13924 - Table Management System

I2P(II) Final Practice

Description

- Hodilo (海底撈?) is a well-known hot pot (火鍋) restaurant, and it's extremely challenging to have a table during peak dining hours.
- Even though, the restaurant does not accept reservations in advance, requiring every guest to visit the restaurant, take a number, and wait for their turn.
- Hodilo is opening a new bra design a queuing system, w guests.

Description

Design a table management system for the restaurant.

Given information including:

Arrival record of each guest

- Arrival Time
- Group Size
- Dining Duration

Number of the tables for each size

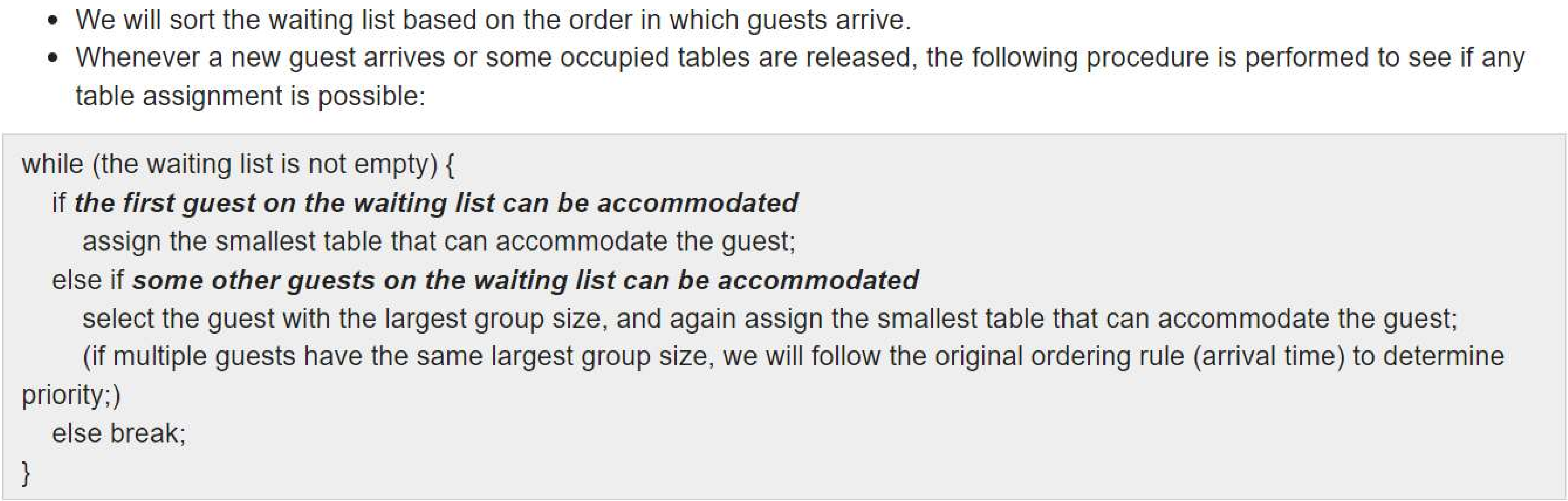
- Tables for 4 ×20
- Tables for 6 ×20
- …

Assign a table to each guest, and

provide an estimated waiting time for their table.

Description

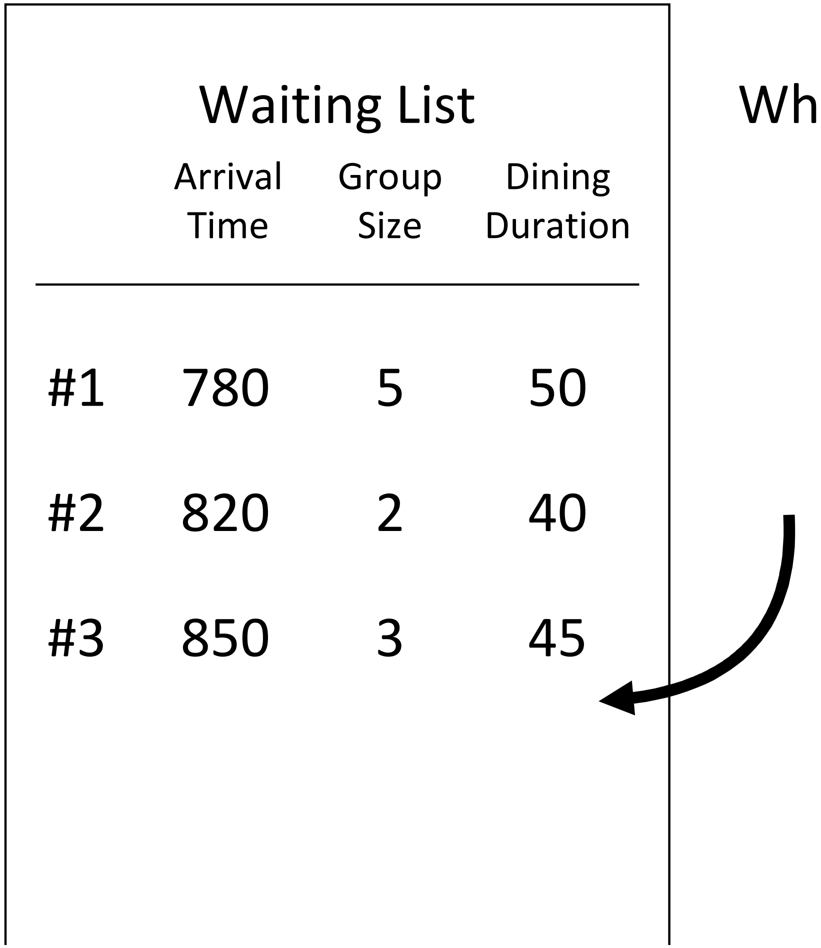
- Each guest's arrival time is unique. We will add them to the waiting list one by one and subsequently assign tables to the guests on the waiting list.



