CS430 HW 5 part 2.
Junzle Zheng
A20254389

Problem 4.

(a) Algorithm: Sort the tasks a_i in an increasing order by their processing time p_i. This gives the minimized average ~~time~~ completion time.

Proof: For simplify, assume $p_i < p_j$ if $i < j$,

By Given by algorithm, the processing sequence is:

$$a_1, a_2, a_3 \cdots a_{i-1} a_i \overset{a_{i+1}}{\text{(i)}} \cdots a_{j-1} a_j \overset{a_{j+1}}{} \cdots a_n \quad (i < j)$$

So $\sum_{k=1}^{n} C_k = P_1 + P_2 \cdots + P_i + \cdots P_j + \cdots P_n$

$$C_k = \sum_{l=1}^{k} P_l \quad 1 \leq l \leq n.$$

$$C_i = \sum_{l=1}^{i} P_l \qquad C_j = C_i + \sum_{l=i+1}^{j} P_l = C_i + \sum_{l=i+1}^{j-1} P_l + P_j$$

$$= \sum_{l=1}^{i-1} P_l + P_i$$

Switch a_i, a_j

$$a_1, a_2, a_3 - a_{i-1} a_j \overset{a_{i+1}}{} \cdots a_{j-1} a_i \cdots a_n$$

$$C_i' = \sum_{l=1}^{i-1} P_l + P_j + \sum_{l=i+1}^{j-1} P_l + P_i$$

$$C_j' = \sum_{l=1}^{i-1} P_l + P_j$$

$$C_i' + C_j' = \sum_{l=1}^{i-1} P_l + P_j + \sum_{l=i+1}^{j-1} P_l + P_i + \sum_{l=1}^{i-1} P_l + P_j$$

$$= \underbrace{(\sum_{l=1}^{i-1} P_l + P_i)}_{C_i} + \underbrace{(\sum_{l=1}^{i-1} P_l + P_i + \sum_{l=i+1}^{j-1} P_l + P_j)}_{C_j} + P_j - P_i$$

①

② $$C_i' + C_j' = C_i + C_j + (P_j - P_i)$$

$$\geq C_i + C_j \qquad (\text{For } i \leq j \quad P_i \leq P_j)$$

~~Thus~~ Thus, for $i \Rightarrow P_i < P_j$ ⊘i

We have $C_i + C_j$ less than any ~~ovether sb~~ other order. This algorithm gives the minimized average completion ~~ft~~ time.

For running time, ~~the~~ it should be seperated into two parts.

Part 1. Sorting: we can use quicksort, which takes $O(n \log n)$

Part 2. Processing: it takes $O(n)$.

Thus time complexity is $O(n \log n)$.

(b) As was proven in the previous one, the average completion time is minimized when all tasks are scheduled in an ascending time by their processing time.

1. We sort r_i's in an ascending order.

2. Then launch r_i by its order continuisly, means when r_i is finished, we launch next r_{i+1} ~~after~~ r_i immediately. Meanwhile, ~~we~~ we processes a_i when r_i is done.

3. If r_{i+1} is done but a_i is still running, we stop a_i, put the ~~remained~~ remained a_i to the sorted task queue (order by P_i) in ascending order), ~~take ou~~ take out the task which requires the minimum processing time from ~~tok~~ task queue and process it.

②

Junzle Zheng
A20254387

If a_i is done when r_{i+1} is still runing, we wait till r_{i+1} is done, then prosess a_{i+1} and launch r_{i+2}.

The key idea is to run the process in an ascending order by processing time and remaining processing time.

Time Complexity.

The initial sort will run in $O(n \lg n)$.

For each new task arrival,

worst: $O(n)$ best $O(1)$

We have n task arrival, thus

worst $O(n^2)$;   best $O(n)$

Thus, running the time of this algorithm in worst case is.

$$O(n \lg n) + O(n^2) = O(n^2)$$

Best case is: $O(n \lg n) + O(n) = O(n \lg n)$.

③

Problem 5.

(a.) ① First of all, I'd like to apply this algorithm
to section 16.5 problem as follows:

Task.

a_i	1	2	3	4	5	6	7.
d_i	4	2	4	3	1	4	6.
w_i	70	60	50	40	30	20	10

← sorted by penalty weight in descending order.

Processing Order (slots list)

Slot

S	1	2	3	4	5	6	7.
a_i	a_4	a_2	a_3	a_1			
w_i	40	60	50	70			

First, take out a_1, which $d_1 = 4$, we search in slots. list
Slot4 is unoccupied, thus we insert a_1 at S4.
Then we look at a_2. $d_2 = 2$, we search in slots list,
Slot2 is unoccupied, thus we insert a_2 at S_2.
Then we look at a_3, $d_3 = 4$ we search in slots list,
slot4 is occupied, but there are slots before Slot4 unoccupied (slot 1 and
slot3), we insert a_3 at latest slot. which is S_3.
Similiar, we insert a_4 at S_1.

Then we look at a_5, $d_5 = 1$, searching the slots list. S_1 is o occupied, and there is no empty slot be for before S_1. which means S_2 ~~and~~ a_5 will suffer a penalty. $w_5 = 30$. In this situation, I prefer ~~don~~ do not insert a_5 to ~~so~~ slots list yet. but to another list which ~~so~~ named penalty list.

