	210	260
5	0	30	30	50	100	130	130	150	160	190	190	220	260

item 1, item 4.

c. In class, the items in the set were in non-decreasing by weight. Does the order in which the items in the set produce different answers? Briefly explain why/why not?

In the 0-1 Knapsack Problem, the order in which the items are presented in the set does not produce different answers, as long as the problem is solved using dynamic programming optimally.

The reason for this is that dynamic programming, which is the most common approach for solving the 0-1 Knapsack Problem, considers all possible combinations of items and their weights, regardless of their order. The dynamic programming algorithm iterates through the items and their respective weights, calculating the maximum value that can be obtained while considering each item either for inclusion or exclusion in the knapsack. This is done systematically for all items and weights.

The dynamic programming approach ensures that all possible combinations are explored, and the order of items in the input set does not affect the final result. As long as the algorithm is correctly implemented, it will provide the optimal solution regardless of the initial order of the items in the set.

3. [8] Exercise R-12.6 from the book. Your work must be shown to receive points otherwise no points are earned.

R-12.6 Suppose we are given a set of telescope observation requests, specified by triples of (s_i, f_i, b_i), defining the start times, finish times, and benefits of each observation request as

$$L = \{(1,2,5), (1,3,4), (2,4,7), (3,5,2), (1,6,3), (4,7,5), (6,8,7), (7,9,4)\}.$$

Solve the telescope scheduling problem for this set of observation requests.

According to the Telescope scheduling algo:

$$B[0] \leftarrow 0$$
$$\textbf{for } i = 1 \text{ to } n \textbf{ do}$$
$$\quad B[i] \leftarrow \max\{B[i-1],\ B[P[i]] + b_i\}$$

We start to run this algo, using our data set.

P[1] = 0

Initialization:

B[0] = 0

P[1] = 0 P[2] = 0 P[3] = 1 P[4] = 2 P[5] = 0 P[6] = 4 P[7] = 5 P[8] = 6

Enter into for loop:

i = 1

B[1] = max{B[0], B[P[1]] + b0}

B[0] = 0

B[P[1]] + b1 = B[0] + 5 = 5

So, B[1] = 5

i = 2

B[2] = max{B[1], B[P[2]] + b2}

B[1] = 5

B[P[2]] + b2 = B[0] + 4 = 4

So, B[2] = 5

i = 3

B[3] = max{B[2], B[P[3]] + b3}

B[2] = 5

B[P[3]] + b3 = B[1] + 7 = 12

So, B[3] = 12

i = 4

B[4] = max{B[3], B[P[4]] + b4}

B[3] = 12

B[P[4]] + b4 = B[2] + 2 = 7

So, B[4] = 12

i = 5

B[5] = max{B[4], B[P[5]] + b5}

B[4] = 12

B[P[5]] + b5 = B[0] + 3 = 3

So, B[5] = 12

i = 6

B[6] = max{B[5], B[P[6]] + b6}

B[5] = 12

B[P[6]] + b6 = B[3] + 5 = 17

So, B[6] = 17

i = 7

B[7] = max{B[6], B[P[7]] + b7}

B[6] = 17

B[P[7]] + b7 = B[5] + 7 = 19

So, B[7] = 19

i = 8

B[8] = max{B[7], B[P[8]] + b8}

B[7] = 19

B[P[8]] + b8 = B[6] + 4 = 21

So, B[8] = 21

Finally, we can know that {(1, 2, 5), (2, 4, 7), (4, 7, 5), (7, 9, 4)}, will enter into the shedule, and they will create the maximum benifit 21.

4. [8] Exercise R-12.9 from the book. Hint: Let n represent the weight of the sack. You must explain the variables for the knapsack problems (both the 0-1 and fractional). You must also explain what conditions must be assumed for the 0-1 Knapsack problem and then the conditions that must be assumed for the fractional knapsack problem with respect to this Internet auction that Sally is hosting. Do not give general information.

R-12.9 Sally is hosting an Internet auction to sell n widgets. She receives m bids, each of the form "I want k_i widgets for d_i dollars," for $i = 1, 2, \ldots, m$. Characterize her optimization problem as a knapsack problem. Under what conditions is this a 0-1 versus fractional problem?

In this problem, Sally is hosting an Internet auction to sell n widgets. She receives m bids, each in the form of "I want ki widgets for di dollars," for i = 1, 2, ..., m.

To characterize Sally's optimization problem as a knapsack problem, we can define the following variables:

Variables for the Knapsack Problem: n: The total number of widgets Sally wants to sell. m: The number of bids she receives. ki: The number of widgets requested by the ith bidder. di: The amount the ith bidder is willing to pay for the requested widgets. xi: A binary variable (0 or 1) representing whether the ith bid is selected (xi = 1 if selected, xi = 0 if not selected). Now, let's discuss the conditions for both the 0-1 Knapsack problem and the fractional Knapsack problem with respect to Sally's Internet auction:

Conditions for the 0-1 Knapsack Problem:

In the 0-1 Knapsack problem, Sally has to make a discrete decision for each bid, meaning she can either accept the entire bid or reject it. The number of widgets (ki) for each bid must be integers, as the 0-1 Knapsack problem deals with selecting or rejecting items in whole quantities. The objective is to maximize the total profit or revenue, which is the sum of the amounts paid by the selected bids, subject to the constraint that the total number of widgets selected does not exceed n. Conditions for the Fractional Knapsack Problem:

In the fractional Knapsack problem, Sally can accept fractions of bids, allowing her to split a bid into smaller parts if necessary. The number of widgets (ki) for each bid can be real numbers, including fractions, allowing for more flexibility in the allocation of widgets. The objective is to maximize the total profit or revenue, which is the sum of the amounts paid by the selected bids, subject to the constraint that the total number of widgets selected does not exceed n. So, whether Sally's optimization problem is a 0-1 Knapsack problem or a fractional Knapsack problem depends on whether she is allowed to split bids into fractions or if she must make a discrete decision for each bid. If she can only accept or reject bids as a whole, it is a 0-1 Knapsack problem. If she can accept fractions of bids, it is a fractional Knapsack problem.

