S212 Super Chat

Problem idea by **Tony Wong**

Problem set by **Wai Ka Hei**, **Jeremy Chow**

30 January, 2021

Statistics

Task	Attempts	Max	Mean	Std Dev	Subtasks				
S212 - Super Chat	49	100	30.326	36.66	16: 31	15: 18	18: 13	27: 10	24: 9

First solved by **dbstoshinari123** at **0:45**

9 contestants got 100

Highest mean among senior problems

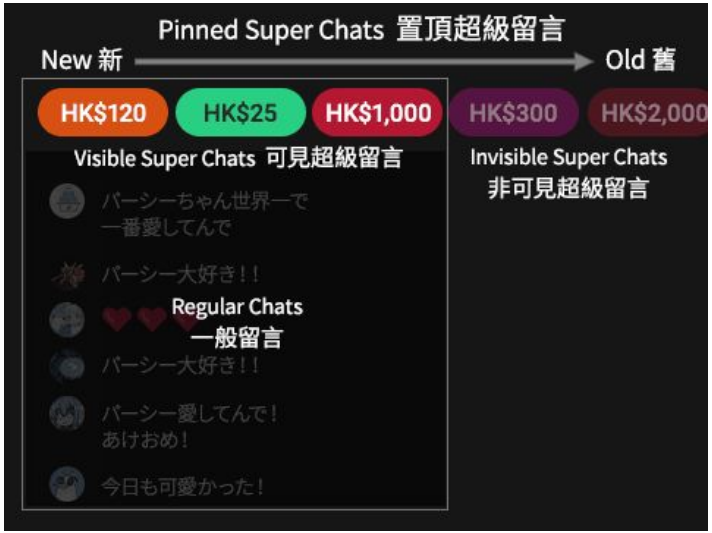
Easiest problem in senior problem set

Task

The Super Chat section can only display at most **3** Super Chats at a time

The **3** latest pinned Super Chats will be shown

Ordered by **purchase time**



HK$120, HK$25, HK$1000 are the 3 latest pinned Super Chats among all

Task

Each Super Chat is described by **purchase time T_i** and **price P_i**

Pin duration is determined by the table

Price	Colour	Pin duration and notes
$5 - $9	Blue	0 minutes. No chat message can be entered.
$10 - $24	Cyan	0 minutes
$25 - $49	Green	2 minutes
$50 - $99	Yellow	5 minutes
$100 - $249	Orange	10 minutes
$250 - $499	Magenta	30 minutes
$500 - $999	Red	1 hour
$1000 - $1499	Red	2 hours
$1500 - $1999	Red	3 hours
$2000 - $2499	Red	4 hours
$2500	Red	5 hours

Given **N** Super Chat **sorted by purchase time**

Find the number of **seconds** that each

Super Chat is **visible**

Sample 1

```
7
300
0 25
35 25
70 25
110 25
140 25
150 25
210 25
```

Time period when each Super Chat is visible:

Super Chat 1: 0 - 110

Super Chat 2: 35 - 140

Super Chat 3: 70 - 150

Super Chat 4: 110 - 210

Super Chat 5: 140 - 260

Super Chat 6: 150 - 270

Super Chat 7: 210 - 330 (The stream ended before the Super Chat expires)

Sample 2

```
4
4000
0 500
1 250
2 100
3 100
```

Time period when each Super Chat is visible:

