Initializing...

Do M=1000
Array: 0.0006809234619140625
Sorted Array: 0.0006809234619140625
Linear Probing: 0.06842803955078125
Quadratic Probing: 0.0035660266876220703

Do M=2000
Array: 0.00048279762268066406
Sorted Array: 0.0004808902740478515 6
Linear Probing: 0.05622601509094238
Quadratic Probing: 0.004364013671875

Do M=3000
Array: 0.0004839897155761719
Sorted Array: 0.0004830360412597656
Linear Probing: 0.07336902618408203
Quadratic Probing: 0.005028724670410156

Do M=4000
Array: 0.0005779266357421875
Sorted Array: 0.0006341934204101562
Linear Probing: 0.1101236343383789
Quadratic Probing: 0.005839824676513672

Do M=5000
Array: 0.0004742145538330078
Sorted Array: 0.00046896934509277344
Linear Probing: 0.089508056640625
Quadratic Probing: 0.00734710693359375

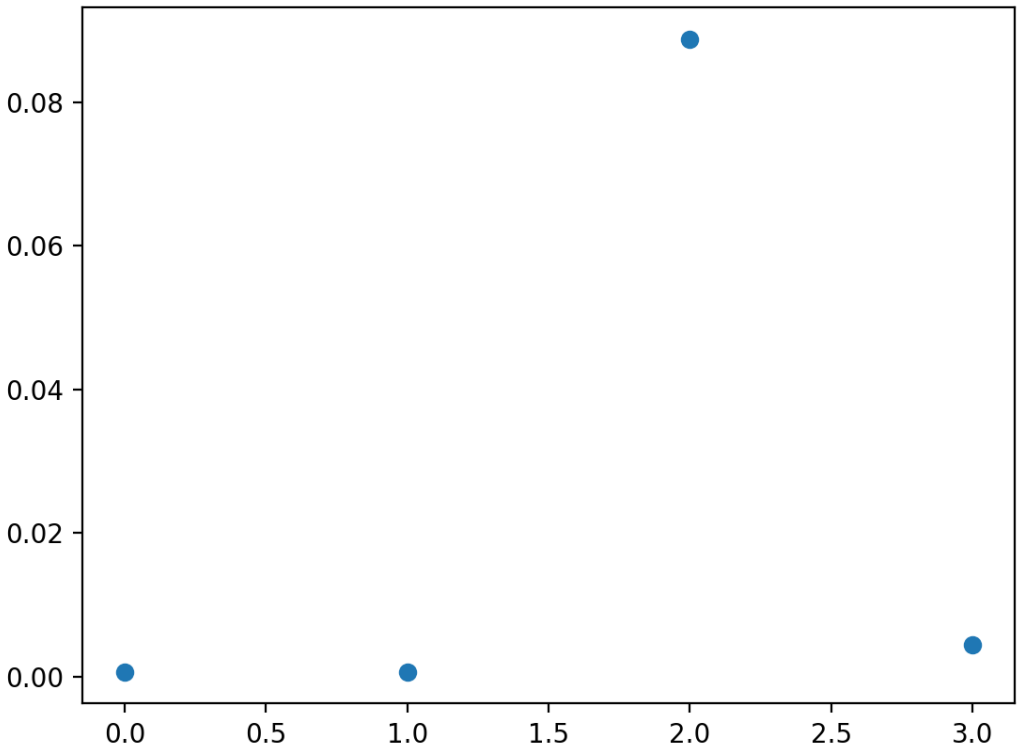
Do M=6000
Array: 0.0005731582641601562
Sorted Array: 0.0005660057067871094
Linear Probing: 0.1823580265045166
Quadratic Probing: 0.0069580078125

