5. (1) arrange: (2, 13) (5, 5) (3, 4) (7, 3) (4, 2)

customer 1: $2+13=15$ 2: $7+5=12$

3: $2+5+3+4=14$ 4: $17+3=20$

5: $21+2 = 23$

mini time needed: 23

(2) algorithm:

① sort customers by their eating time 's descending order $O(n\log n)$

② compute mini time

```
sum = 0 ;   min = 0
for (p_i, e_i) in customer_list:
    sum += p_i
    if ((sum + e_i) > min):
        min = sum + e_i
```

$\Big\}$ $O(n)$.

total time complexity is $O(n\log n)$

(3).

For adjacent customer:

......		
i	p_i	e_i
$i+1$	p_{i+1}	e_{i+1}
......		

the order of i, $i+1$ don't affect the Maximum of their front customers and their behind customers.

thus: we assumed:

front customers' maximum: Max_front

behind customers' maximum: Max_behind

thus: result = max (Max_front, Max_behind, Max_i,i+1)

for i: $T_i = \sum_{k-1}^{i-1} p_k + p_i + e_i$

for i+1: $T_{i+1} = \sum_{k=1}^{i} p_k + p_i + p_{i+1} + e_{i+1}$

when exchange i, i+1:

....

i	p_{i+1}	e_{i+1}
$i+1$	p_i	e_i

$T_i' = \sum_{k=1}^{i-1} p_k + p_{i+1} + e_{i+1}$

$T_{i+1}' = \sum_{k=1}^{i-1} p_k + p_{i+1} + p_i + e_i$

....

$T_{i+1}' - T_i = p_{i-1} > 0 \qquad T_{i+1}' > T_i$

$T_{i+1} - T_i' = p_i > 0 \qquad T_{i+1} > T_i'$

just need to compare T'_{i+1} T_{i+1}

$T'_{i+1} - T_{i+1} = e_i - e_{i+1}$ while: $e_i > e_{i+1}$ $T'_{i+1} > T_{i+1}$

choose T_{i+1}

while $e_i < e_{i+1}$ $T'_{i+1} < T_{i+1}$ choose T_{i+1}'

thus: we just need to choose $e_i > e_{i+1}$

thus: best arrangement:

choose customers by their eating time's descending order

(4) disprove, counterexample:

as (2):

100	1
1	100
5	5

00	01
1 100	5 5
100 1	

max = 1+100+1 = 102

better:

00	01
1 100	100 1
5 5	

max = 101

(5). ex: (2,13) (5,5) (3.4) (7,3) (4,2)

the arrangement is same with (2) $O(N\log N)$.

① compute the time for each one, storing in an array:

arr: [15, 20, 14, 20, 23] $O(N)$

② compute left_max_arr, right_max_arr.

left_max_arr [0, 15, 20, 20, 20]

right_max_arr [23, 23, 23, 23, 0] $O(N)$

③ min = Max for i → delete i:

left_max don't change

new_right_max = right_max − p[i] $O(N)$.

result = max(left_max, arr[i], new_right_max)

if result < min

min = result.

return min.

time complexity: $O(N\log N)$.

6. (1) 3 → at 4 (1, 7) is ok

2 → at 12 (11, 12) is ok

5 → at 22 (17) is ok.

(2) algorithm:

```
i = 0    cnt = 0    num = len(d)
while (i < len):
    // give a diner at xi + d.
    left = xi - d        right = xi + d
    cnt ++ ;   i++ ;   if (cnt > num): return error.
    // find next i uncovered
```

```
        while ( i<len && left≤ xi ≤ right):
            i++;
    return cnt                          O(N+m)
```

correctness as (3)

(3) algorithm:

```
    i=0    cnt=0    num = len(d)
    while (i < len):
        //give a diner at xi+dcnt
        left = xi - dcnt    right = xi + dcnt
        cnt++;    i++;    if (i<len && cnt>num) return error
        //find next is uncovered.
        while (i< len && left ≤ xi ≤ right)
            i++;
    return cnt                          O(N+M)
```

prove: for x_i uncovered. and d_{cnt} to be used.

when put d_{cnt} at $x_i + d_{cnt}$ location:

we can cover $x_i \sim x_i + 2d_{cnt} + 1$

when we put d_{cnt} at $x_i + d_{cnt}$ left.

right boundary < $x_i + 2d_{cnt} + 1$. although left boundary changes, it's useless.

when we put d_{cnt} at $x_i + d_{cnt}$ right

it can't cover x_i.

thus: this greedy algorithm is correct.

(4)

dp [i] [j] means for 0 ~ i classes and 0 ~ j diners, the minimum number of mobile diners. -1 means it can't cover all classes.

$$dp[i][j] \begin{cases} dp[i][j-1] \\ 1 + dp[i] \end{cases}$$

7. (1).

$$dp[i][j] = \begin{cases} \end{cases}$$