Super Chat 1: 0 - 3, 602 - 3600

Super Chat 2: 1 - 1801

Super Chat 3: 2 - 602

Super Chat 4: 3 - 603

Sample 3

```
7
9000
60 2000
80 300
650 1000
820 5
930 25
1000 120
1590 50
```

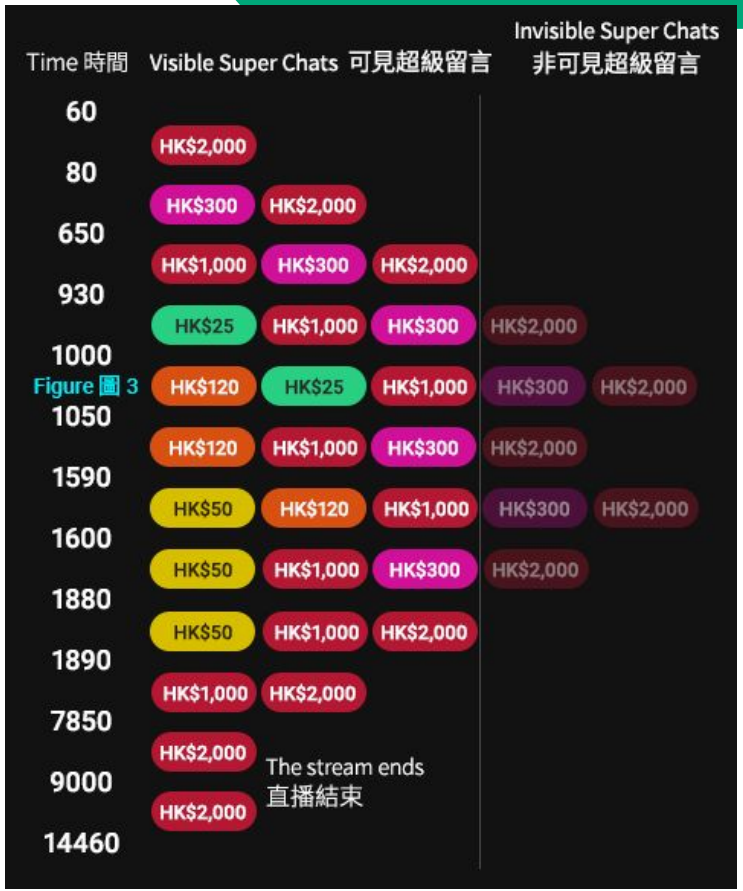


Figure 圖 3

Solutions

Ideas

Price is given instead of **Pin duration**

Write a **function** to convert Price into Pin duration

Price	Colour	Pin duration and notes
$5 - $9	Blue	0 minutes. No chat message can be entered.
$10 - $24	Cyan	0 minutes
$25 - $49	Green	2 minutes
$50 - $99	Yellow	5 minutes
$100 - $249	Orange	10 minutes
$250 - $499	Magenta	30 minutes
$500 - $999	Red	1 hour
$1000 - $1499	Red	2 hours
$1500 - $1999	Red	3 hours
$2000 - $2499	Red	4 hours
$2500	Red	5 hours

```
#define MIN 60
#define HR 3600
int getTime(int price) {
    if (price <= 24) return 0;
    if (price <= 49) return 2 * MIN;
    if (price <= 99) return 5 * MIN;
    if (price <= 249) return 10 * MIN;
    if (price <= 499) return 30 * MIN;
    if (price <= 999) return 1 * HR;
    if (price <= 1499) return 2 * HR;
    if (price <= 1999) return 3 * HR;
    if (price <= 2499) return 4 * HR;
    return 5 * HR;
}
```

Ideas

We can imagine Super Chats as **Segments** on a timeline

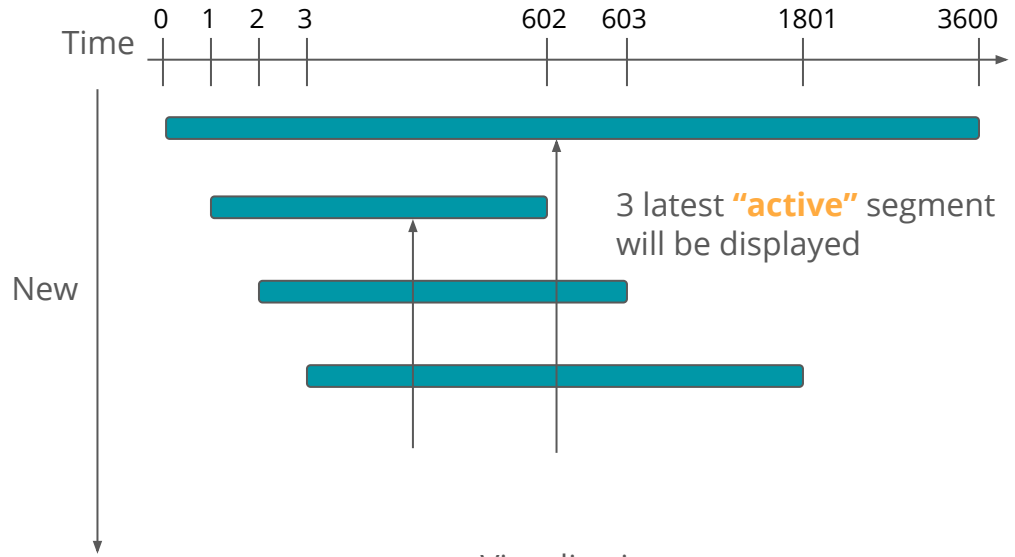
Start from purchase time T_i, end at T_i + **getTime(P_i)** // Pin duration

```
4
4000
0 500
1 250
2 100
3 100
```

→

```
4
4000
0 3600
1 1801
2 602
3 603
```

→



Visualization

Ideas

The **last 3** Super Chat among **N** chats will **always be visible** within their Pin duration

$(N-2)^{th}$, $(N-1)^{th}$, N^{th} Super Chat

No newer Super Chat can **"take"** their spot in the display section

Answer for them = their **Pin duration**

How about **1^{st}** to **$(N-3)^{th}$** Super Chat?