5. [8] Show that, in the coins-in-a-line-game, a greedy strategy of having the first player, Alice, always choose the available coin with the highest value will not necessarily result in an optimal solution (or even a winning solution) for her. Do this by giving an example with 6 coins.

[2 100 6 5 4 1]

Alice: 2. 6 4

Bob: 100 5 1.

100+5+1 > 2+6+4

∴ Greedy method can't help Alice win this game.

6. [8] Show that, in the coins-in-a-line game, a greedy-denial strategy of having the first player, Alice, always choose the available coin that minimizes the maximum value of the coin available to Bob will not necessarily result in an optimal solution for her. You can do this by giving an example with coins.

[3, 6, 6, 5, 4, 1]

Alice:	1	5	6
Bob:	4	6	3

Alice will choose 1 firstly, because if Alice choose 3, bob will get 6 which is larger than 4.

$4+6+3=13 > 2+6+4$

∴ Greedy-denial method can't help Alice win this game.

7. [8] Exercise R-10.1 from the book. Show your work.

   R-10.1 Let S = {a, b, c, d, e, f, g} be a collection of objects with benefit-weight values, a: (12, 4), b: (10, 6), c: (8, 5), d: (11, 7), e: (14, 3), f : (7, 1), g: (9, 6). What is an optimal solution to the fractional knapsack problem for S assuming we have a sack that can hold objects with total weight 18? Show your work.

Va = 3 benefit/weight Vb = 5/3 ≈ 1.67 Vc = 8/5 = 1.6 Vd = 11/7 ≈ 1.57 Ve = 14/3 ≈ 4.67 Vf = 7 Vg = 3/2 = 1.5

we can get that Vf > Ve > Va > Vb > Vc > Vd > Vg

W = 18 weight B is total benefit, we set B = 0 at first.

Therefore, we load items with the highest value per unit weight in order until the knapsack is full.

Thus f is the first.

W - wf = 18 - 1 = 17

B = 0 + bf = 0 + 7 = 7

And then e is the second.

W - we = 17 - 3 = 14

B = B + be = 7 + 14 = 21

And then a is the third.

W - wa = 14 - 4 = 10

B = B + ba = 21 + 12 = 33

And then b is the forth.

W - wb = 10 - 6 = 4

B = B + bb = 33 + 10 = 43

And then c is the final.

W = 4 < wc = 5

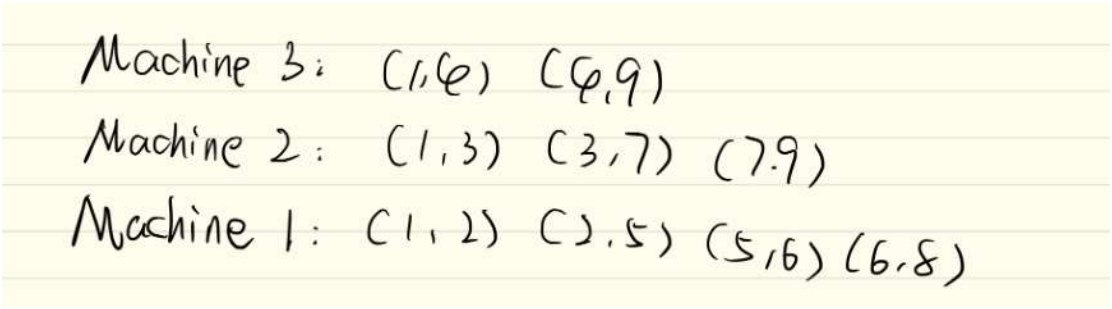
so we can just load 4 weight c into knapsack.

B = B + 4 * Vc = 43 + 4 * 1.6 = 49.4

In conclusion, we use all knapsack's weight 18, and get the maximum benefit 49.4.

8. [8] Exercise R-10.3 from the book. Show your work.

R-10.3 Suppose we are given a set of tasks specified by pairs of the start times and finish times as T = {(1, 2), (1, 3), (1, 4), (2, 5), (3, 7), (4, 9), (5, 6), (6, 8), (7, 9)}. Solve the task scheduling problem for this set of tasks.



9. [8] Exercise C-10.2 from the book.

R-10.2 Describe how to implement the TaskSchedule method to run in O(n log n) time.

The TaskSchedule algorithm is designed to schedule a set of n tasks with their start and finish times on machines in a way that minimizes the number of machines required. The algorithm works by sorting the tasks based on their start times and then iteratively assigning tasks to machines in a way that avoids conflicts.

The time complexity of the algorithm primarily comes from the sorting step. When you sort the n tasks by their start times, you are essentially performing a comparison-based sort operation, which typically has a time complexity of O(n log n) in the worst case for algorithms like quicksort or mergesort. This sorting step dominates the time complexity of the algorithm.

After sorting, the algorithm iterates through the sorted list of tasks once to assign them to machines, which can be done in linear time, O(n).

So, the overall time complexity of TaskSchedule is dominated by the sorting step, which is O(n log n). This is why it is stated that TaskSchedule produces a schedule in O(n log n) time.