- We will sort the waiting list based on the order in which guests arrive.
- Whenever a new guest arrives or some occupied tables are released, the following procedure is performed to see if any table assignment is possible:

```
while (the waiting list is not empty) {
   if the first guest on the waiting list can be accommodated
      assign the smallest table that can accommodate the guest;
   else if some other guests on the waiting list can be accommodated
      select the guest with the largest group size, and again assign the smallest table that can accommodate the guest;
      (if multiple guests have the same largest group size, we will follow the original ordering rule (arrival time) to determine
priority;)
   else break;
}
```

Description



Waiting List

	Arrival Time	Group Size	Dining Duration
#1	780	5	50
#2	820	2	40
#3	850	3	45

When a guest arrives, add them to the list

Description

Waiting List

	Arrival Time	Group Size	Dining Duration
#1	780	5	50
#2	820	2	40
#3	850	3	45

When a guest arrives, add them to the list

Whenever a new guest arrives or some occupied tables are released,

- The 1st guest on the waiting list can be accommodated.
  Assign the smallest table to them!
- Some other guests on the waiting list can be accommodated then, select the guest with the largest group size
  Assign the smallest table to them!

Description

Waiting List

	Arrival Time	Group Size	Dining Duration
#1	780	5	50
#2	820	2	40
#3	850	3	45

When a guest arrives, add them to the list

Whenever a new guest arrives or some occupied tables are released,

- The 1st guest on the waiting list can be accommodated. Assign the smallest table to them!
- Some other guests on the waiting list can be accommodated then, select the guest with the largest group size Assign the smallest table to them!

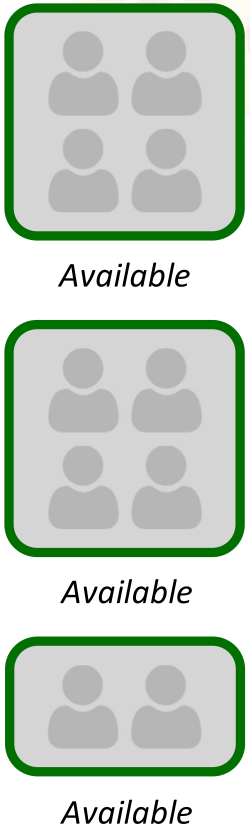
Sample I/O

```
6 2
780 1 75
820 2 40
830 3 30
840 4 100
845 1 60
850 2 65
2 1
4 2
```

arrival timestamp	group size	dining duration	answer

Current Time

0



Sample I/O

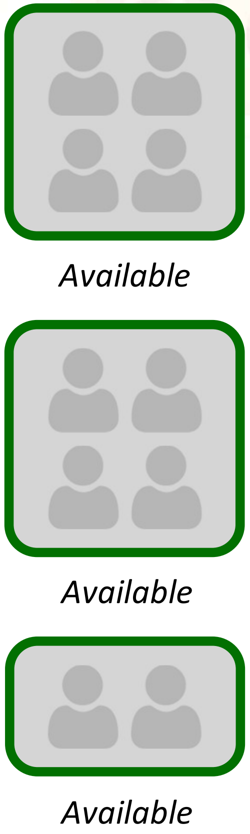
```
6 2
780 1 75
820 2 40
830 3 30
840 4 100
845 1 60
850 2 65
2 1
4 2
```