Subtask 1 (16 points)

Pi = 25, i.e. the pin duration of each and every Super Chat is **2 minutes**.

$1 \le N \le 200000$

$N \le K \le 500000$

Pin duration (getTime(P_i)) is the same for every **i**

If a superchat **start earlier** than another superchat, it **ends earlier** too

$T_i < T_j \Rightarrow T_i + \text{getTime}(P_i) < T_j + \text{getTime}(P_j)$

Subtask 1 (16 points)

If a Super Chat become **invisible**, it **won't become visible** again

Other visible Super Chats are **newer** => **end later**

If i^{th} Super Chat become **invisible** before its end time (overtaken by others SC), ...

Case 1

Case 2

Subtask 1 (16 points)

If i^{th} Super Chat become **invisible** before its end time

it must be overtaken by **$(i+3)^{th}$** Super Chat takes its spot

Compute the display time of **i^{th}** Super Chat by considering the difference between **i^{th}** and **$(i+3)^{th}$** Super Chat's **purchase time**

```
for (int i = 0; i < n; i++) {
    if (i + 3 < n) {
        int diff = a[i + 3].t - a[i].t;
        printf("%d\n", min(diff, 120));
    }
    else printf("%d\n", 120);
}
```

Time Complexity: **O(N)**

Subtask 2 (15 points)

N = 4

$4 \le K \le 20000$

~~If a Super Chat become **invisible**, it **won't become visible** again~~

```
4
4000
0 500
1 250
2 100
3 100
```

Time period when each Super Chat is visible:

Super Chat 1: 0 - 3, 602 - 3600

Super Chat 2: 1 - 1801

Super Chat 3: 2 - 602

Super Chat 4: 3 - 603

Subtask 2 (15 points)

N = 4

$(N-2)^{th}$, $(N-1)^{th}$, N^{th} Super Chat will **always be visible** within their Pin duration

How about the 1^{st} Super Chat?

1^{st} Super Chat is visible for at most **two** separate time periods

Subtask 2 (15 points)

Analyse carefully when 1st Super Chat is visible

One way is to consider when 1st Super Chat is blocked by 2nd, 3rd and 4th Super Chats

Let the end time $\mathbf{end_i}$ of i^{th} Super Chat be $\mathbf{T_i + getTime(P_i)}$

1st Super Chat is **possibly** blocked between $\mathbf{[T_4, \min(end_2, end_3, end_4)]}$

Subtask 2 (15 points)

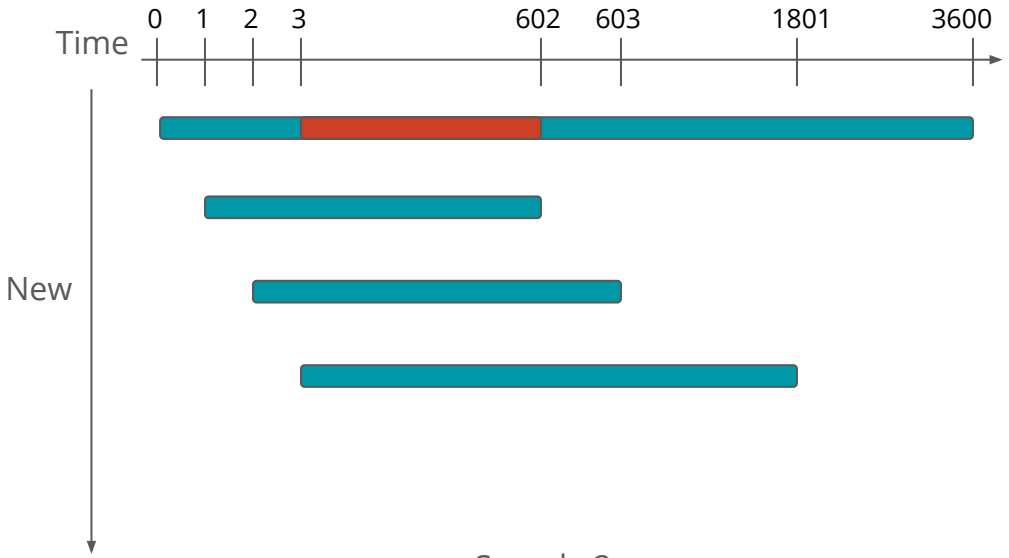
1st Super Chat is **possibly** blocked between **[T_4, min(end_2, end_3, end_4)]**

