

연어덮밥

최종 레포트

연어덮밥 



연어덮밥 알고리즘

연어덮밥 알고리즘

메인이 되는 연어 알고리즘

+

뒷받침 하는 와사비, 밥 알고리즘

연어 알고리즘

1. $K = 1$

2. K 개씩 양팔에 매단다 (K 개와 K 개를 비교) - **Balance()**

3-1. 양팔의 무게가 같다면 (스텝사이즈 늘리기!)

(1) 오른팔에 있는 동전을 왼팔로 다 가져온다

(2) $K = K * 2$

(3) 오른팔에 K 개 만큼 분류 안된 동전을 가져온다.

(4) 2번 과정으로 이동

3-2. 양팔 무게가 다른데 $K = 1$ 이라면

무게가 작은 것은 가벼운 동전 그룹, 큰 것은 무거운 동전 그룹에 담는다.


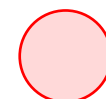
3-3. 양팔 무게가 다른데 $K > 1$ 이라면

왼팔이 무거우면 왼팔을 모두 무거운 동전 그룹에,

오른팔이 무거우면 왼팔을 모두 가벼운 동전 그룹에 담는다.

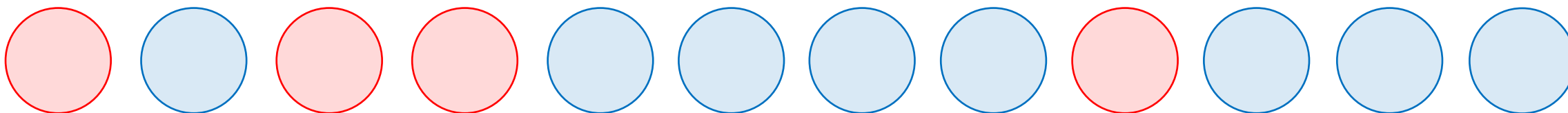
4. 오른팔에 있는 것은 그대로 분류 되지 않은 동전 그룹으로 돌려 놓는다. $K=1$ 로 두고 2번으로 이동

연어 알고리즘

 진짜 동전
 가짜 동전


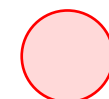
예를 들어

다음과 같이 동전이 나열되어 있다고 생각합시다.

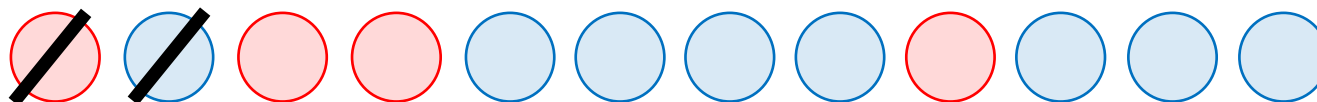
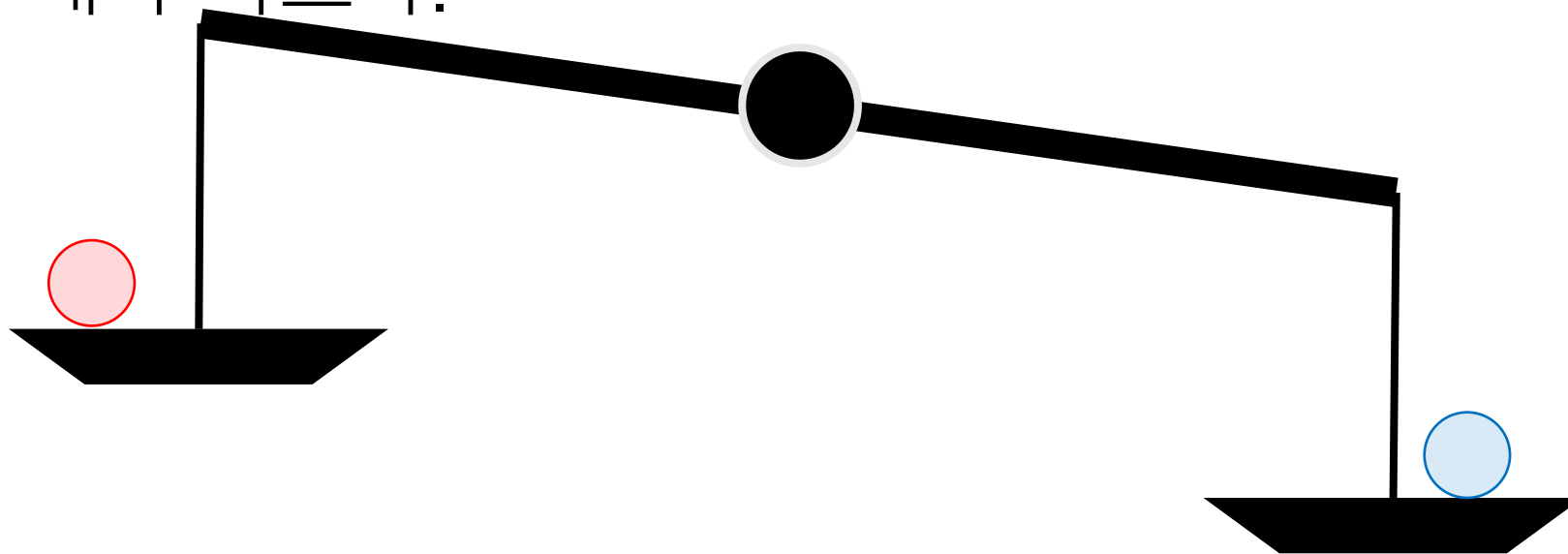




연어 알고리즘


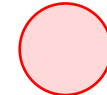
-  진짜 동전
-  가짜 동전

$K=1$ 인데 무게가 다르다.

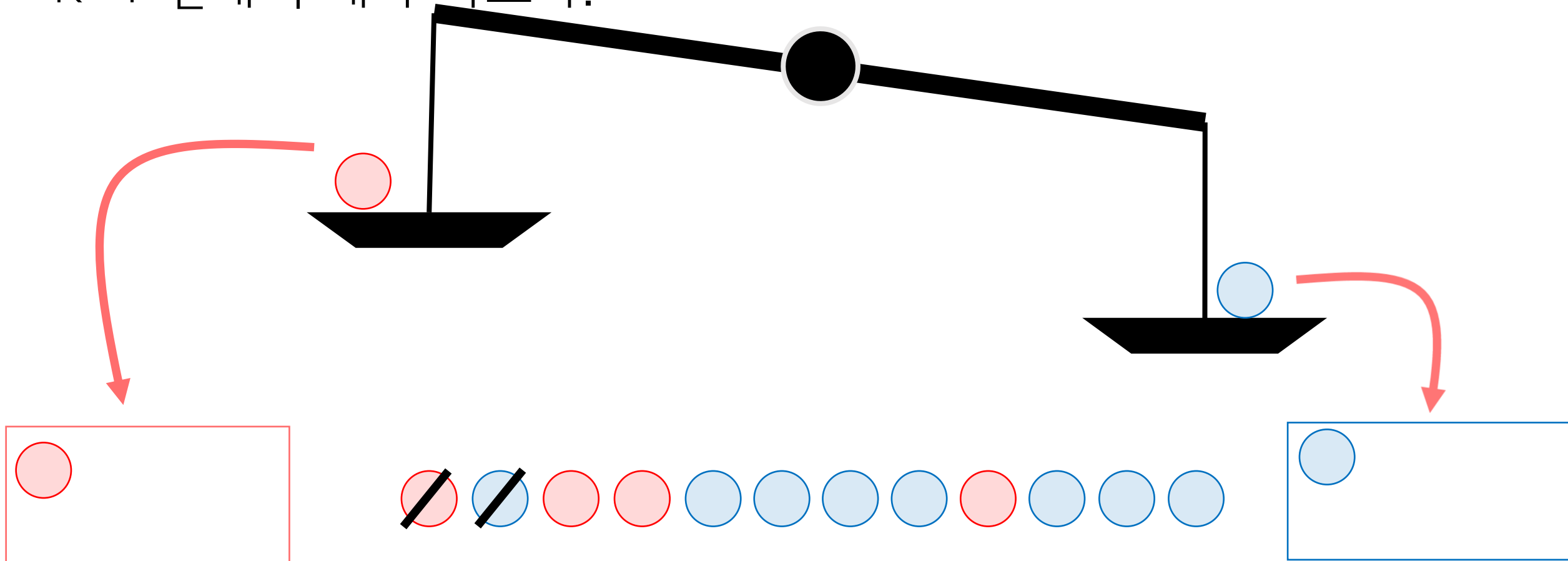




연어 알고리즘

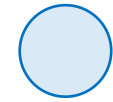
-  진짜 동전
-  가짜 동전

K=1 인데 무게가 다르다.

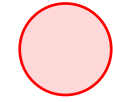




연어 알고리즘

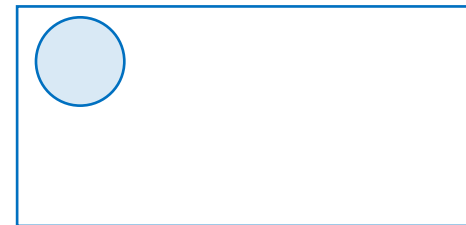
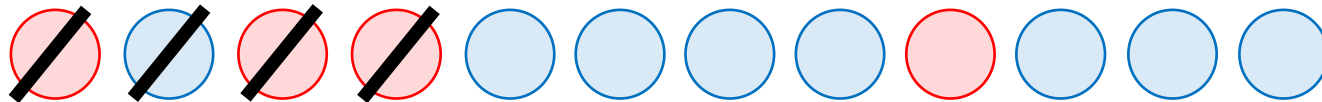
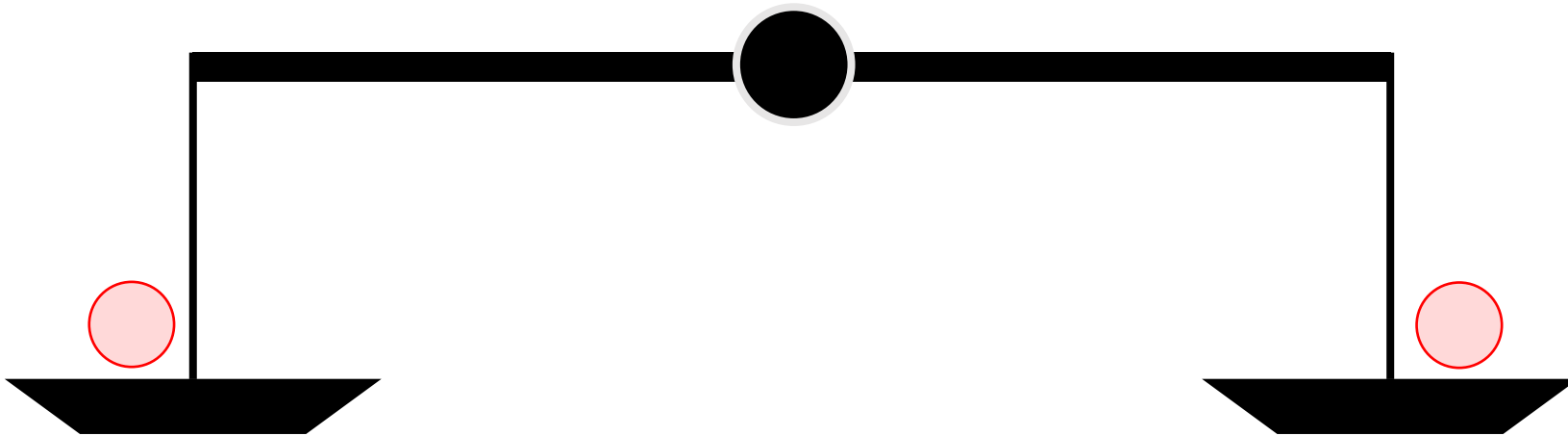


진짜 동전



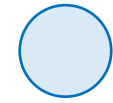
가짜 동전

$K=1$ 인데 무게가 같다.