← Add to waiting list

arrival timestamp	group size	dining duration	answer
780	1	75	780

Current Time

780



Sample I/O

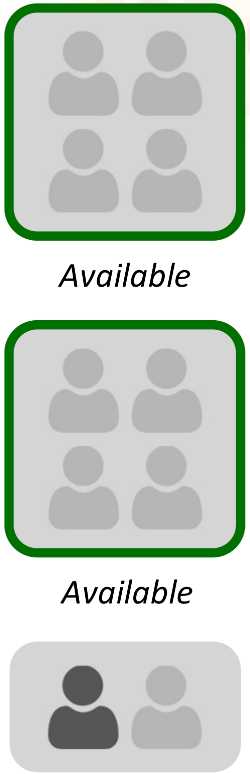
Assigning the smallest table

```
6 2
780 1 75
820 2 40
830 3 30
840 4 100
845 1 60
850 2 65
2 1
4 2
```

arrival timestamp	group size	dining duration	answer
780	1	75	780

Current Time

780



Available

Available

780~855

Sample I/O

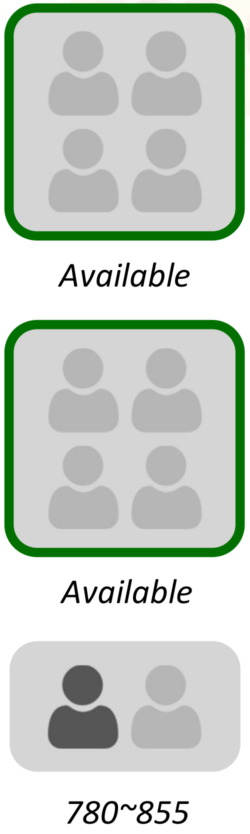
```
6 2
780 1 75
820 2 40
830 3 30
840 4 100
845 1 60
850 2 65
2 1
4 2
```

← Add to waiting list

arrival timestamp	group size	dining duration	answer
780	1	75	780
820	2	40	

Current Time

820



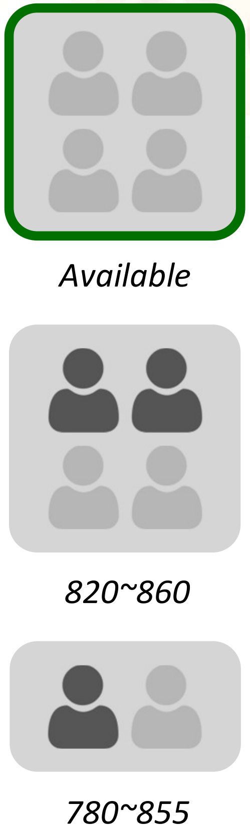
Sample I/O

```
6 2
780 1 75
820 2 40
830 3 30
840 4 100
845 1 60
850 2 65
2 1
4 2
```

arrival timestamp	group size	dining duration	answer
780	1	75	780
820	2	40	820

Current Time

820



Sample I/O

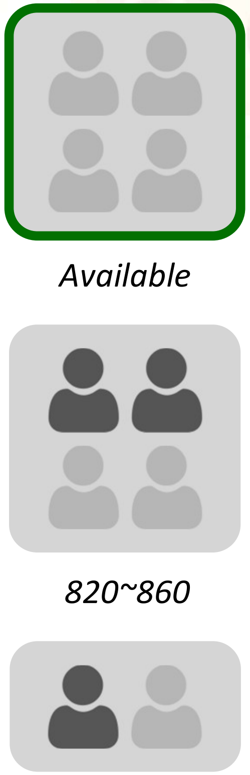
```
6 2
780 1 75
820 2 40
830 3 30
840 4 100
845 1 60
850 2 65
2 1
4 2
```

← Add to waiting list

arrival timestamp	group size	dining duration	answer
780	1	75	780
820	2	40	820
830	3	30	

Current Time

830



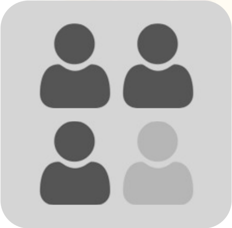
Sample I/O

```
6 2
780 1 75
820 2 40
830 3 30
840 4 100
845 1 60
850 2 65
2 1
4 2
```

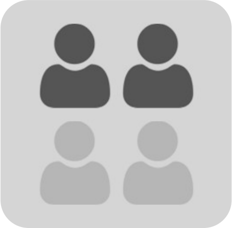
arrival timestamp	group size	dining duration	answer
780	1	75	780
820	2	40	820
830	3	30	830

Current Time

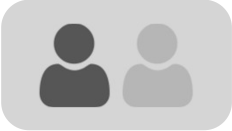
830



830~860



820~860



780~855

Sample I/O

No available table

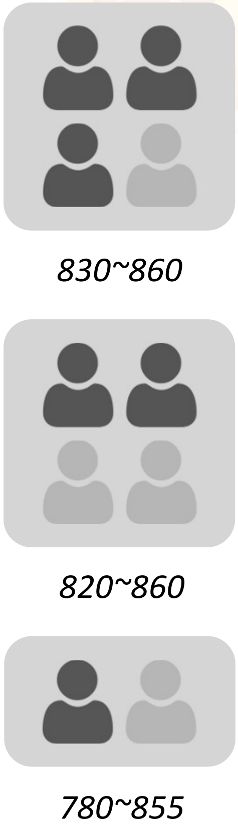
```
6 2
780 1 75
820 2 40
830 3 30
840 4 100
845 1 60
850 2 65
2 1
4 2
```

