- Bloom Filter의 false positive 비율이 의도한 대로 잘 나오는지 확인하세요.

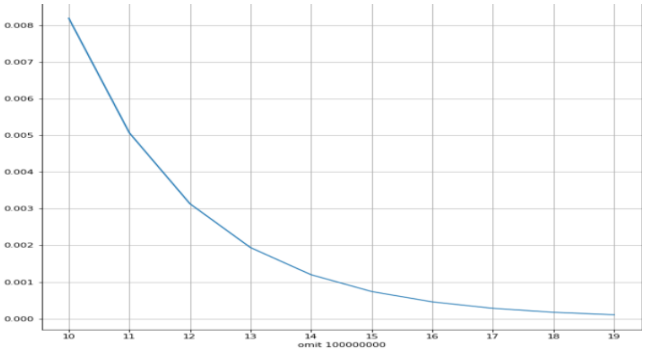
```
string = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890"
prob = 0.1
bloom = BloomFilter(10000, prob)
arr = []
for i in range(10000) :
    a = string[random.randint(0,len(string)-1)]
    b = string[random.randint(0,len(string)-1)]
    c = string[random.randint(0,len(string)-1)]
    d = string[random.randint(0,len(string)-1)]
    e = string[random.randint(0,len(string)-1)]
    f = string[random.randint(0,len(string)-1)]
    s = a+b+c+d+e+f
    bloom.put(s)
    arr.append(s)
```

```
h6b True
yvy True
Random
false positive 비율: 0.1 true: 979 false: 9021
PS C:\Users\pbk54\Desktop\3학년1학기\빅최기>
```

필터에 넣지 않은 랜덤한 문자열을 10000번을 Bloom Filter에 넣고 확인해본 결과 false positive한 횟수가 979번이 나온 것을 확인 할 수 있고 이는 의도한 비율인 0.1과 979./10000과 비슷한 결과라고 볼 수 있다.

- 1억명의 사용자 계정이 시스템에 저장되어있고, 사용자가 회원가입 중에 동일한 계정명이 서버에 존재하는지 즉각 확인해주는 시스템을 개발하려고 합니다. 이 때, Bloom Filter를 어떻게 활용하면 좋을까요? 비트배열의 크기, false positive 값 등은 어떻게 설정하면 적절할까요? 생각을 서술해보세요.

Bloom Filter을 이용하여 사용자가 회원가입 중에 새로운 아이디를 입력하면 Bloom Filter을 이용하여 기존에 있는 아이디면 즉각적으로 존재하는 아이디라고 나타내주는 시스템으로 이용하면 될 것 같다. 최적의 해시함수의 개수인 k가 최적화가 되면 ln(2) * n/m 값이 된다. 저는 새로운 아이디에 대한 false positive의 비율이 0.001정도면 충분히 괜찮다고 생각을 했다. 비트배열의 크기를 1억씩 증가시켜 false positive 값을 시각화 해본 결과. 아래와 같이 나왔다



이 때 false positive값은 0.000741값이 나왔으며 ln2 값은 0.69314718056라고 설정을하고 계산을 하였다.

- Bloom Filter를 사용하면 좋을만한 상황을 얘기하고, 왜 그러한 상황에서 Bloom Filter가 도움이 되는지 설명해보세요.

Bloom filter의 장점은 false negative가 없고 memory를 상대적으로 적게 사용한다. 이런 Bloom filter를 도메인 네임을 생성할 때 사용하면 좋을 것 같다. 현재 다양한 사이트 주소나 도메인 들이 있어서 새로운 도메인을 만들 때 이게 이미 존재하는지 중복여부를 확인해주면 좋을 것 같다.