= [3, 602]

Display time of 1st Super Chat

= [0, 3] + [602, 3600]

= 3001



Sample 2

Subtask 2 (15 points)

Compute the display time of **1st** Super Chat carefully

Time Complexity = **O(1)**

```
pair <int, int> forbid = make_pair(a[3].t, min({a[1].ed, a[2].ed, a[3].ed}));
if (forbid.first < forbid.second) {
    int totalTime = 0;
    totalTime += min(getTime(a[0].p), forbid.first - a[0].t);
    totalTime += max(0, a[0].ed - forbid.second);
    printf("%d\n", totalTime);
}
else printf("%d\n", getTime(a[0].p));
for (int i = 1; i < n; i++) printf("%d\n", getTime(a[i].p));
```

Subtask 2 (15 points)

As **N** is small and **K ≤ 20000**

Simulate the Super Chats for **each second** from **0** to ~~**K**~~ **K+17999**

Check from **4th** Super Chat to **1st** Super Chat **(t_i ≤ currentTime ≤ ed_i)**

Add **one** second to the **top 3 latest active** Super Chats **at that moment**

Break when found **3** active Super Chats

Time complexity = **O(NK)**

Subtask 3 (18 points)

$1 \le N \le$ **1000**

$N \le K \le$ **20000**

N is small and **K ≤ 20000**, can use the previous solution

NK at most 2×10^7

O(NK) solution can pass within **1** second

Subtask 3 (18 points)

Simulate the Super Chats for **each second** from **0** to ~~**K**~~ **K+17999**

Check from **Nth** Super Chat to **1st** Super Chat **($t_i \le$ currentTime $\le ed_i$)**

Add **one** second to the **top 3 latest active** Super Chats **at that moment**

Break when found **3** active Super Chats

Time complexity = **O(NK)**

Subtask 4 (27 points)

$1 \le N \le$ **200000**

$N \le K \le$ **500000**

NK $\approx 10^{11}$

O(NK) solution can't pass in **1** second

Let's try to improve the **O(NK)** solution!

Subtask 4 (27 points)

Currently we find **top 3 latest active** Super Chats by linear scan

Scan from N^{th} Super Chat to 1^{st} Super Chat

Each second takes **O(N)** to search those **3** Super Chats

Result in **O(NK)**

If we can use less than **O(N)** to search for the top **3** Super Chats

We can achieve a better solution

Subtask 4 (27 points)

Want to find the **top 3 Super Chats** at a moment quickly

We can maintain the **lists of active Super Chats** by **stack**

The stack will store the **id** of the active Super Chats

At the **end** of **i^{th}** second, if there is a **new** Super Chat, **push** the **id** of it into the stack

Super Chats are **sorted in purchase time** in the stack

The **latest** Super Chat is on the **top** of the stack

Subtask 4 (27 points)

At the **beginning** of **i^{th}** second, we want to find the **top 3 Super Chats**

Scan from the **top** of stack to **bottom**

If the current super chat is **not expired** ($end_x < i$), add **one second** to its answer

- Save it to some temp memory and push it back (the stack need to remain sorted)

else **pop** it out

If we **already found 3 active Super Chats** in the stack, **break**

Still **O(NK)**?

Subtask 4 (27 points)

Let's say in **i^{th}** second, we accessed **m_i** elements in the stack

m_i-3 of them are **popped**

We **pushed** **N** elements into the stack (**N** Super Chats)

Sum(m_i-3) ≤ N

Sum(m_i) = O(N)

Time complexity = ~~**O(NK)**~~ **O(N + K)**

Full Solution

$1 \le N \le$ **200000**

$N \le K \le$ **10^9**

K is **too big** that **O(N + K)** solution **can't** pass in one second

Instead of simulating the super chats for each second

We can simulate the process in a **smarter** way

Full Solution

If the current super chat is **not expired** ($end_x < i$), add **one second** to its answer

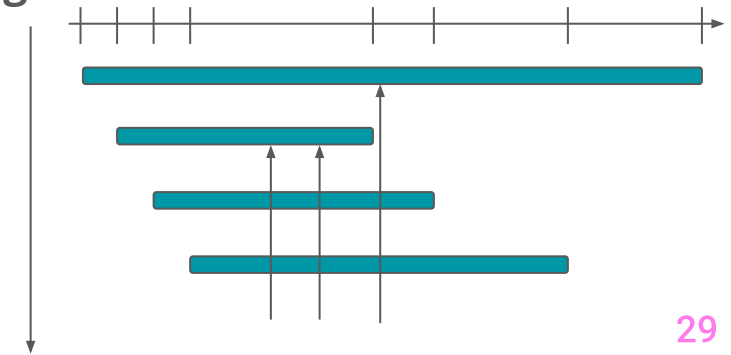
else **pop** it out **($end_x = i$)**

If there is a **new** Super Chat **($T_x = i$)**, **push** the **id** of it into the stack