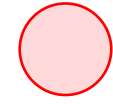




연어 알고리즘

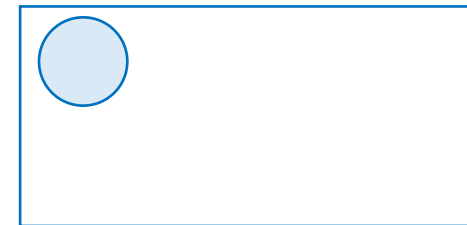
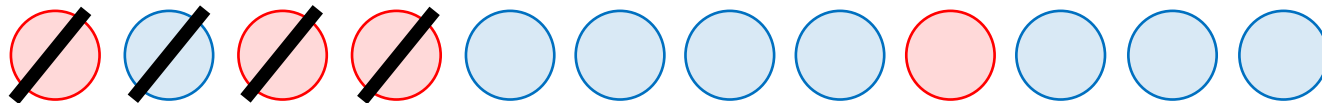
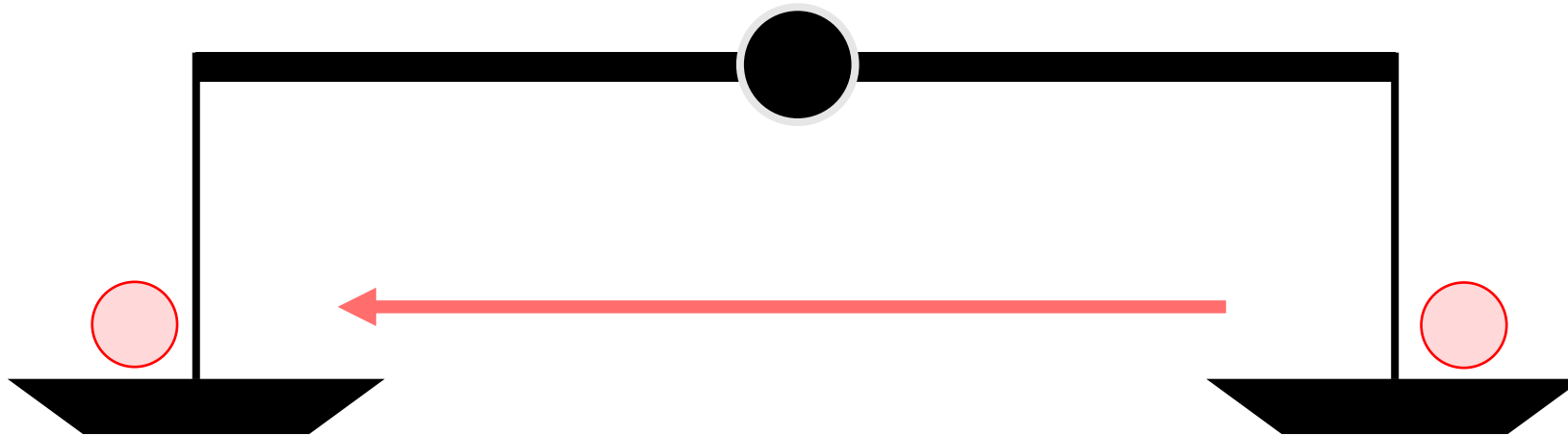


진짜 동전



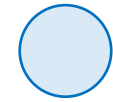
가짜 동전

$K=1$ 인데 무게가 같다.

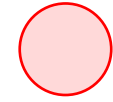




연어 알고리즘

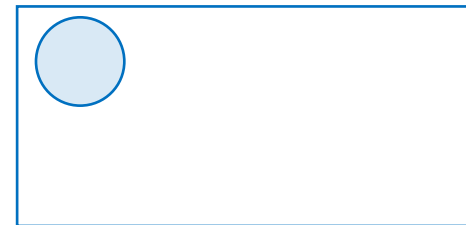
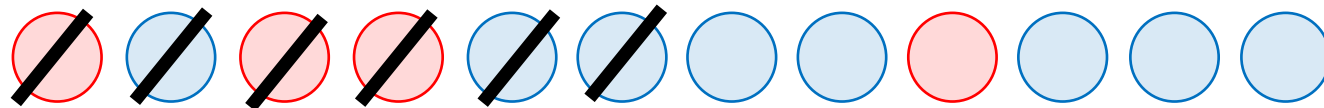
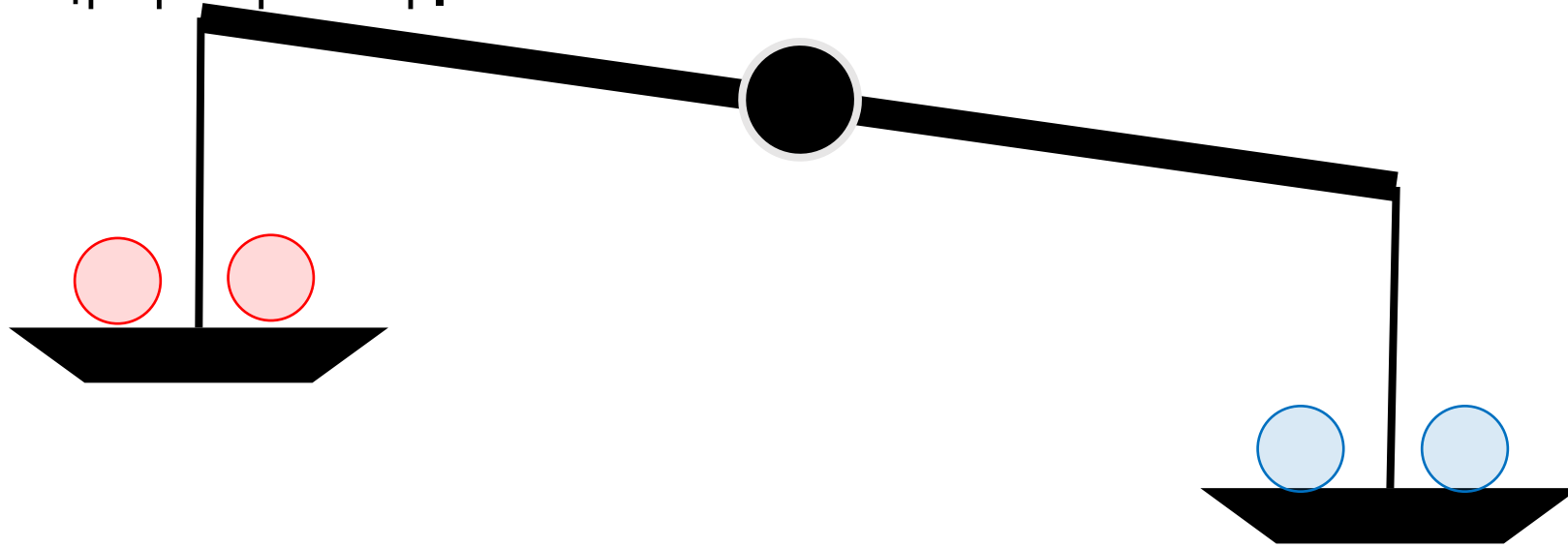


진짜 동전



가짜 동전

K=2 인데 무게가 다르다.