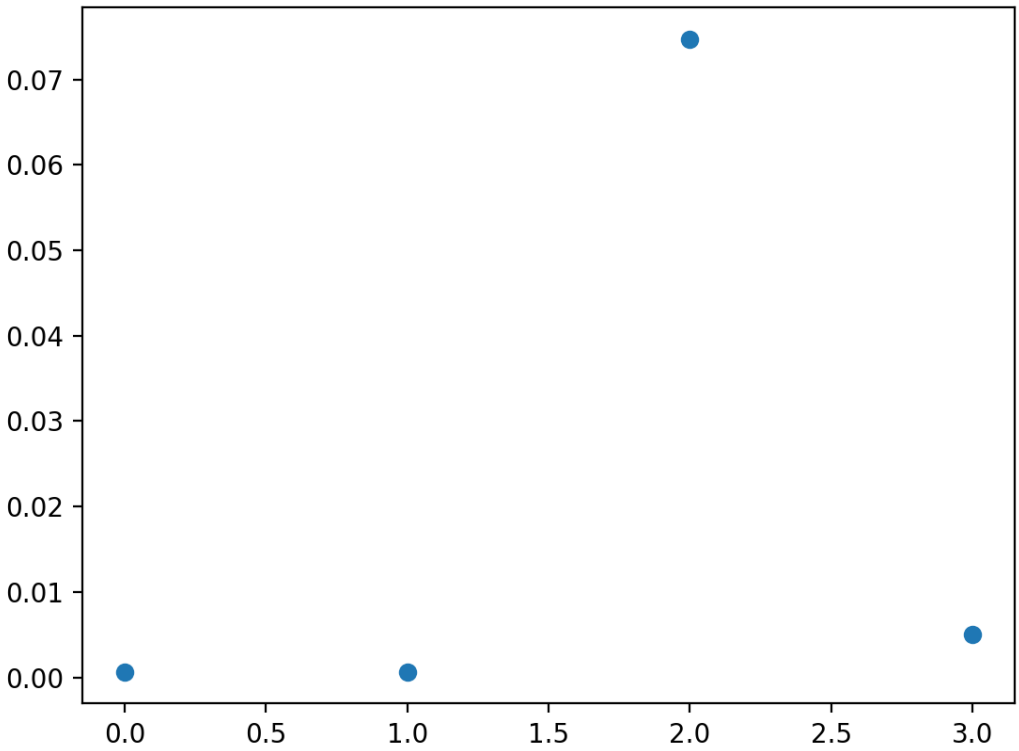
Do M=7000
Array: 0.0004858970642089844
Sorted Array: 0.0004820823669433594
Linear Probing: 0.23624110221862793
Quadratic Probing: 0.007470130920410156

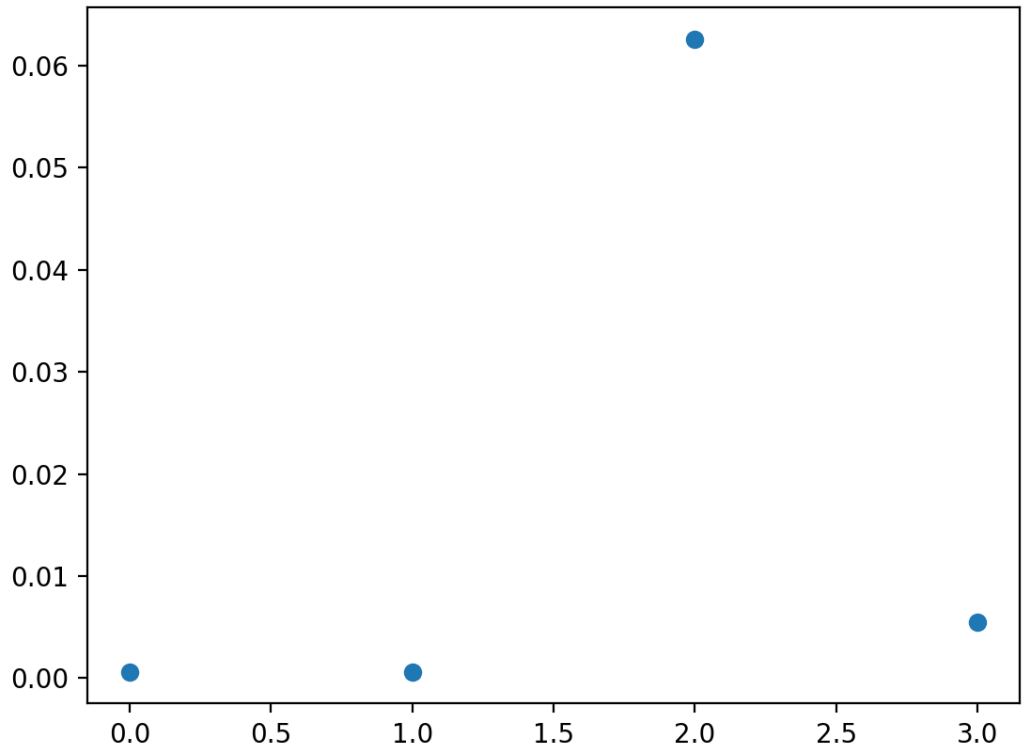
Do M=8000
Array: 0.0004711151123046875
Sorted Array: 0.0004680156707763672
Linear Probing: 0.2980632781982422