When **$i = T_x$** or **end_x**, the **top 3 Super Chats may change**

else the **top 3 Super Chats remain unchanged**

$i = T_x$ or **end_x** => there are **2N important timestamp**



Full Solution

Instead of simulating the process for **each second**

Simulate the process for **each important timestamps**

Calculate the **display time** of **top 3 Super Chats** between **important timestamps**

- Instead of adding **one second** at a time

Full Solution

Add all the important timestamps into an array

Store **(time, id, type)** for each timestamps

- type 0 = **start** of the Super Chat, type 1 = **end** of the Super Chat

Sort it by **ascending time**

Process the important timestamps one by one

Full Solution

At the **beginning** of **i^{th}** ~~second~~ **timestamp**, we want to find the **top 3 Super Chats**

Scan from the **top** of stack to **bottom**

If the current super chat is **not expired**, add ~~one second~~ **the difference between the current and previous timestamp** to its answer

else **pop** it out

If we **already found 3 active Super Chats** in the stack, **break**

Full Solution

At the **end** of **i^{th}** ~~second~~ **timestamp**

If it is a **type 0** timestamp (start of a Super Chat), **push** the **id** of it into the stack

If it is a **type 1** timestamp (end of a Super Chat), **mark** the Super Chat as **expired**

Time complexity = **O(NlogN)**

- Bottleneck: sort

Full Solution

```
for (int i = 0; i < event.size(); i++) {
    vector <int> updateId;
    while (updateId.size() < 3 && stk.size()) {
        if (removed[stk.top()]) stk.pop();
        else {
            updateId.push_back(stk.top());
            stk.pop();
        }
    }
    int addTime = event[i].t;
    if (i - 1 >= 0) addTime -= event[i - 1].t;
    reverse(updateId.begin(), updateId.end());
    for (auto id : updateId) {
        stk.push(id);
        res[id] += addTime;
    }

    if (!event[i].type) stk.push(event[i].id);
    else removed[event[i].id] = 1;
}
```

Full Solution

You can also implement maintain the **active Super Chats** with **std::set**

Easier implementation

Larger constant

O(NlogN)

Full Solution

Another way is to **maintain the top 3 latest active SCs for each duration tier**

- Instead of maintain every SCs in a single stack

There are **9** duration tier for SC (ignore 0 mins)

We can maintain the active SCs for each tier by **queues**

Price	Colour	Pin duration and notes
$5 - $9	Blue	0 minutes. No chat message can be entered.
$10 - $24	Cyan	0 minutes
$25 - $49	Green	2 minutes
$50 - $99	Yellow	5 minutes
$100 - $249	Orange	10 minutes
$250 - $499	Magenta	30 minutes
$500 - $999	Red	1 hour
$1000 - $1499	Red	2 hours
$1500 - $1999	Red	3 hours
$2000 - $2499	Red	4 hours
$2500	Red	5 hours

Unlike maintaining in stack, **when a SC expires**

It **always locate in the front of the queue of its iter**

Full Solution

We can **remove** a SC **immediately** when it expires

- In stack, we remove it **lazily** (remove when we face an expired SC)

If we want to find the **top 3 latest SCs overall**

We **only care about top 3 SCs in each duration tier**

Full Solution

Therefore, there are only **9 x 3 = 27 candidates**

We want to **find the top 3 candidates** (by **sorting** / **partitioning**) and **update** their answers

- **Top 3 largest id**

Time complexity = **O(NlogN + 3Tlog(3T))** or **O(NlogN + 3T)**

T = number of duration iters

Full Solution

```
for (auto p : events) {
    if (p.first != last_time) {
        int last_duration = p.first - last_time;
        last_time = p.first;
        vector<pair<int, int>> candidates;
        for (const auto& v : scs) {
            for (int i = max(0, int(v.size()) - 3); i < v.size(); i++) {
                candidates.push_back(v[i]);
            }
        }
        sort(candidates.begin(), candidates.end());
        int count = 3;
        while (count && !candidates.empty()) {
            auto candidate = candidates.back();
            ans[candidate.second] += last_duration;
            count--;
            candidates.pop_back();
        }
    }
    if (p.second < 0) {
        scs[t[-p.second]].pop_front();
    } else {
        scs[t[p.second]].push_back(p);
    }
}
```

Any Questions?

Probably no...