Question 1

Solution to (a)

```
define MinimimalHall(S)
  m <- 0 // Maximum number of halls ever used
  H <- empty queue // Available halls released
  P <- Construct a priority queue P and insert all the intervals time points from S and assign a type (start or end) and its associated end point to each time point, and priority queue should compare the time points in ascending order, if there are two end time points with the same value, then one with larger start time point should be in the front; if the start time points has the same value, then the one with smaller end time point should be in the front.

  while P is not empty
    T <- extract-min(P)
    if T is a start time point
      if H is empty
        m <- m+1
        h <- m // Assign a new hall
      else
        h <- dequeue(H)

      T.h <- h
      T.end.h <- h // Assign the hall to the event and its associated end point
    else
      enqueue(H, T.h)
  return m
```

Running time is O(nlog n) since the priority queue has n elements and each operation takes O(log n) time.

Solution to (b)

```
define MinimalUnitSets(P)
  Sort P in ascending order
  S <- empty set
  i <- 1
  while i <= n
    Add (P[i], P[i]+1) to S
    Move i to the next element that is not covered by the interval (P[i], P[i]+1)

  return S
```

Prove correctness by induction

- Base case: When $n = 1$, the algorithm will add $(P[1], P[1] + 1)$ to S and return 1, which is correct.
- induction step: Assume the algorithm is correct for n = k, we need to prove it is correct for n = k+1. When n = k+1, the algorithm will add (P[k+1], P[k+1]+1) to S and move i to the next element that is not covered by the interval (P[k+1], P[k+1]+1). Since the algorithm is correct for n = k, the new interval (P[k+1], P[k+1]+1) will not overlap with any existing intervals in S. Therefore, the algorithm is correct for n = k+1.

Solution to (c)

Assume the greedy algorithm has the following steps

1. sort the coins in descending order
2. for each coin, if the coin is less than or equal to the remaining amount, add the coin to the change and subtract the coin from the remaining amount

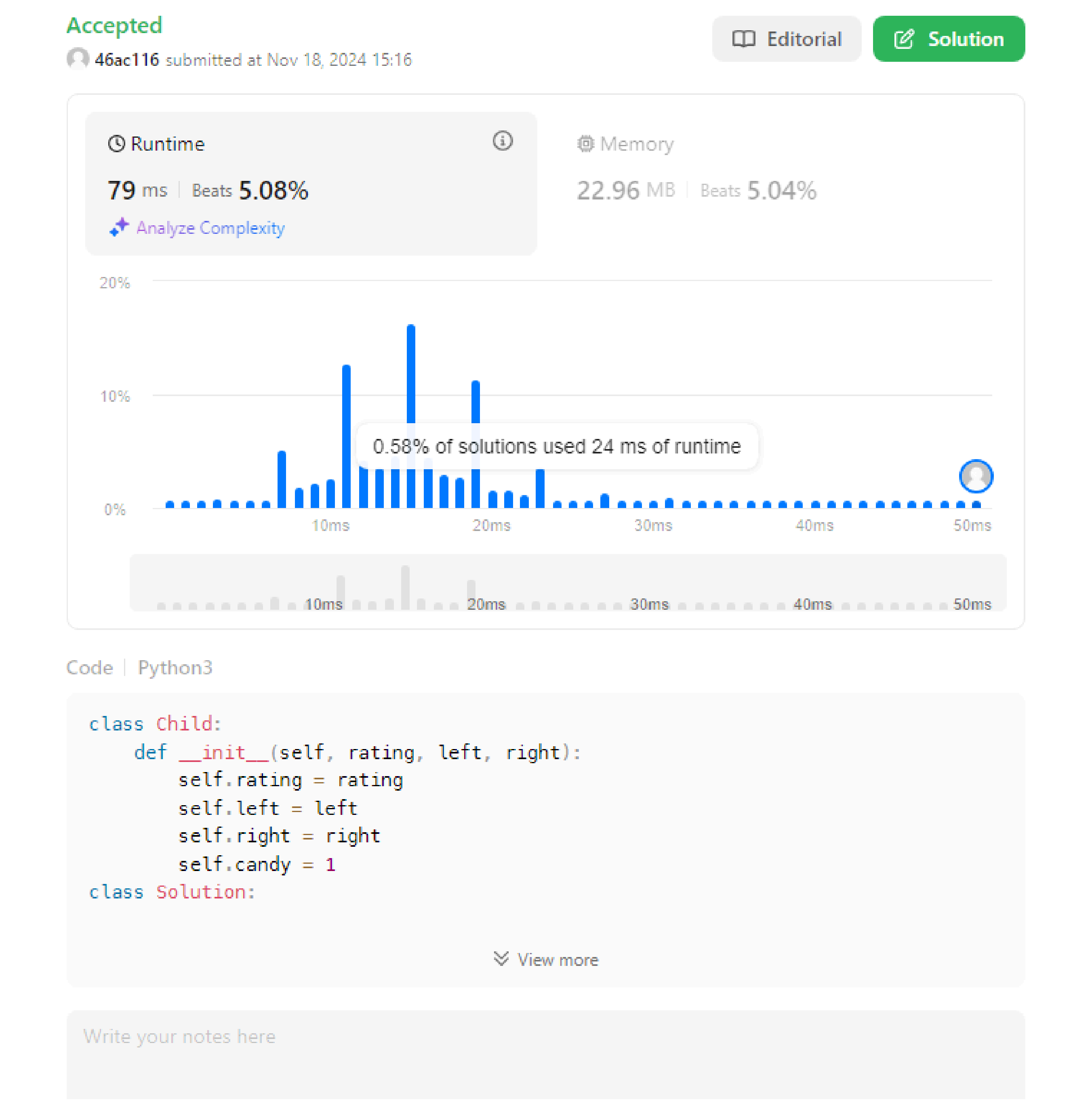
3. repeat step 2 until the remaining amount is 0