← Add to waiting list

arrival timestamp	group size	dining duration	answer
780	1	75	780
820	2	40	820
830	3	30	830
840	4	100	

Current Time

840



Sample I/O

No available table

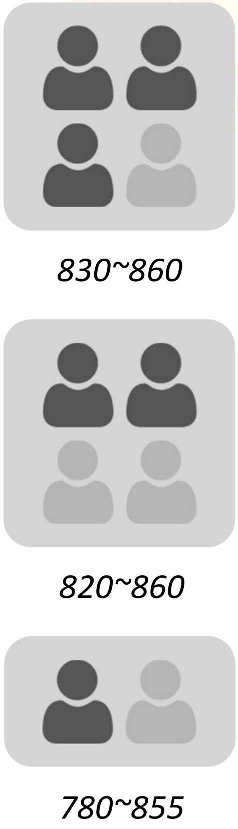
```
6 2
780 1 75
820 2 40
830 3 30
840 4 100
845 1 60
850 2 65
2 1
4 2
```

← Add to waiting list

arrival timestamp	group size	dining duration	answer
780	1	75	780
820	2	40	820
830	3	30	830
840	4	100	
845	1	60	

Current Time

845



Sample I/O

No available table

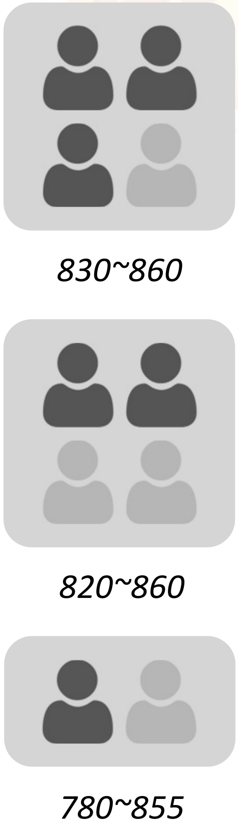
```
6 2
780 1 75
820 2 40
830 3 30
840 4 100
845 1 60
850 2 65
2 1
4 2
```

← Add to waiting list

arrival timestamp	group size	dining duration	answer
780	1	75	780
820	2	40	820
830	3	30	830
840	4	100	
845	1	60	
850	2	65	

Current Time

850



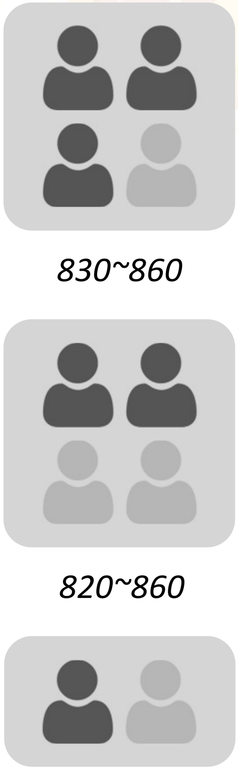
Sample I/O

```
6 2
780 1 75
820 2 40
830 3 30
840 4 100
845 1 60
850 2 65
2 1
4 2
```

arrival timestamp	group size	dining duration	answer
780	1	75	780
820	2	40	820
830	3	30	830
840	4	100	
845	1	60	
850	2	65	

Current Time

855



830~860

820~860

780~855

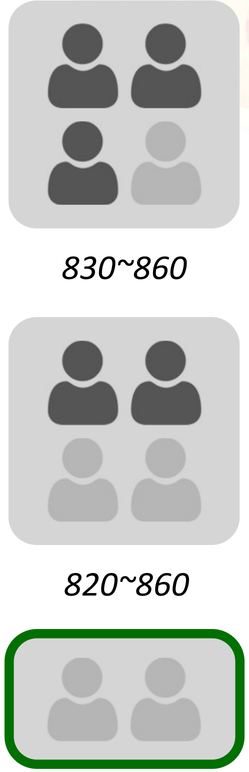
Sample I/O

```
6 2
780 1 75
820 2 40
830 3 30
840 4 100
845 1 60
850 2 65
2 1
4 2
```

arrival timestamp	group size	dining duration	answer
780	1	75	780
820	2	40	820
830	3	30	830
840	4	100	
845	1	60	
850	2	65	

Current Time

855



830~860

820~860

Available

Sample I/O

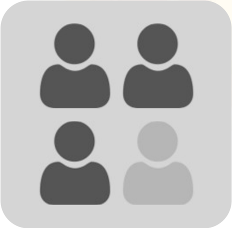
Select the guest
with the largest group size

```
6 2
780 1 75
820 2 40
830 3 30
840 4 100
845 1 60
850 2 65
2 1
4 2
```

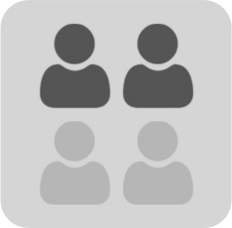
arrival timestamp	group size	dining duration	answer
780	1	75	780
820	2	40	820
830	3	30	830
840	4	100	
845	1	60	
850	2	65	855