Processing Order (slots list)						
S	1	2	3	4	5	6	7
a_i	a_4	a_2	a_3	a_4		a_7	
w_i	40	50	60	70		10	

Penalty list.						
P	1	2	3	4	5	6	7
a_i	a_i	a_5	a_6				
w_i	30	20					

Similar, we insert a_6 to Penalty list ~~@~~ P_2.

Last, we insert a_7 to S_6.

Now. we note that S_5 is unoccupied. then we can ~~wo~~ move a_7 to S_5.

Processing Order (slots. list)						
S	1	2	3	4	5	6	7
a_i	a_2	a_4	a_2	a_3	a_1	a_7	a_5	a_6
w_i	40	50	60	70	10	30	20

$\rightarrow$ penalty $= 30 + 20 = 50$

Finally. we move all items from P_2 to S, ~~filled~~ to fill all empty slots.

This algorithm always ~~deal~~ deal with a_i with most penalty at first. and put it at last possible slot. $\&$ leave slots with ~~so~~ before this possible slot to other a_i 0 in case they ~~required~~ require a ~~early~~ earlier deadline time.

Thus. ~~we~~ this algorithm can guarantee ~~that~~ that we can insert as much as we can $\&$ and all the ~~tasks~~ with a higher penalty, which leads to a less penalty.

Problem 5.

(b) Algorithms.

Step 1. ① Initialization

Each position $0, 1, 2 \ldots n$ is different set and

$$F(\{i\}) = i, \quad 0 \le i \le n. \quad \{i\} \text{ is a set.}$$

a Job List. J. $J[i] = 0, 0 \le i \le n$.

Step 2. ② For Start from first task in the ordered task list,
if assume this task with a deadline d.

Find the set that contains d, for assuming it is the
set k, 1. assign this take task to $J[F(k)]$

2. Find the set that contains $F(k) - 1$. $O \to$ call it
set L.

3. Union set k and L, $F(\text{Union}(k, L)) = F(L)$.
(There is no set k and any to after merge).

repeat & Step 2. till last task.

Step 3. return J

Time Complexity. There are at most $2n$ Find operations, n,
make set operations and $n-1$ union operations.
Thus $O it$ is $O(n)$

CS430 HW5 Part 2.
Junxle Zheng
A20254438P

Problem 6.

(a) Prove by contradiction.

Assuming A and B are both MST of a graph G.

assuming A has an edge e_1 with least weight. but B do does have not.

Add e_1 to B. then B must have a cycle C that contains edge e_1.

Because A, B are MST of G, e_1 is also the edge with least weight in B. In addition, C in B has has an edge e_2 with a weight greater than e_1.

Removing e_2 will generate a new B has a less weight than original B and A.

But at the begining we assumed A and B are both MSTs of G.

There is a contradiction. Thus. MST of G is unique.

④

(b) Let's consider most exertrem situation. all edges have a same weight, obeviously ~~their all~~ there are many possible MSTs.

If e_1 and e_2 have a same weight and both of them ~~a~~ are in a cycle. We can use any one of them. ~~ther~~ this will generate two different MSTs

If e_1 and e_2 have a same weight and not ~~a~~ in a cycle, we choose e_1 or e_2 or both of them ~~a~~ ~~a~~ ~~utess~~ unless ~~if~~ ~~there~~ no cycle will be generated. ~~Wether~~ a different MSTs will be generated or ~~not~~ can not be decided.

Thus, ~~But~~ multiple spanning trees can be generated, but not ~~not~~ always.

We can use Kruskal's algorithm. ~~Wa~~ When we ~~forgot~~ get multiple possible edges with a same weight. We can pick ~~any~~ ~~onb~~ any one of them ~~w~~ ~~a~~ as long as picking it ~~want~~ won't violate any algrothm and MST rules.

Problem 7.

~~Let k_1 = Number of 25c.~~

~~k_2 =~~

To use minimun number of coins. we just ~~have to~~ need to use larger value coin as much as possible.

Let k_1, k_2, k_3, k_4 donate the number of 25c, 10c, 5c, 1c.

$$k_1 = \lfloor v/25 \rfloor$$

~~$k_2 = v - \lfloor v/25 \rfloor$~~

$$k_2 = \lfloor (v - k_1 \cdot 25)/10 \rfloor$$

$$k_3 = \lfloor (v - k_1 \cdot 25 - k_2 \cdot 10)/5 \rfloor$$

$$k_4 = \lfloor (v - k_1 \cdot 25 - k_2 \cdot 10 - k_3 \cdot 5)/1 \rfloor$$

Pseudocode:

```
Change (V. d)             d is array contains denominations
{
    for i=1 to d.size)

    change [ ];           declare array to hold number of changes.
    for (i=1 to change.size)        for each denomination.
        change [i] = 0;
    for (i=1 to d.size)
    { if V > 0
        { change[i] = V / d[i];
          V = V % d[i]
        }
    }                     ↳ backpage
```

(P6)

```
        else
            break;
    }
    return change[ ];
}
```