Given the optimal solution $S = a_0c^0 + a_1c^1 + ... + a_nc^n$ where $a_i > 0$
First we will prove $a_0 < c$ by contradiction, assume $a_0 \geq c$, then we can always reduce a_0 by c and increase a_1 by 1, which will result in a smaller number of coins which contradicts with the optimality. Therefore, $a_0 < c$
Then we will proof any natural number that can be represetated in base b is unique, that is for $N = a_0 + a_1b + a_2b^2 + ... + a_nb^n$ where $0 \leq a_i < b$ and $a_n > 0$ is unique. We will prove this by induction

- Base case: when $n = 0$, the representation is $0 = 0k^0$ is trivially unique
- Induction Step: assume the representation is unique for $n = k$, note for $n = k + 1$, we can write $N = a_0 + qb = a_0 + a_1b + qb^2 = a_0 + a_1b + a_2b^2 + qb^3 = ... = a_0 + a_1b + a_2b^2 + ... + a_kb^k + qb^{k+1}$. Since the representation is unique for $n = k$, the coefficient q is unique, therefore the representation is unique for $n = k + 1$

Notice that the coin change can be written as the based q represetation $a_0c^0 + a_1c^1 + ... + a_nc^n$ where $a_i > 0$ and and this representation is unique. Observe that the greedy algorithm will pick the largest coin that is less than or equal to the remaining amount, that is it will generate the power of c in descending order. Therefore, the greedy algorithm will generate the representation of the coin change in the base c representation, which is unique. Therefore, the greedy algorithm is optimal.