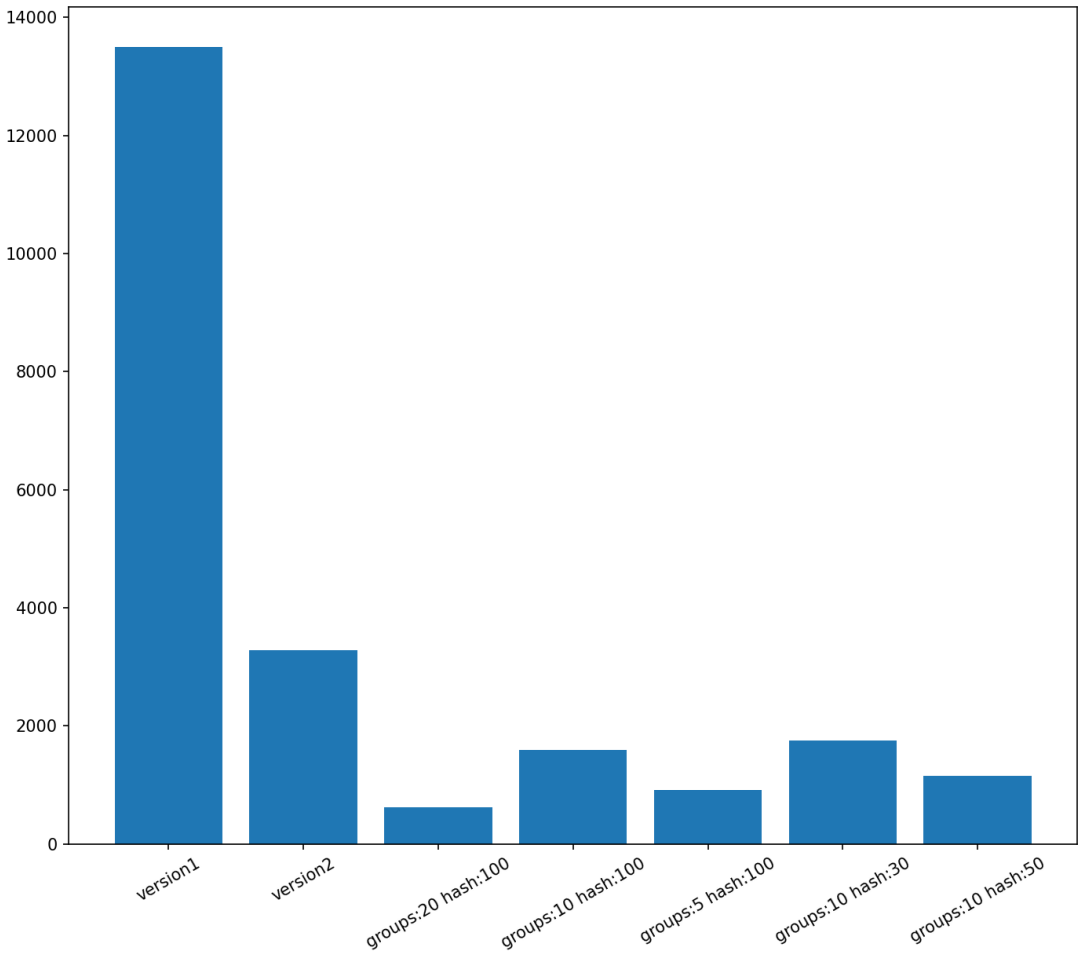
왜냐하면 Bloom Filter은 이미 존재하는 도메인에 대해서는 존재한다고 무조건 나타내지고 종종 false positive한 결과가 있긴 하지만 Bloom Filter는 빠르고 상대적으로 메모리도 적게 쓰기 때문에 이러한 상황에 적합할 것 같다. 이와 유사하게 휴대전화 번호, 차량 번호 등을 만들 때에도 false negative 한 상황이 발생하면 안되기 때문에 이러할 때도 Bloom Filter 시스템을 이용하면 좋을 것 같다.

- Version 1도 구현해보고, Version 1과 2의 정확도를 비교해보세요.

- hash function을 여러개 사용하고 '중앙 값 평균내기' 기법을 활용하여 Version 2의 정확도를 올려보세요.

- 사용하는 hash function의 수, group의 수 등을 조절해가며 정확도가 어떻게 변화하는지 확인해보세요.

Version1, version2와 group와 hash함수의 수를 조절해가면서 비교해본 결과 (RMSE로 오차 계산)