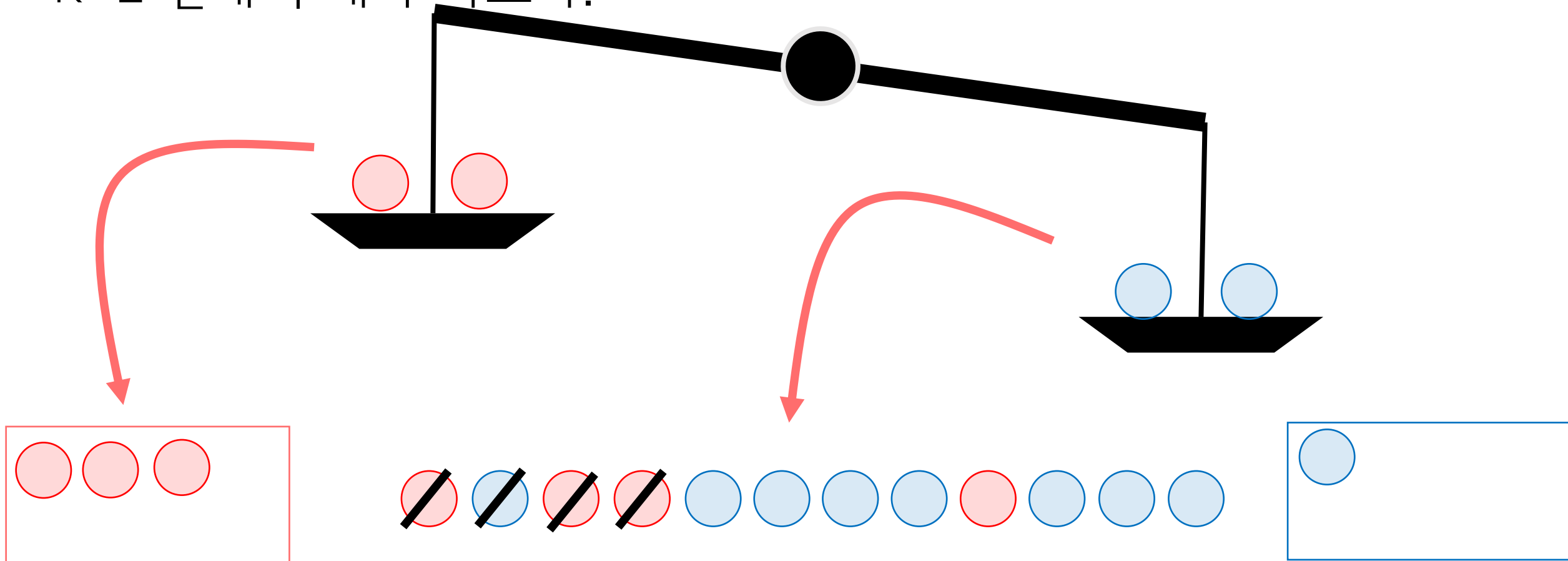





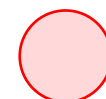
연어 알고리즘

● 진짜 동전
● 가짜 동전

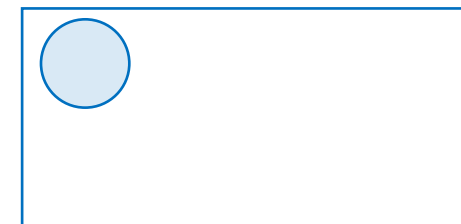
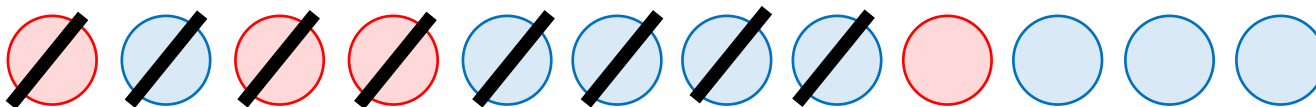
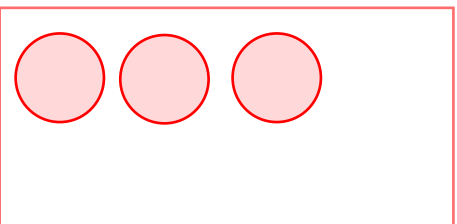
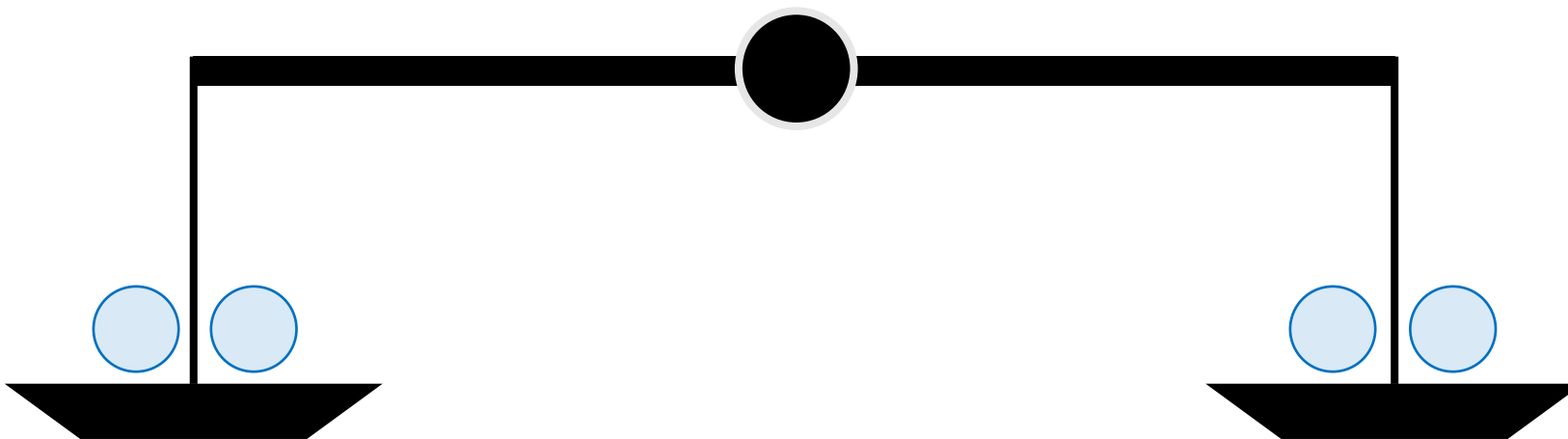
K=2 인데 무게가 다르다.




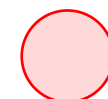
연어 알고리즘

 진짜 동전
 가짜 동전

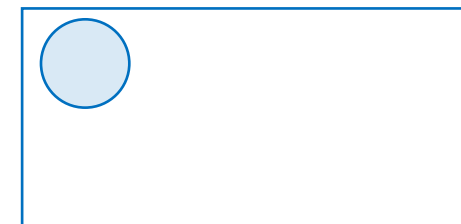
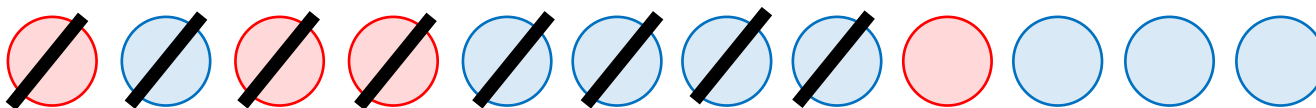
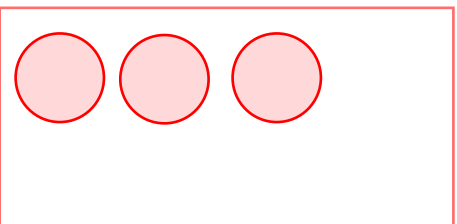
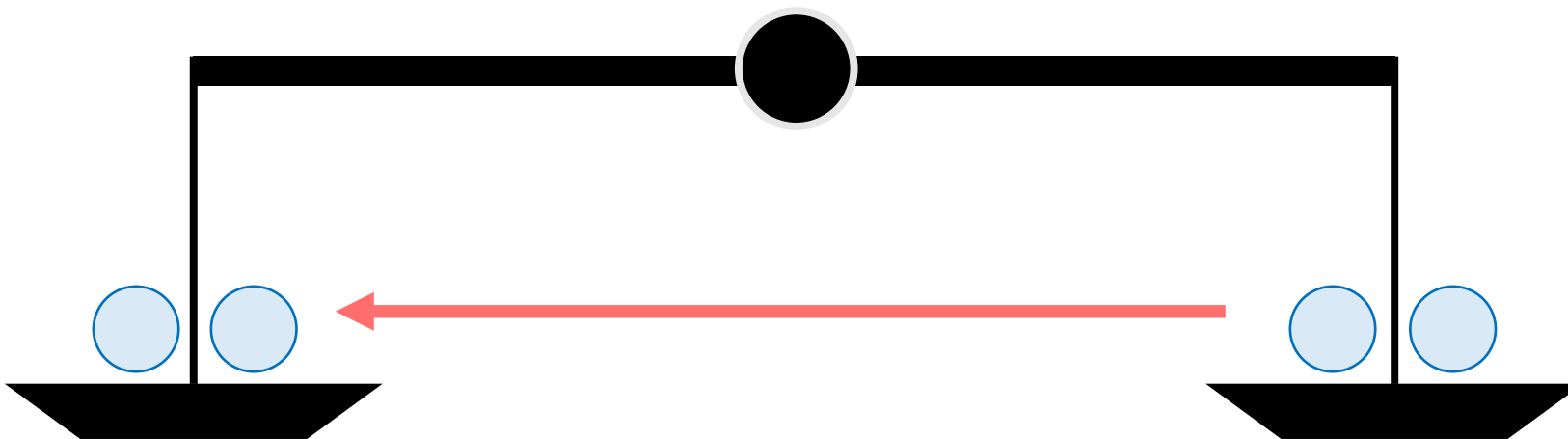
K=2 인데 무게가 같다.



연어 알고리즘

 진짜 동전
 가짜 동전

$K=2$ 인데 무게가 같다.

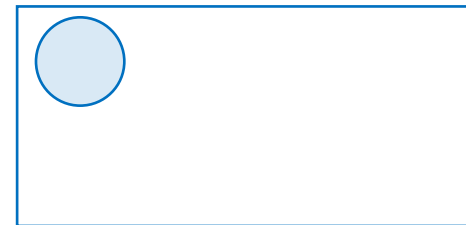
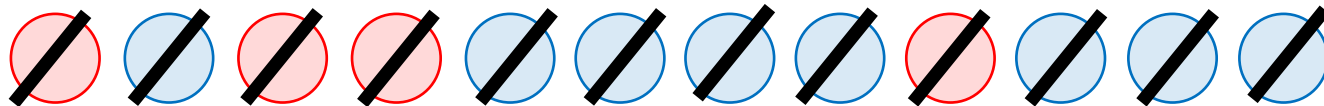
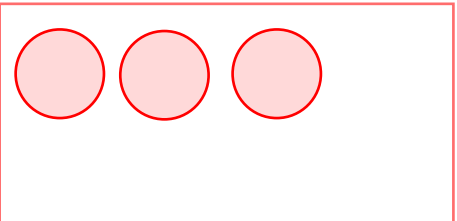
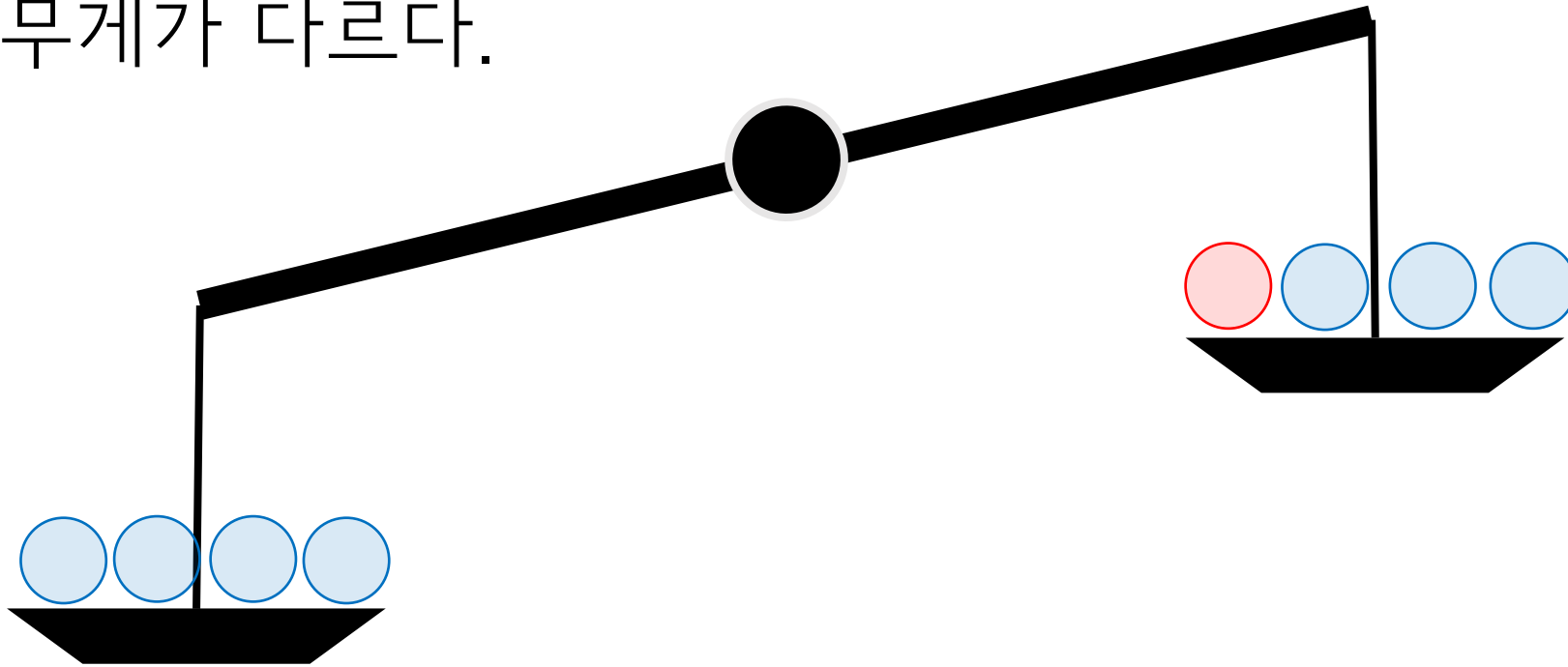




연어 알고리즘


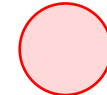
진짜 동전
가짜 동전

K=4 인데 무게가 다르다.