Current Time

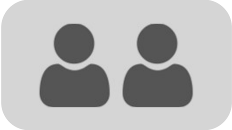
855



830~860



820~860



855~920

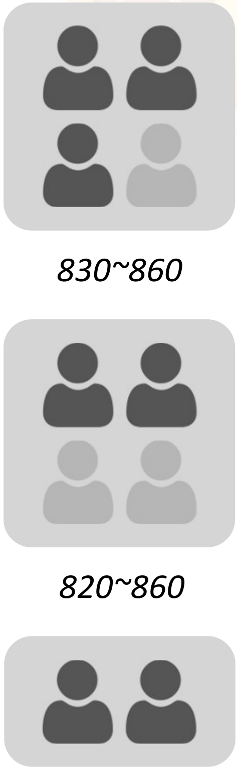
Sample I/O

```
6 2
780 1 75
820 2 40
830 3 30
840 4 100
845 1 60
850 2 65
2 1
4 2
```

arrival timestamp	group size	dining duration	answer
780	1	75	780
820	2	40	820
830	3	30	830
840	4	100	
845	1	60	
850	2	65	855

Current Time

860



830~860

820~860

855~920

Sample I/O

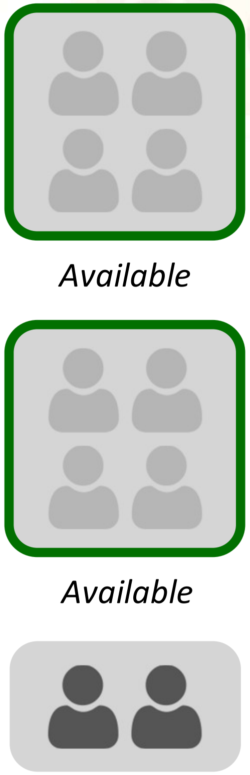
Tables may be released at the same time!

```
6 2
780 1 75
820 2 40
830 3 30
840 4 100
845 1 60
850 2 65
2 1
4 2
```

arrival timestamp	group size	dining duration	answer
780	1	75	780
820	2	40	820
830	3	30	830
840	4	100	
845	1	60	
850	2	65	855

Current Time

860



Available

Available

855~920

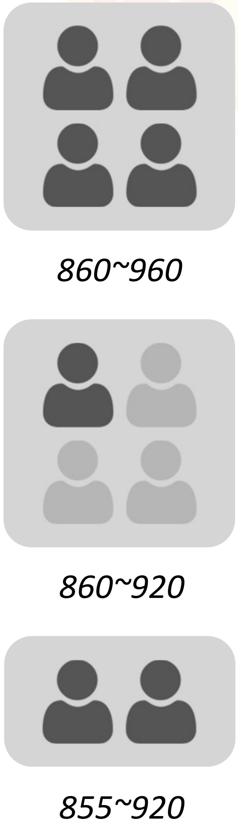
Sample I/O

```
6 2
780 1 75
820 2 40
830 3 30
840 4 100
845 1 60
850 2 65
2 1
4 2
```

arrival timestamp	group size	dining duration	answer
780	1	75	780
820	2	40	820
830	3	30	830
840	4	100	860
845	1	60	860
850	2	65	855

Current Time

860



Idea

- Maintain the Waiting List
- Maintain the Table Status
- Release & Assign the Table
- Solving the Problem

Idea: Maintain the Waiting List

Using structure to store a guest's info, and
std::set to implement the waiting list

```
struct Guest {
    int id;
    int arrival;
    int group;
    int duration;
};
```

```
set<Guest> waiting_list;
// May Compile Error
```

Idea: Maintain the Waiting List

std::set with custom comparator   Reference↗

Approach 1   Functor

```
struct cmp {
   bool operator() (Guest a, Guest b) const {
      return a.arrival < b.arrival;
   }
};

// sort by arrival time
set<Guest, cmp> waiting_list;
```

Idea: Maintain the Waiting List

std::set with custom comparator   Reference↗

Approach 2   Lambda Function (C++11)

```
auto cmp = [](Guest a, Guest b) {
   return a.arrival < b.arrival;
};

// sort by arrival time
set<Guest, decltype(cmp)> waiting_list(cmp);
```

Idea: Maintain the Waiting List

std::set with custom comparator   Reference↗

Approach 3   Lambda Function (C++20)

```
auto cmp = [](Guest a, Guest b) {
   return a.arrival < b.arrival;
};

// sort by arrival time
set<Guest, decltype(cmp)> waiting_list;
```

Idea: Maintain the Waiting List

How can we select the 1st guest on the list?

```
// sort by arrival time
set<Guest, decltype(cmp)> waiting_list(cmp);

// 1st guest on the list
Guest guest = *waiting_list.begin();
```

Idea: Maintain the Waiting List

When the 1st guest can't be accommodated,
how can we select the guest with the largest group size?

Is one set not sufficient?   Use two sets instead!