Question 2

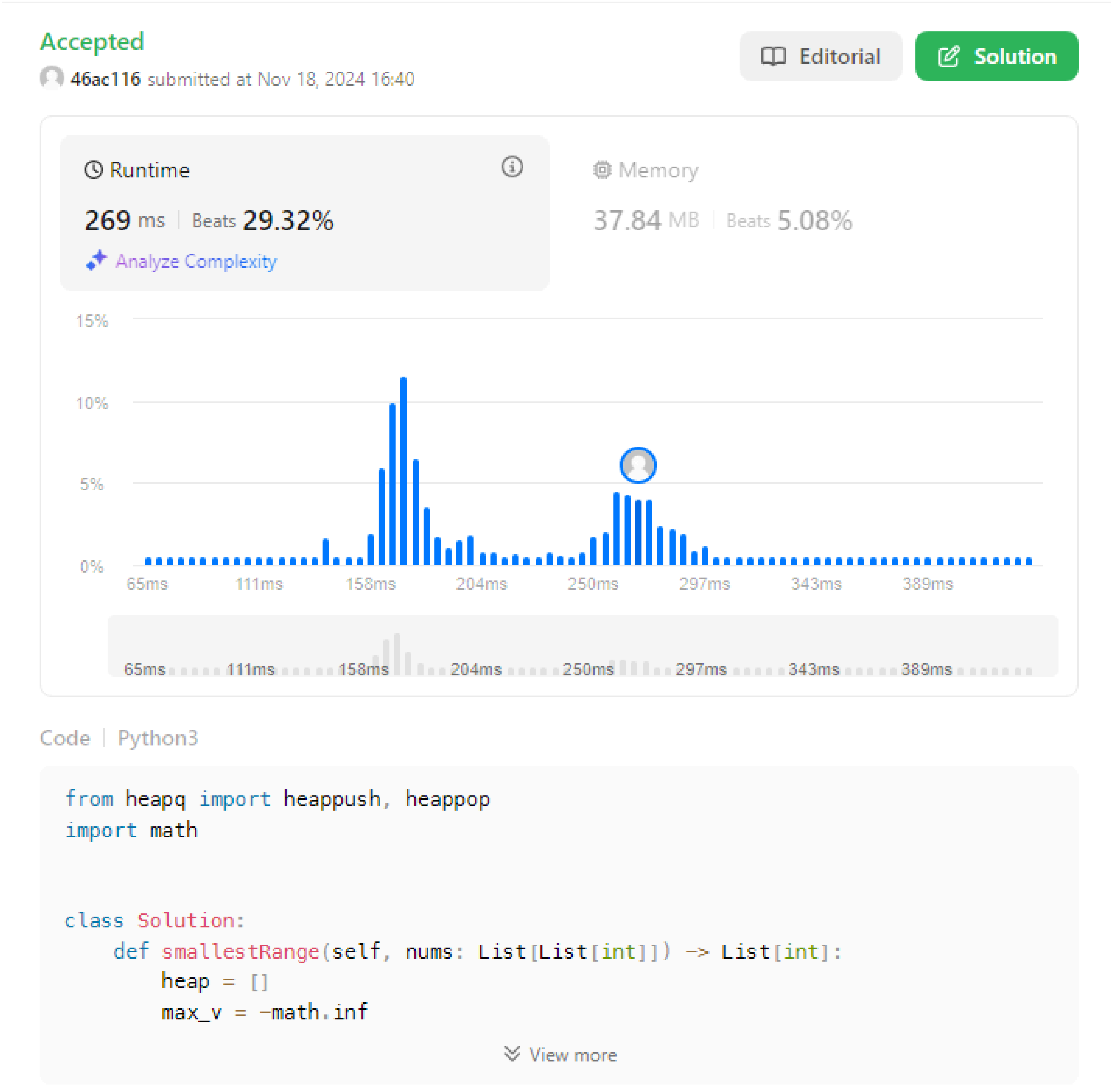

Accepted
46ac116 submitted at Nov 18, 2024 15:16

Runtime
79 ms | Beats 5.08%

Memory
22.96 MB | Beats 5.04%

Code | Python3

```python
class Child:
    def __init__(self, rating, left, right):
        self.rating = rating
        self.left = left
        self.right = right
        self.candy = 1
class Solution:
```


Accepted

46ac116 submitted at Nov 18, 2024 16:40

Runtime

269 ms | Beats 29.32%

Memory

37.84 MB | Beats 5.08%

Code | Python3

```
from heapq import heappush, heappop
import math


class Solution:
    def smallestRange(self, nums: List[List[int]]) -> List[int]:
        heap = []
        max_v = -math.inf
```