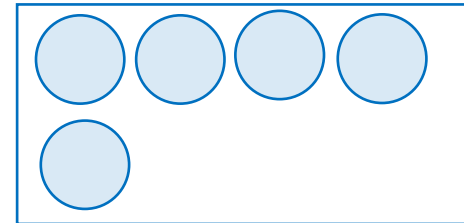
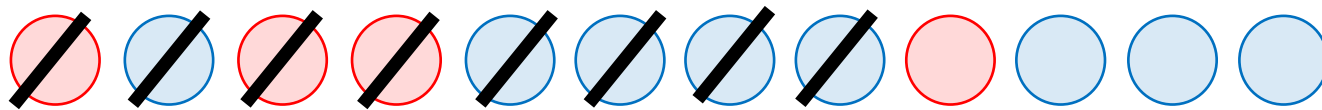
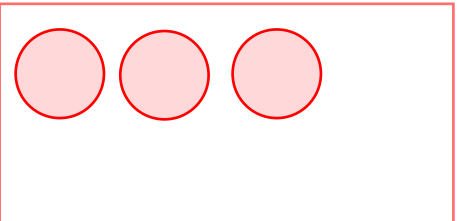
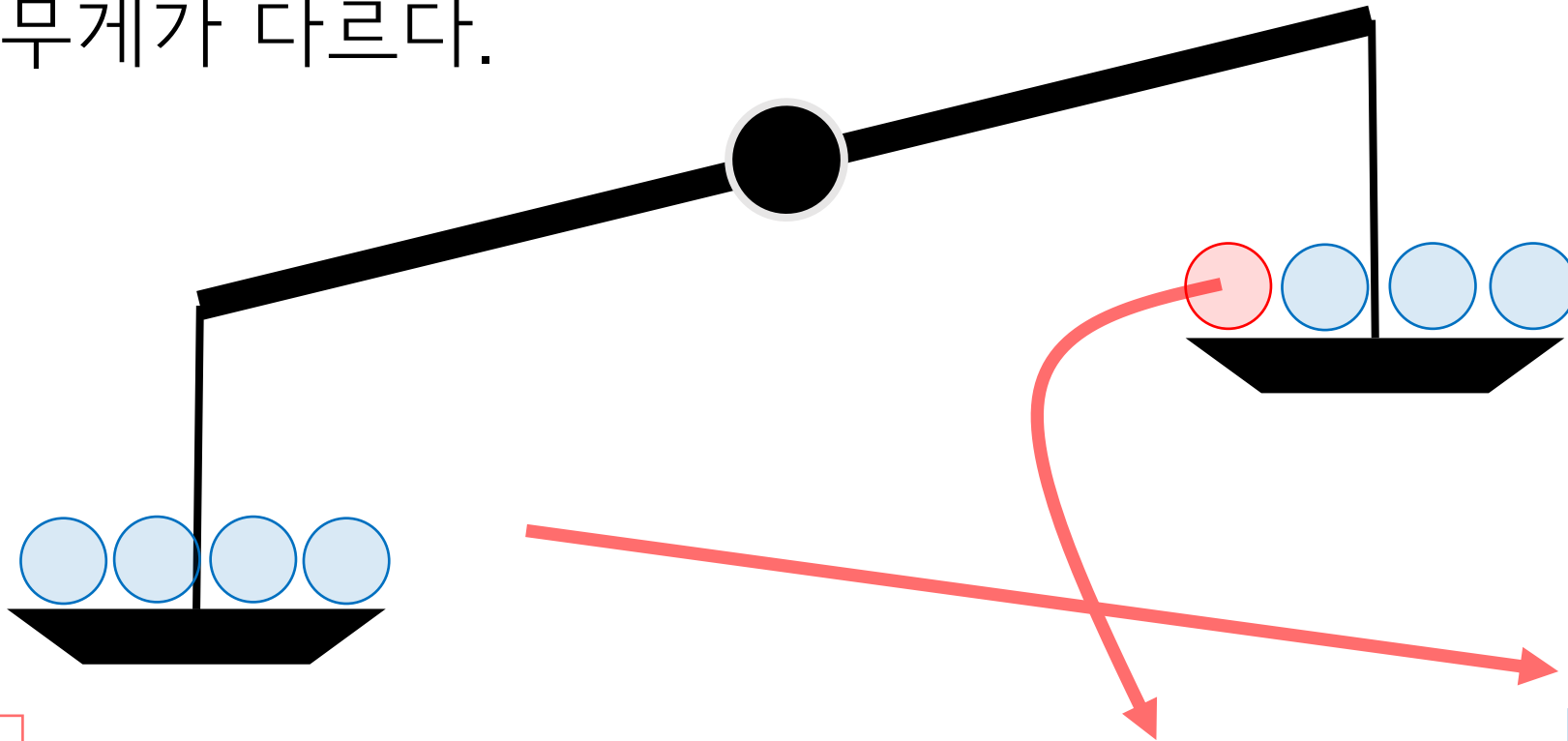




연어 알고리즘

-  진짜 동전
-  가짜 동전

K=4 인데 무게가 다르다.





연어 알고리즘

요약

연어 알고리즘은 unknown set의 크기를 2^n 의 크기로 늘려가며 분류한다.

연어 알고리즘

연어 알고리즘은 다음과 같은 이유로 잘 동작합니다.

unknown set과 비교했을 때, equal이면(=성공하면) 지수적($K = K * 2$)으로 확장한다.

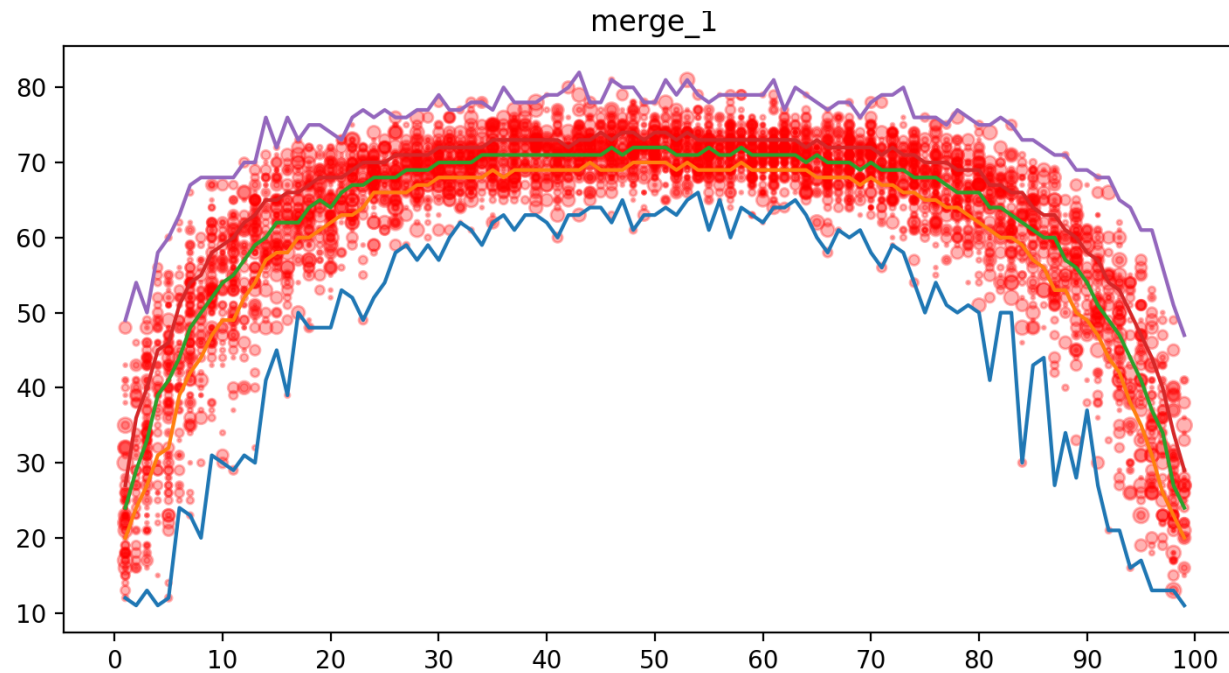
✓ 무게가 같은 동전이 많으면 빠르게 분류가 된다.

unknown set과 비교했을 때, equal이 아니면(=실패하면) 무조건 unknown set은 분류가 끝난다.

✓ unknown set이 모두 무거우면 이 set보다 무거운 set은 없다.

✓ unknown set이 모두 가벼우면 이 set보다 가벼운 set은 없다.

연어 알고리즘



연어 알고리즘

와사비 알고리즘

연어 알고리즘을 개선해보겠습니다.

와사비 알고리즘은 다음 논문을 참고했습니다.