Idea: Maintain the Waiting List

```
auto cmp_arrival = [](Guest a, Guest b) {
    return a.arrival < b.arrival;
};
auto cmp_group = [](Guest a, Guest b) {
    return a.group == b.group ? a.arrival > b.arrival : a.group < b.group;
};

// sort by arrival time
set<Guest, decltype(cmp_arrival)> waiting_arrival(cmp_arrival);

// sort by group size (small to large) -> arrival time (late -> early)
set<Guest, decltype(cmp_group)> waiting_group(cmp_group);
```

Idea: Maintain the Waiting List

```
// sort by group size (small to large) -> arrival time (late -> early)
set<Guest, decltype(cmp_group)> waiting_group(cmp_group);

auto it = waiting_group.upper_bound(Guest{0, 0, largest_size, 0});
if (it != waiting_group.begin())
    Guest guest = *(--it);
```

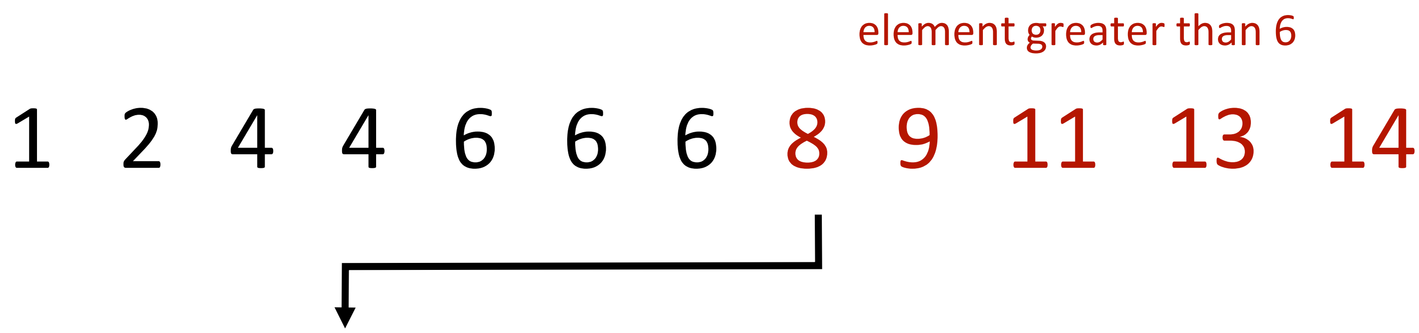
Upper Bound↗

Returns an iterator pointing to the first element that is greater than key.

Idea: Maintain the Waiting List

Upper Bound↗   Find the first element that is greater than key.

Suppose key = 6, to find the first element which’s greater than 6


element greater than 6

1   2   4   4   6   6   6   8   9   11   13   14

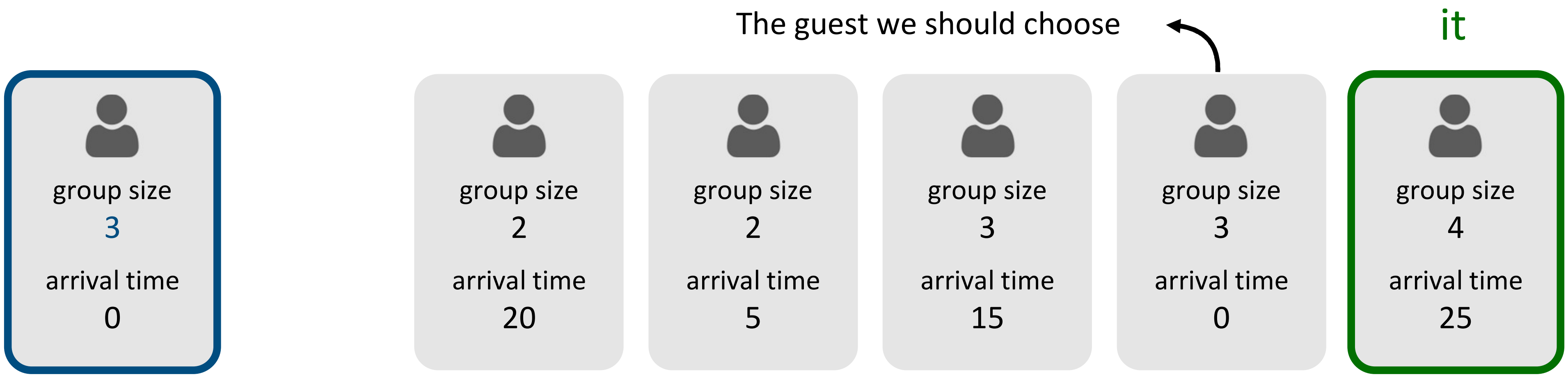
```
int x = *set.upper_bound(6);  // x = 8
```

Idea: Maintain the Waiting List

To find the guest with the largest group size and the earliest arrival time.