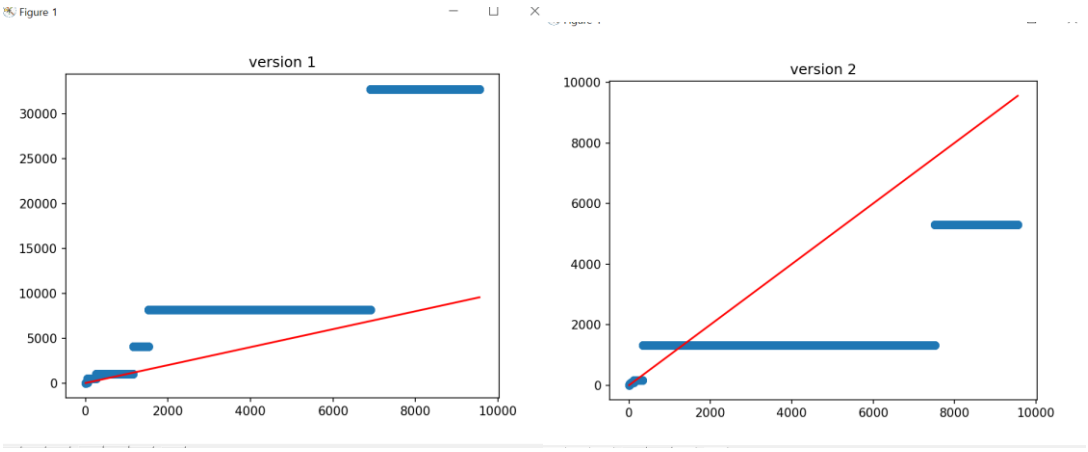


이러한 결과가 나왔다. 여기서 groups와 hash는 각각 group와 hash함수의 수를 의미한다.

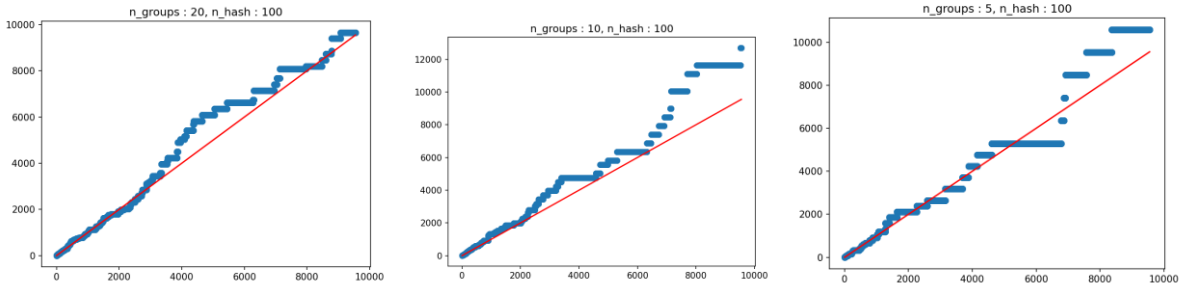
Version1에서는 값이 튀면 정확도가 매우 낮아지므로 오차값이 계속 변동이 크게 있었다.

각각의 경우를 시각화를 통해 나타내면

- Version 1 Version 2

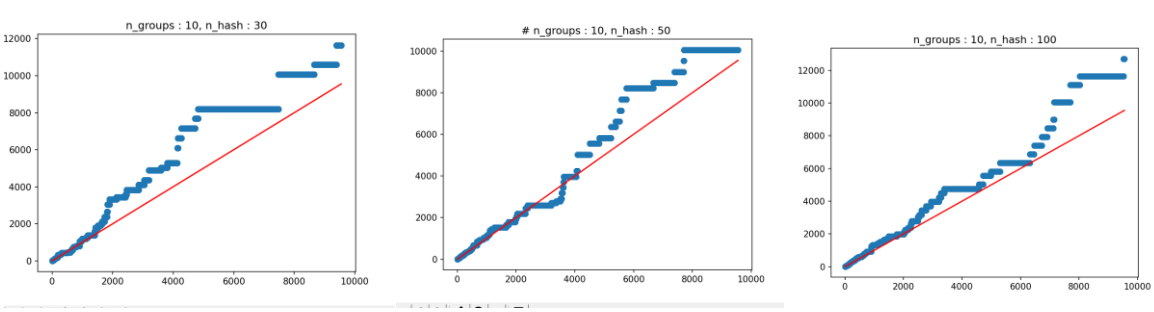


그룹의 수를 변화시켜본 결과



그룹의 수를 줄이게 된 결과 평균적으로 정확도가 떨어짐을 확인 할 수 있고 그룹의 수가 적어질수록 그래프가 양자화 되어있다고 볼 수 있다.

해시함수의 수를 변화시켜본 결과



일반적으로 해시함수의 수가 증가할수록 일반적으로 더 정확한 값이 나옴을 확인 할 수 있다. 결론적으로 적당한 그룹의 수와 해시함수의 수를 설정을 하면 예측값이 정확하게 나오는 것을 확인 할 수 있다.