P.J. Wan, Q. Yang and D. Kelley,
A $\frac{3}{2} \log 3$ -competitive algorithm for the
counterfeit coin problem,
Theoretical computer science (1997)



Theoretical Computer Science 181 (1997) 347–356

Theoretical
Computer Science

A $\frac{3}{2} \log 3$ -competitive algorithm for the counterfeit coin problem

Peng-Jun Wan^{a,*}, Qifan Yang^{a,l}, Dean Kelley^b

^aComputer Science Department, University of Minnesota, Minneapolis, MN 55455, USA

^bSt. Mary's University

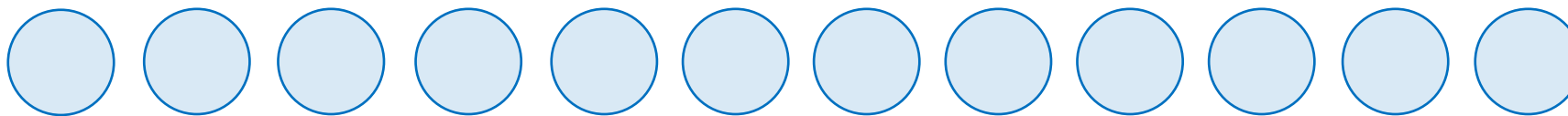
Abstract

We study the following *counterfeit coin problem*: Suppose that there is a set of n coins. Each one is either *heavy* or *light*. The goal is to sort them according to weight with a minimum number of weighings on a balance scale. Hu and Hwang gave an algorithm with a competitive ratio of $3 \log 3$ (all logarithms are base-2). Hu, Chen and Hwang also gave an algorithm with a competitive ratio of $2 \log 3$. In this paper we give an improved algorithm whose competitive ratio is $\frac{3}{2} \log 3$.

와사비 알고리즘 - 용어 정의

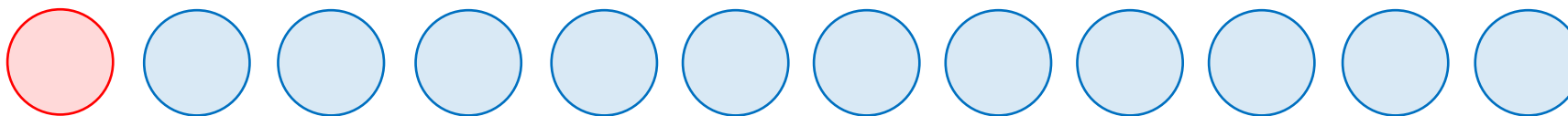
✓ uniform set

모두 같은 type인 set을 말합니다.



✓ unique set

동전 하나를 제외하고 모두 같은 type인 set을 말합니다.



와사비 알고리즘

✓ 기존 방법

1. **uniform set**을 확장하면서 비교한다.
2. 실패하면 uniform set을 모두 분류하고 **1개부터 다시** uniform set을 만든다.

✓ 개선 방법

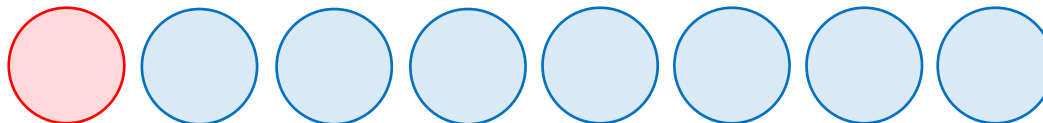
1. **uniform set** 또는 **unique set**으로 확장하면서 비교한다.
2. 실패를 하면 **binary search**를 수행한다.

와사비 알고리즘

uniform set이 아닌 **unique set**을 이용하면 어떻게 개선할 수 있을까요?

다음과 같은 **unique set**을 이용한 비교를 예를 들어서 살펴봅시다.

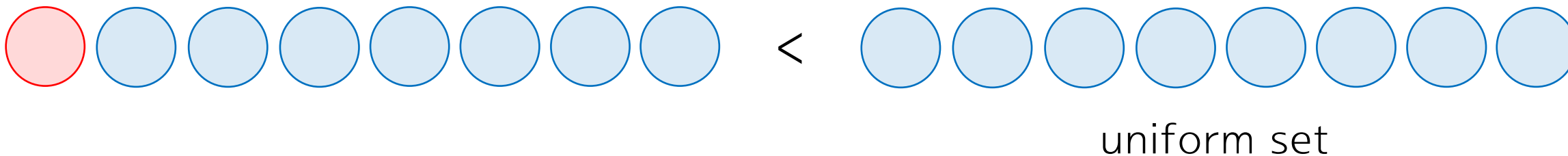
ex) light coin 1, heavy coin 7



와사비 알고리즘

① SMALL

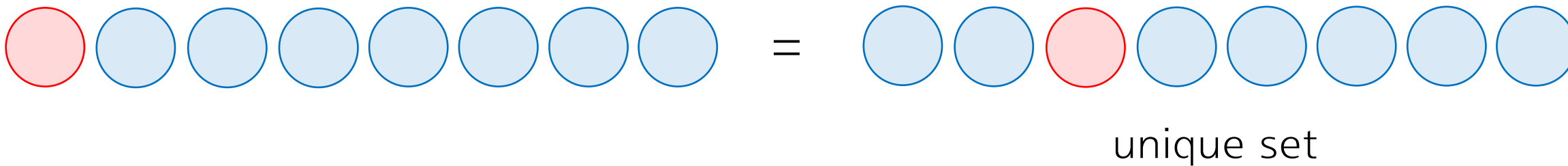
- ★ 비교한 동전은 **uniform set** 입니다.
- 하나로 합쳐도 unique set이다. 따라서 두 set을 합쳐서 지수적으로 확장합니다.



와사비 알고리즘

② EQUAL

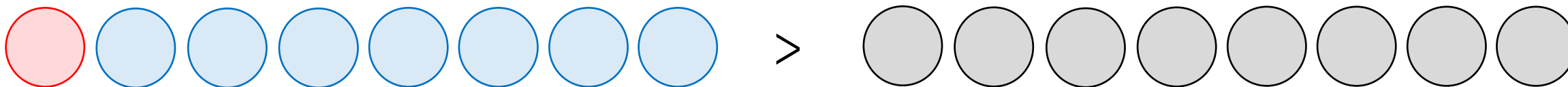
- ★ 비교한 동전은 **unique set** 입니다.
- unique set에서 **binary search**를 해서 동전 하나를 찾습니다.



와사비 알고리즘

③ LARGE

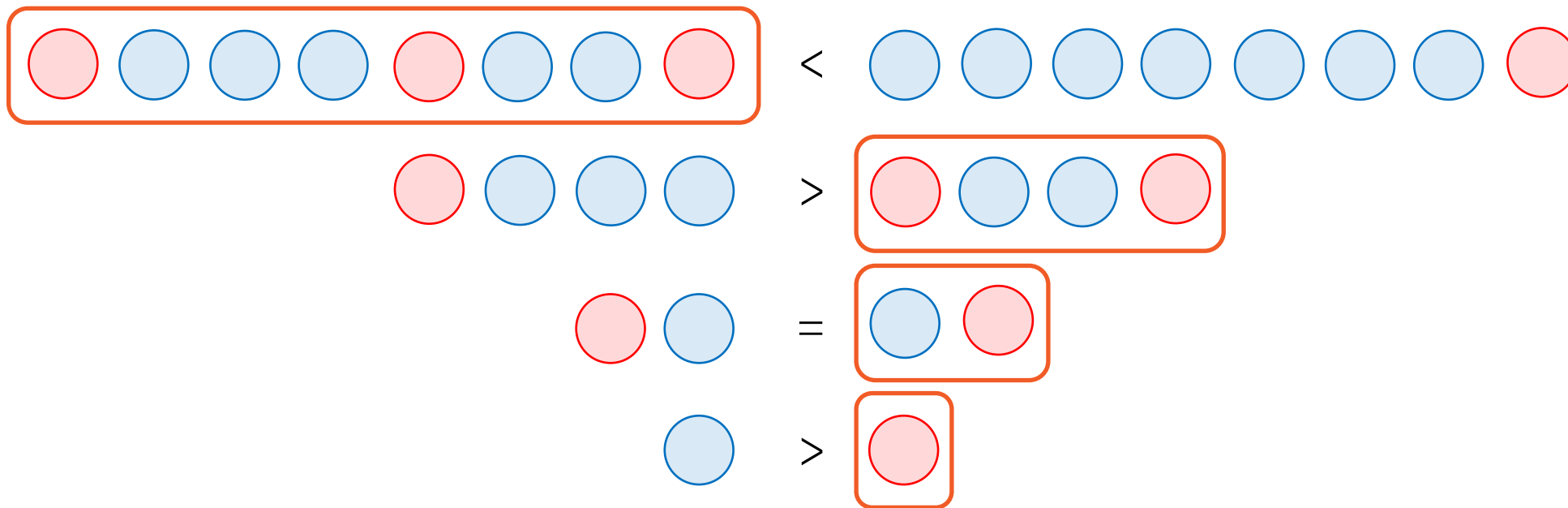
- 적어도 2개 이상이 light coin 입니다.
- 일단 이 경우에도 **binary search**를 수행합니다.



와사비 알고리즘

✓ binary search

- set을 $\frac{1}{2}$ 씩 나눠서 비교하며 동전을 찾는 과정을 말합니다.
- set의 크기를 N이라 할 때, binary search는 $\log_2 N$ 만에 원하는 동전을 찾습니다.



와사비 알고리즘

★ binary search를 더 활용해보시다.

와사비 알고리즘

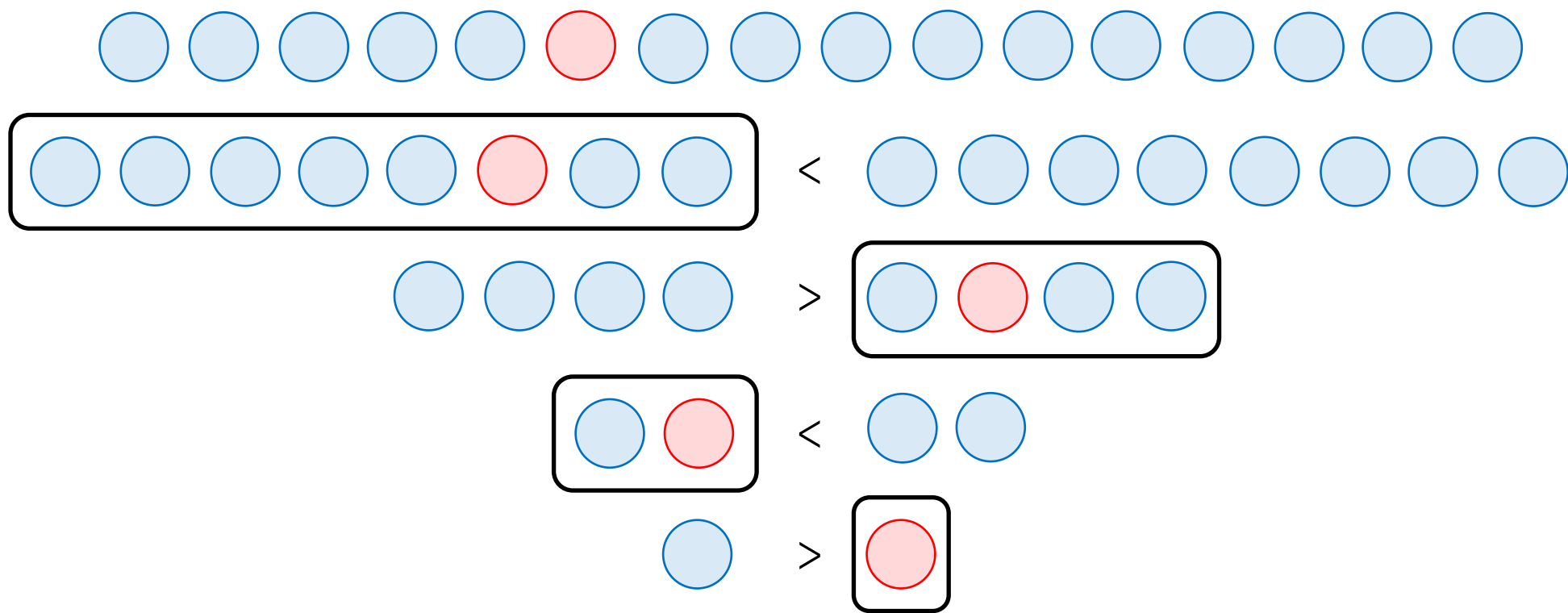
★ binary search를 더 활용해보시다.