(suppose largest available table size is 3)

```
auto it = waiting_group.upper_bound(Guest{0, 0, 3, 0});
```



The guest we should choose

upper bound key

sort by group size (small -> large) -> arrival time (later -> earlier)

Idea

- Maintain the Waiting List
- Maintain the Table Status
- Release & Assign the Table
- Solving the Problem

Idea: Maintain the Table Status

Keep track of the available tables and the tables currently in use.

Available Tables

```
map<int, int> table_avl;   // {table size: number of tables}
```

Tables in use

```
multiset<pair<int, int>> table_use; // {release time, table size}
```

```
priority_queue<pair<int, int>> table_use;  // alternative
```

Priority Queue↗

Idea: Maintain the Table Status

Available Tables

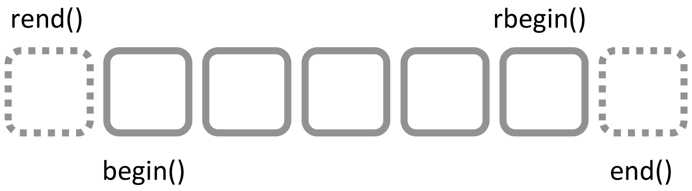
```
map<int, int> table_avl;   // {table size: number of tables}
```

To find the size of the largest available table

```
int largest_size = (*--table_avl.end()).first;
```

```
int largest_size = (*table_avl.rbegin()).first;
```

Reverse Iterator↗



Idea: Maintain the Table Status

Available Tables

```
map<int, int> table_avl;   // {table size: number of tables}
```

To find the smallest table that can accommodate a guest

```
auto table = table_avl.lower_bound(guest.group);
// iterator of pair element
```

Lower Bound↗

Returns an iterator pointing to the first element

that is not less than (i.e. greater or equal to) key.

Idea: Maintain the Table Status

Tables in use

```
multiset<pair<int, int>> table_use; // {release time, table size}
```

Simply insert the pair of a table's release time and size into the multiset

Since there might be simultaneous releases of two tables
with the same size, we need to use a multiset instead of a set.

Idea

- Maintain the Waiting List
- Maintain the Table Status
- Release & Assign the Table
- Solving the Problem

Idea: Release & Assign the Table

assignTable()

Do the table assignment procedure at the moment of “time”

```
bool assignTable(int time) {

    // no table or no one is waiting
    if (table_avl.empty() || waiting_arrival.empty()) return false;

    // table (iterator) and guest are about to be assigned
    Guest guest = *waiting_arrival.begin();
    auto table = table_avl.lower_bound(guest.group); // to find the smallest table that can accommodate the guest


    ...
```

Idea: Release & Assign the Table

assignTable() Do the table assignment procedure at the moment of “time”

```
bool assignTable(int time) {
   ...
   if (table == table_avl.end()) { // no such table

      int largest_size = (*table_avl.rbegin()).first; // largest table size

      // to find the largest-sized guest that can be accommodated
      auto it = waiting_group.upper_bound(Guest{0, 0, largest_size, 0});
      if (it == waiting_group.begin()) return false;
      guest = *(--it);

      // to find the smallest table that can accommodate the guest
      table = table_avl.lower_bound(guest.group);

   }
   ...
```

Idea: Release & Assign the Table

assignTable()   Do the table assignment procedure at the moment of “time”

```
bool assignTable(int time) {
    ...
    // handle table availability
    int table_size = table->first;
    table_avl[table_size]--;
    if (!table_avl[table_size]) table_avl.erase(table_size);
    table_use.insert({time+guest.duration, table_size});

    // record the answer and remove guest from the list
    ans[guest.id] = time;
    waiting_arrival.erase(guest); waiting_group.erase(guest);
    return true;

}
```

Idea: Release & Assign the Table

releaseTable()

Release all the tables before the moment of “time”

```
void releaseTable(int time) {
    while (table_use.size() && (*table_use.begin()).first <= time) {

        int release_time = (*table_use.begin()).first;
        table_avl[(*table_use.begin()).second]++;
        table_use.erase(table_use.begin());

        // table release simultaneously
        if (table_use.size() && ((*table_use.begin()).first) == release_time) continue;
        // table release simultaneously

        // attempt to assign table
        while (assignTable(release_time));
    }
}
```

Idea

- Maintain the Waiting List
- Maintain the Table Status
- Release & Assign the Table
- Solving the Problem