- ✓ binary search를 하는 과정에서 **equal**이 한 번도 없었다면
 - 무조건 전체는 **unique set**입니다.
 - unique set은 binary search만 수행하면 전체 분류가 끝납니다.

와사비 알고리즘

equal이 한 번도 없는 경우

→ 전체가 unique set



와사비 알고리즘

★ binary search를 더 활용해보시다.

- ✓ binary search를 하는 과정에서 **equal**이 단 한 번이면
 - 한번의 equal일 때 비교한 두 set을 A_1, A_2 라 하면 A_1, A_2 는 **unique set**입니다.
 - A_1, A_2 를 제외한 나머지 $U - (A_1 \cup A_2)$ 는 **uniform set** 또는 **unique set**입니다.

★ $U - (A_1 \cup A_2)$ 에 대해서 binary search를 수행해보시다.

uniform set인 경우 → 한번의 비교로 전체가 판별됩니다. (equal이 나옵니다.)

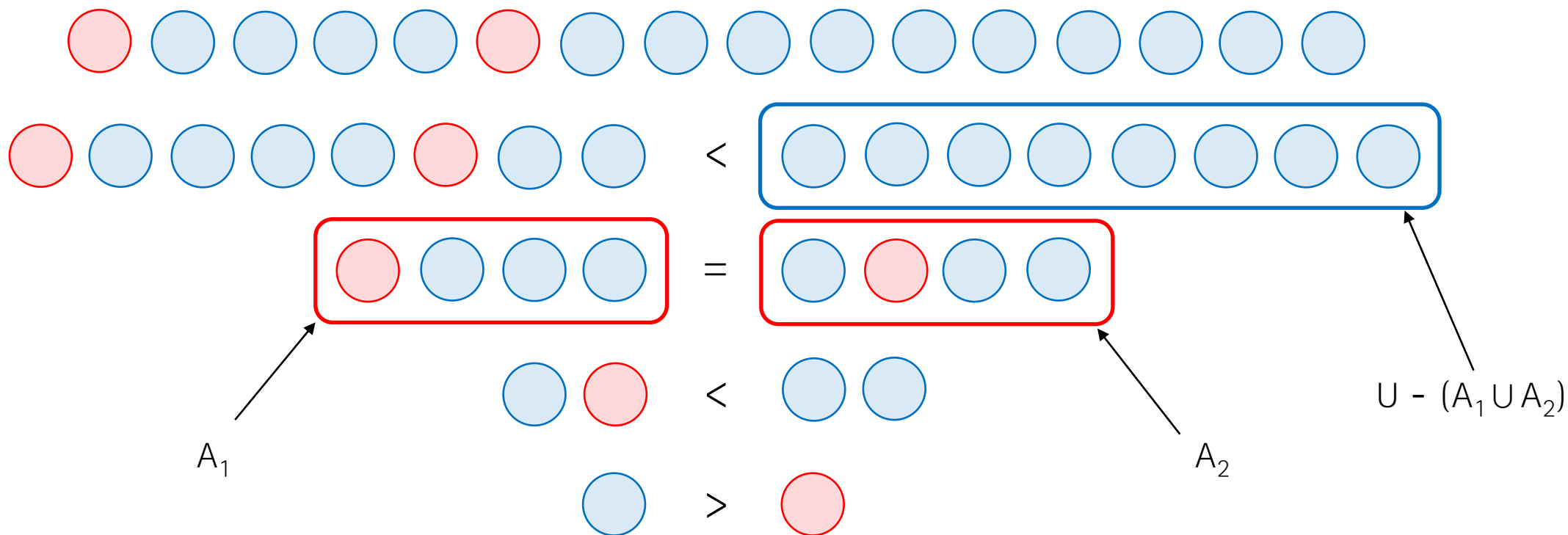
unique set 인 경우 → binary search를 끝까지 수행합니다. (equal이 한번도 나올 수 없습니다.)

와사비 알고리즘

equal이 단 한 번인 경우

A_1, A_2 : unique set

$U - (A_1 \cup A_2)$: uniform 또는 unique set



와사비 알고리즘

★ binary search를 더 활용해보시다.

- ✓ binary search를 하는 과정에서 **equal**이 두 번 이상이면
 - 마지막 equal일 때 비교한 두 set을 A_1, A_2 라 하면 A_1, A_2 는 **unique set**입니다.
 - A_1, A_2 를 제외한 나머지 $U - (A_1 \cup A_2)$ 는 두 개 이상의 찾으려는 동전을 가집니다.

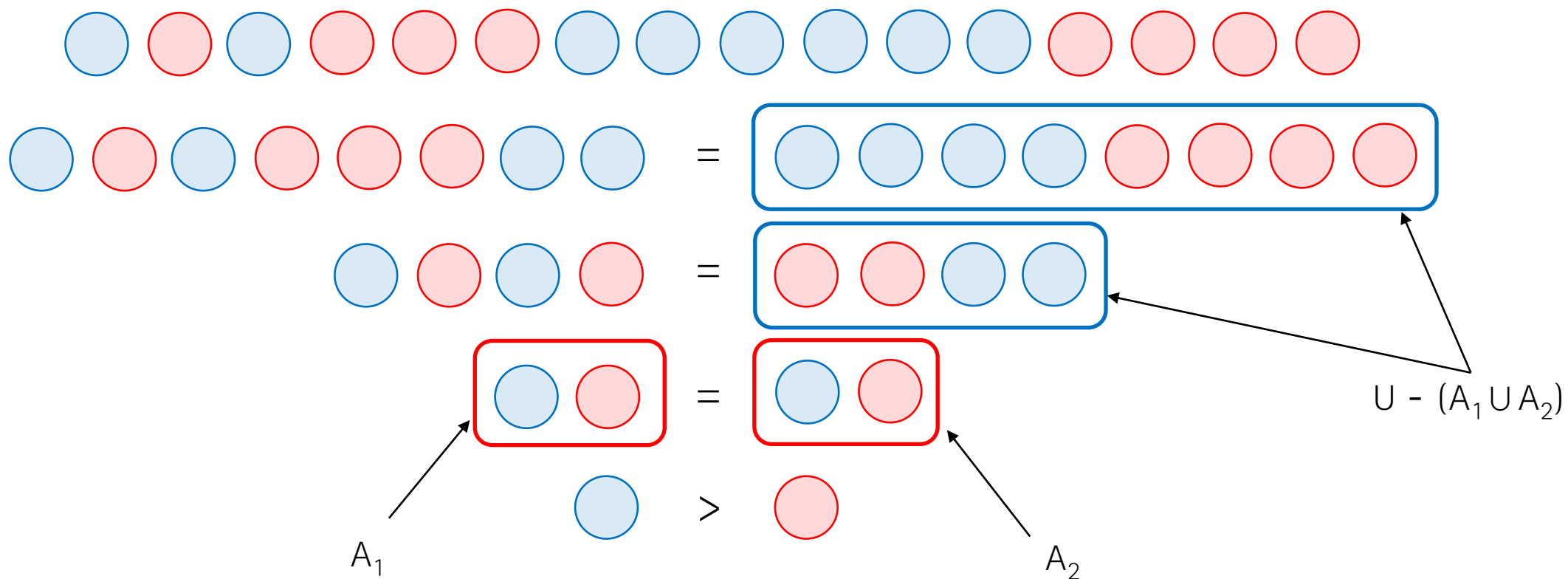
A_2 에 대해서는 다시 binary search를 수행하여 분류합니다.

$U - (A_1 \cup A_2)$ 는 활용할 수 없습니다.

와사비 알고리즘

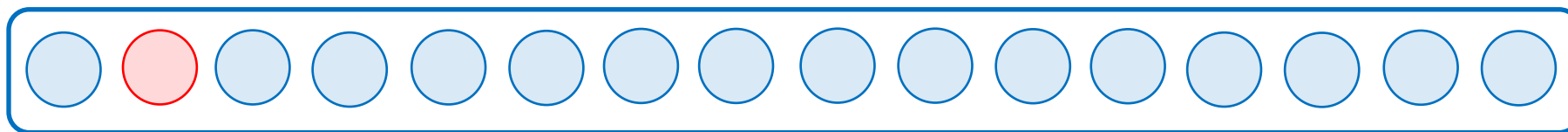
equal이 두 번 이상인 경우

A_1, A_2 : unique set

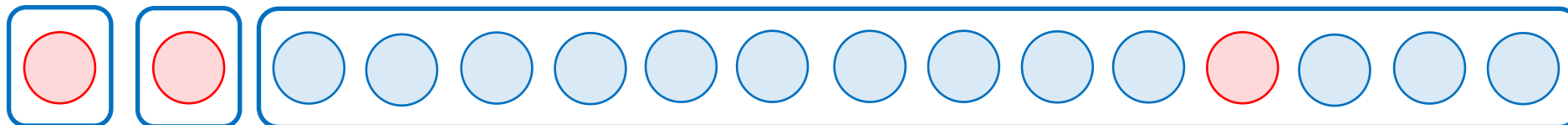


와사비 알고리즘

※ binary search는 같은 type의 동전이 많을 때, 적은 횟수로 동전을 분류합니다.



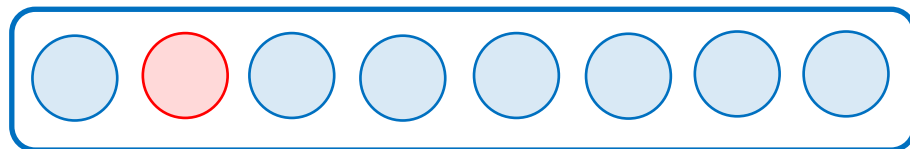
unique set



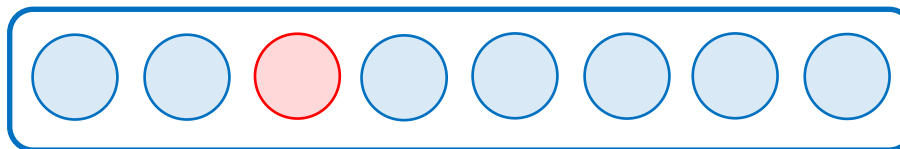
A_1

A_2

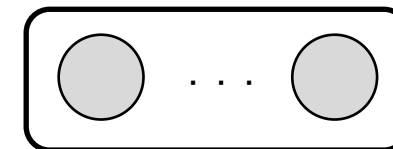
$U - (A_1 \cup A_2)$



A_1



A_2



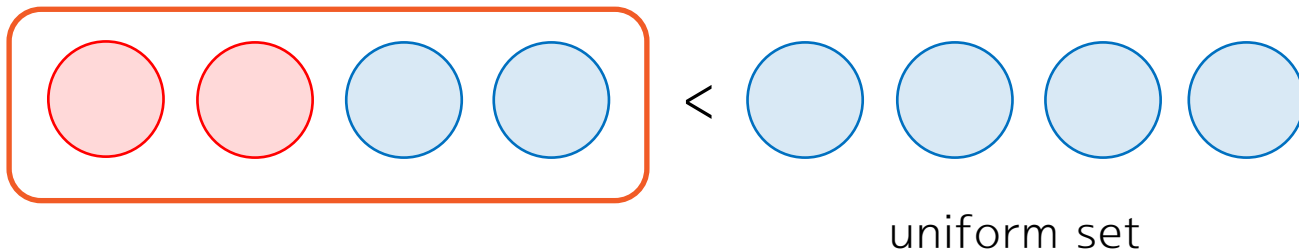
$U - (A_1 \cup A_2)$

와사비 알고리즘

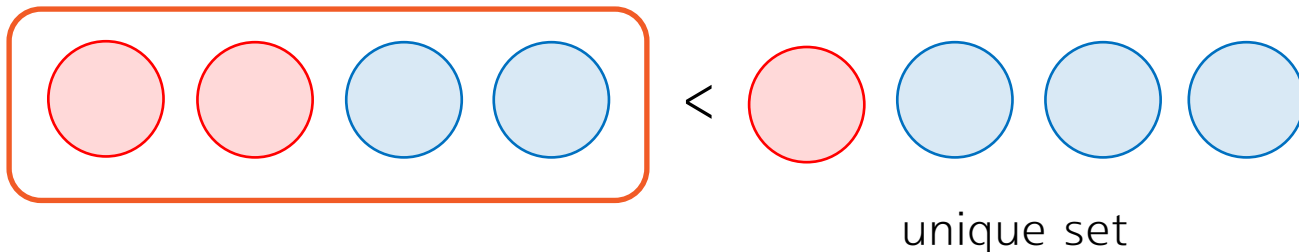
※ [참고] two coins set을 활용한 개선 - unique set이 두 개가 생기면 이를 활용해보시다.

(이 방법은 100개의 동전이 unique set에 가까울 때만 효과가 있습니다.)

1. unique set 두 개를 합친다. (size ↑)



2. 이 set과 비교해서 **LARGE**인 경우
unique 또는 uniform set 이다.



3. 나머지 경우는 활용할 수 없다.



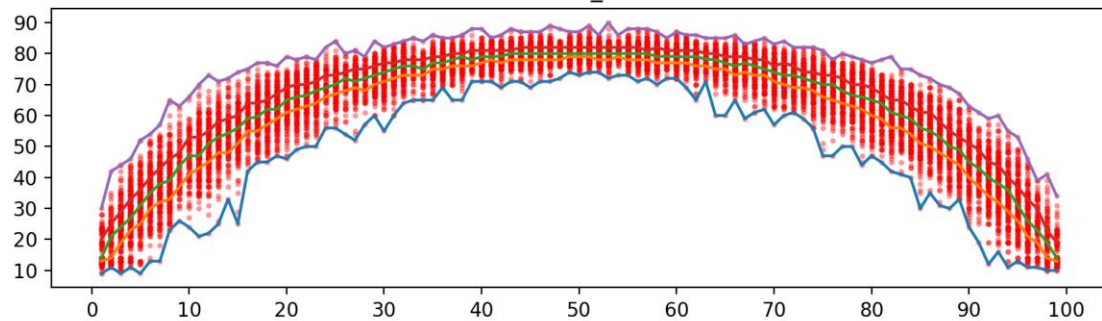
와사비 알고리즘

요약

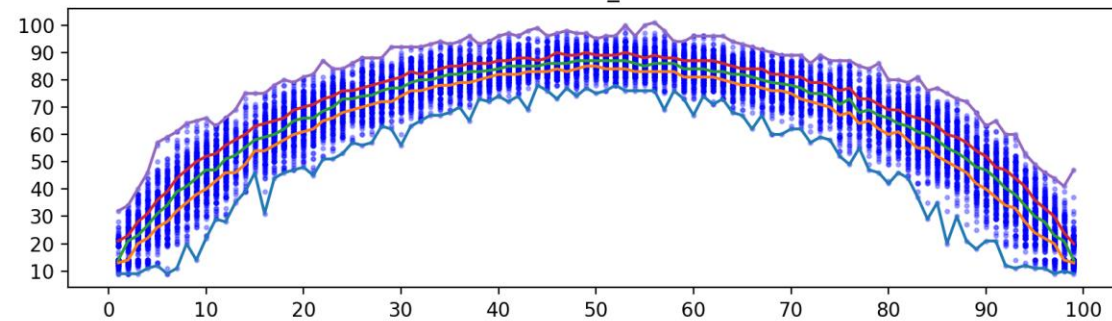
와사비 알고리즘은 연어 알고리즘과 같이 set의 크기를 2^n 의 크기로 늘려가며 분류한다.
하지만 와사비 알고리즘은 실패할 경우 binary search를 수행하여 개선하였다.