Idea: Now we have…

- Waiting list (sort by…)
  - arrival time
  - group size

waiting_arrival

waiting_group

- Table
  - Available tables
  - Tables in use

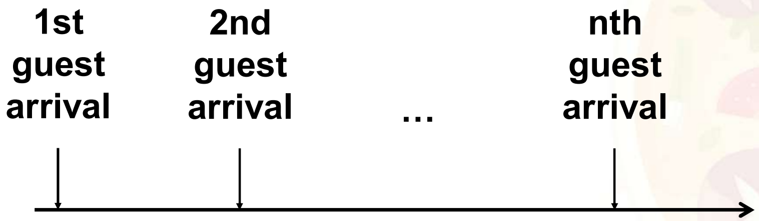
table_avl

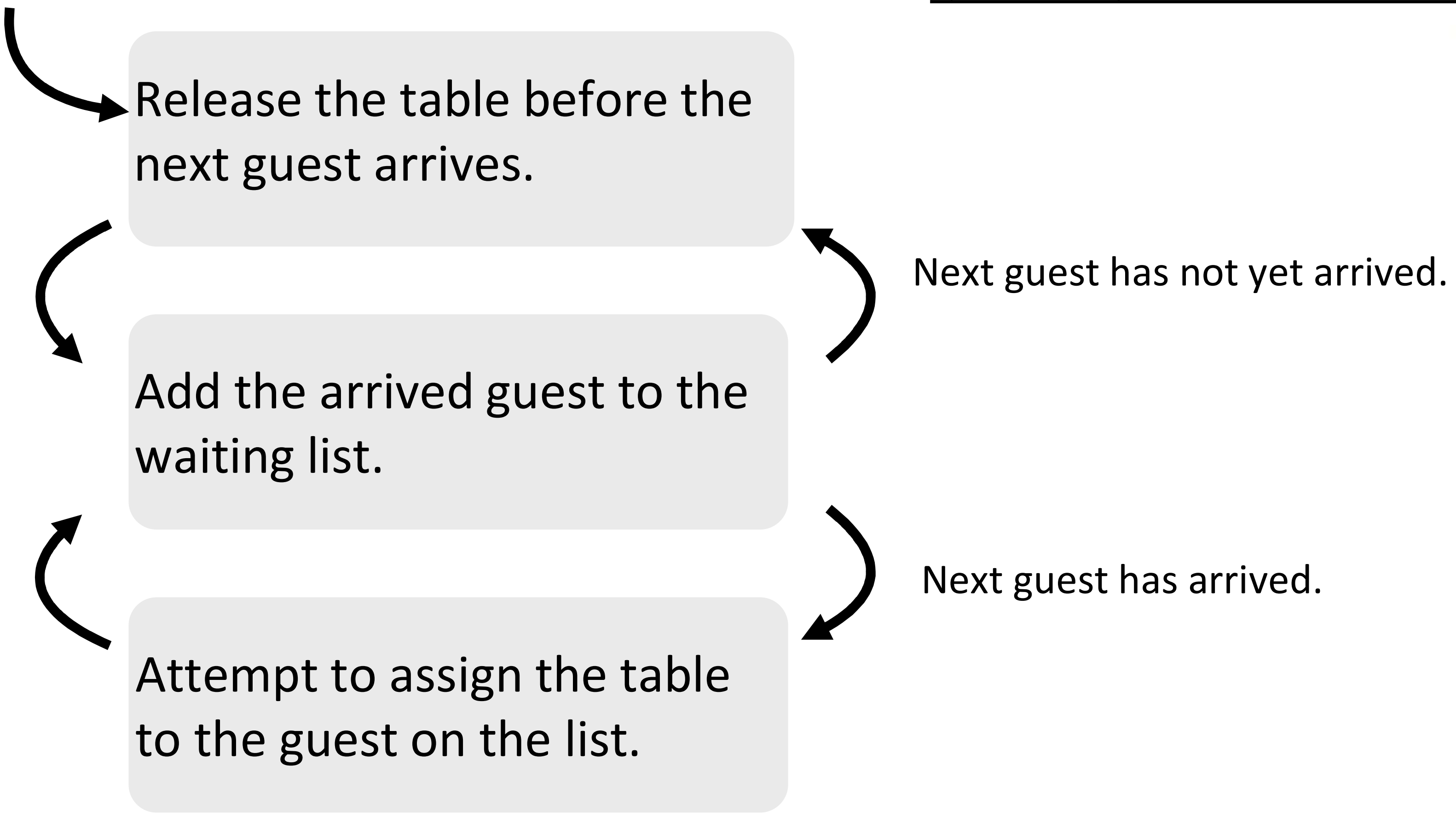
table_use

- Procedures
  - assignTable()

    releaseTable()

Idea: Solving the Problem





Release the table before the next guest arrives.

Add the arrived guest to the waiting list.

Attempt to assign the table to the guest on the list.

Next guest has not yet arrived.

Next guest has arrived.

Idea: Solving the Problem

main()

```
Guest arr[N];

for (int i=0; i<n; i++) {
    // clear and assign table before i-th arrival
    releaseTable(arr[i].arrival-1);

    // add to waiting list
    waiting_arrival.insert(arr[i]); waiting_group.insert(arr[i]);

    // clear all the table and try to assign table on i-th arrival
    releaseTable(arr[i].arrival); assignTable(arr[i].arrival);
}

// clear all the remaining table
if (waiting_arrival.size()) releaseTable(1e9);
```

Good Luck!

Let's become more familiar with these commonly used STLs through this problem!

I feel hungry again!!

Happy Summer Holidays~