와사비 알고리즘

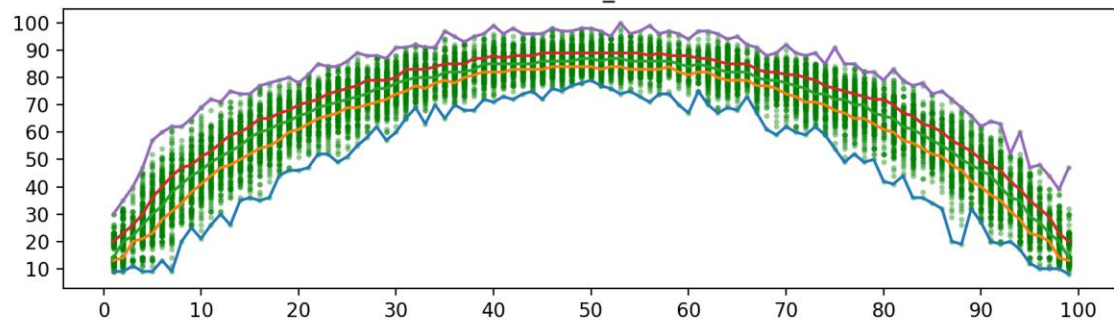
final_1



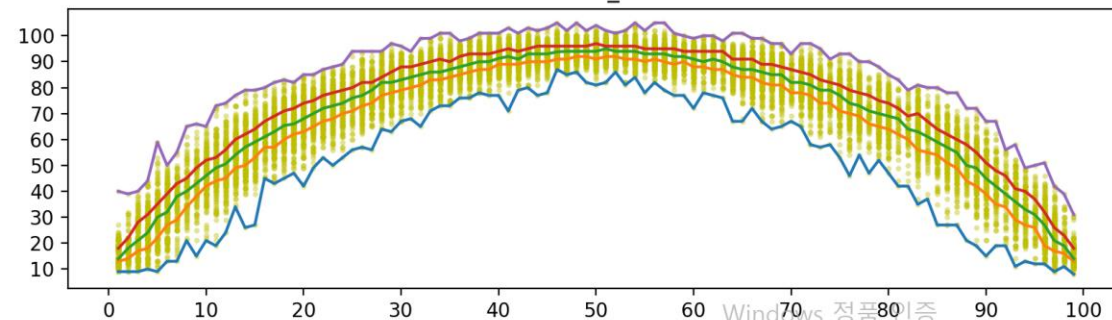
final_2



final_3



final_4





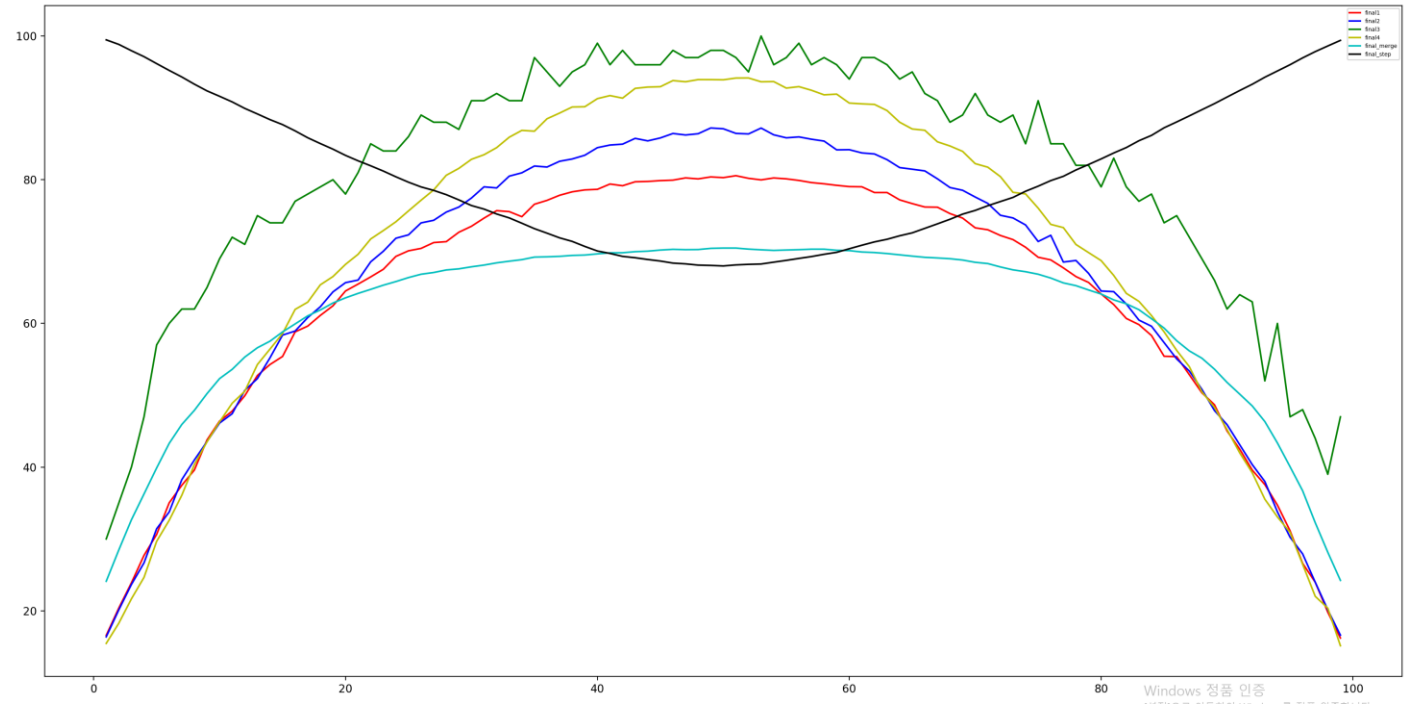
와사비 알고리즘

하늘색 : 연어 알고리즘

빨강색 : binary search 활용

파란색 : binary search, unique set 활용

노란색 : binary search, unique set, two coin set 활용

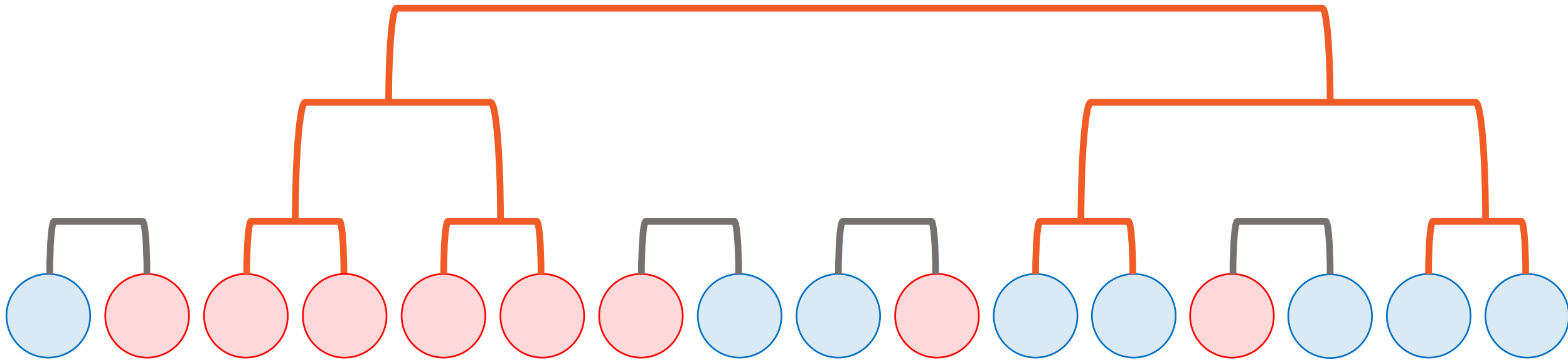


밥 알고리즘

1. 양팔에 각각 1개씩 동전을 올려서 무게 비교 - 총 50번 비교를 하고나서 시작! (50번의 balance()는 default)
2. 무게가 **다르면** 가벼운 건 가벼운 동전 그룹에, 무거운 건 무거운 동전 그룹에 담는다.
3. 무게가 **같다면** 양 쪽 동전을 한 쪽으로 모아 또 다른 무게가 같은 동전 2개를 반대에 올린다.
4. 이렇게 스텝사이즈를 키워가며 비교해 나간다.



밥 알고리즘은 가짜 동전과 진짜 동전의 갯수가 비슷할 때 잘 동작합니다.



연어덮밥 알고리즘

최종 성능 분석

알고리즘 성능 분석 - simulator

```
import subprocess

if __name__ == '__main__':
    p_start = 81
    p_end = 99

    epoches = 200

    file_name = "test8.csv"

    method_type = "merge2"

    for p in range(p_start, p_end+1):
        for _ in range(epoches):
            # exe 파일 경로
            subprocess.call(["C:\\Users\\gudfl\\source\\repos\\Algorithm_Project\\Debug\\Algorithm_Project.exe",
                             str(p), method_type, file_name], shell=True)

            file = open("C:\\Users\\gudfl\\Desktop\\test\\" + file_name, "at")

            file.write("\n")
            file.close()
```

- P값 하나에 200회씩 실행
- 200회의 평균 도출
- 99개의 P값에 대해 평균 조사
- 총 19800회의 시뮬레이션



알고리즘 성능 분석 - visualizer

```
df['mean2'] = df2['mean']
df['mean3'] = df3['mean']
array = df.values

x = [a for a in range(1, 100) for _ in range(0, 200)]

freq_df1 = [[0]*130 for _ in range(99)]
freq_df2 = [[0]*130 for _ in range(99)]
freq_df3 = [[0]*130 for _ in range(99)]

for i, val in enumerate(point_df1):
    freq_df1[int(i/200)][val] += 2

for i, val in enumerate(point_df1):
    freq_df2[int(i/200)][val] += 2

for i, val in enumerate(point_df1):
    freq_df3[int(i/200)][val] += 2

fig, axes = plt.subplots(nrows=2, ncols=2)
plt.setp(axes, xticks=range(0, 101, 10), yticks=range(0, 101, 10))

axes[0][0].plot(df3.index, df3['25'], c='r')
axes[0][0].plot(df3.index, df3['50'], c='r')
axes[0][0].plot(df3.index, df3['75'], c='r')

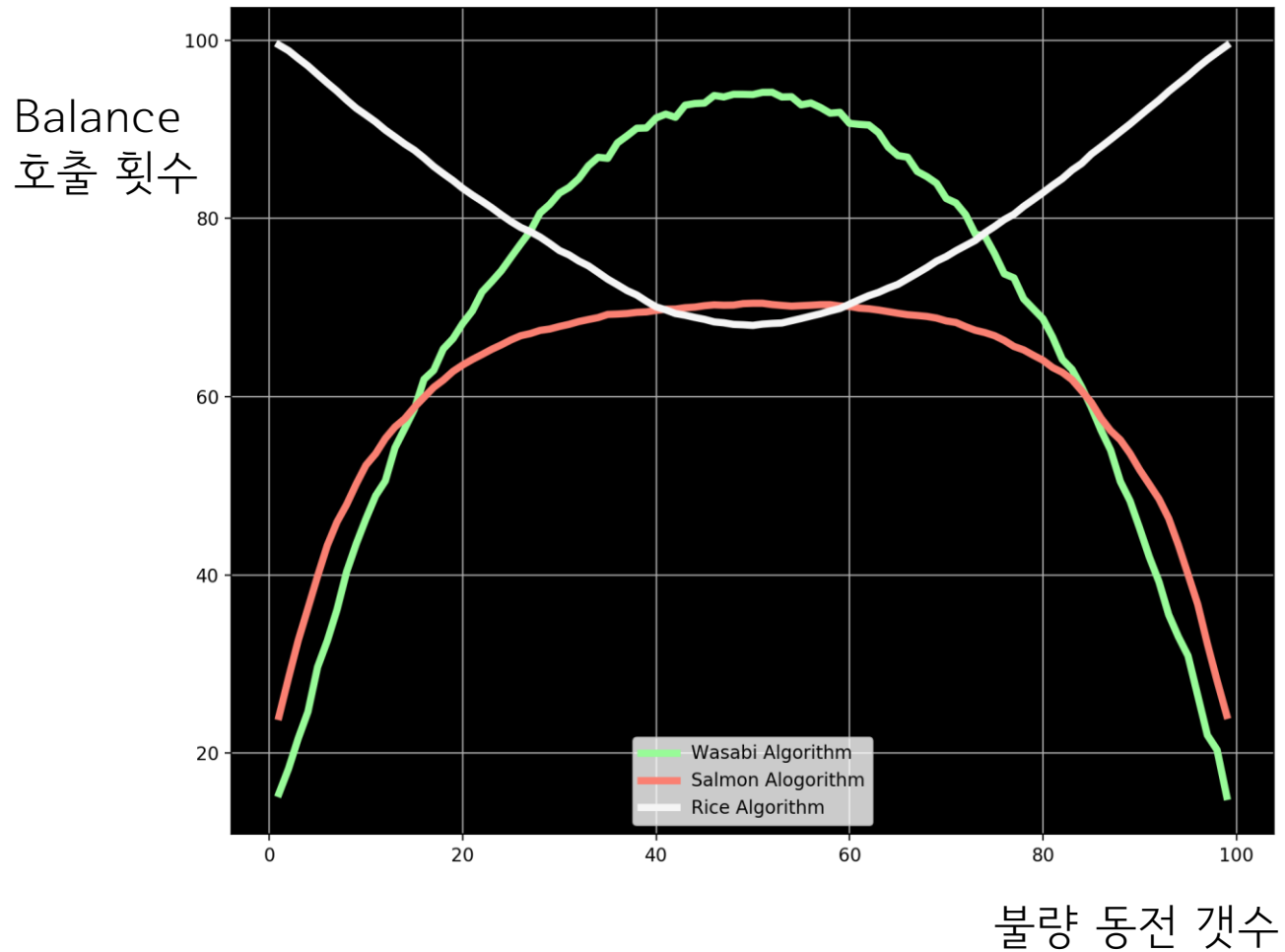
axes[0][0].plot(df2.index, df2['25'], c='b')
axes[0][0].plot(df2.index, df2['50'], c='b')
axes[0][0].plot(df2.index, df2['75'], c='b')
|
axes[0][0].plot(df1.index, df1['25'], c='g')
axes[0][0].plot(df1.index, df1['50'], c='g')
axes[0][0].plot(df1.index, df1['75'], c='g')

axes[0][1].scatter(x, point_df3, alpha=0.3, c='r', s=freq_df2)
axes[0][1].set_title("merge_1")
axes[0][1].plot(df3.index, df3['min'])
axes[0][1].plot(df3.index, df3['25'])
axes[0][1].plot(df3.index, df3['50'])
axes[0][1].plot(df3.index, df3['75'])
axes[0][1].plot(df3.index, df3['max'])

axes[1][0].scatter(x, point_df2, alpha=0.3, c='g', s=freq_df2)
axes[1][0].set_title("step")
```

- 파이썬 코드
- Simulator에서 얻어진 값들을 이용
- 알고리즘 간 비교가 쉽도록 시각화

알고리즘 성능 분석 - result



연어 + 와사비 + 밥



감사합니다.