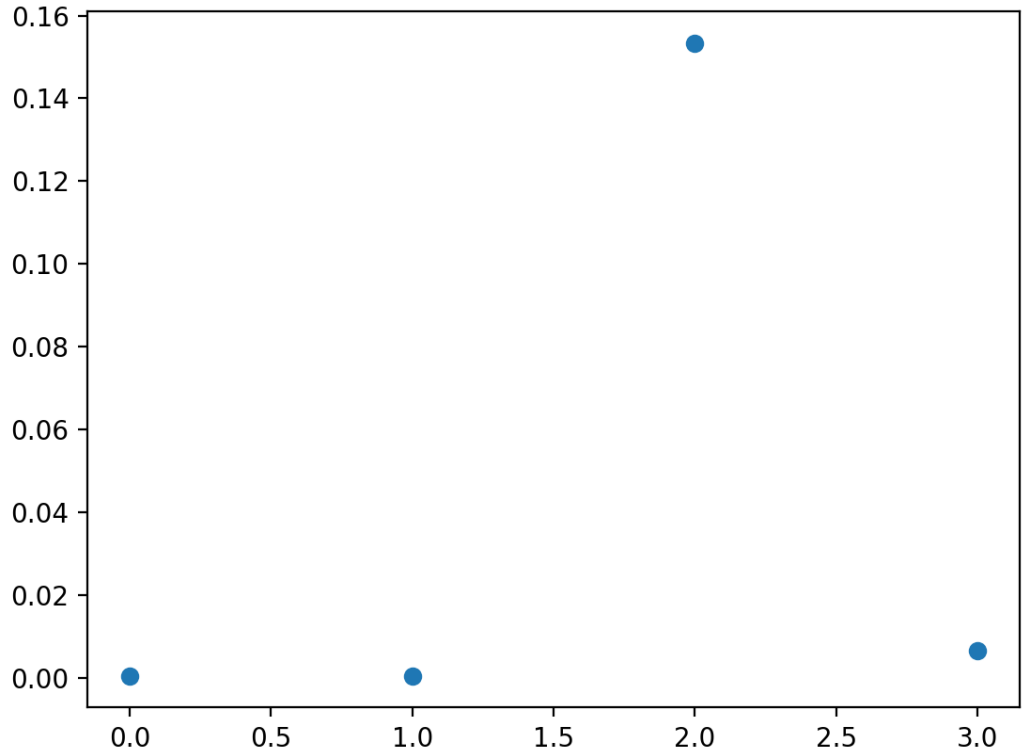
Quadratic Probing: 0.0072710514068603516

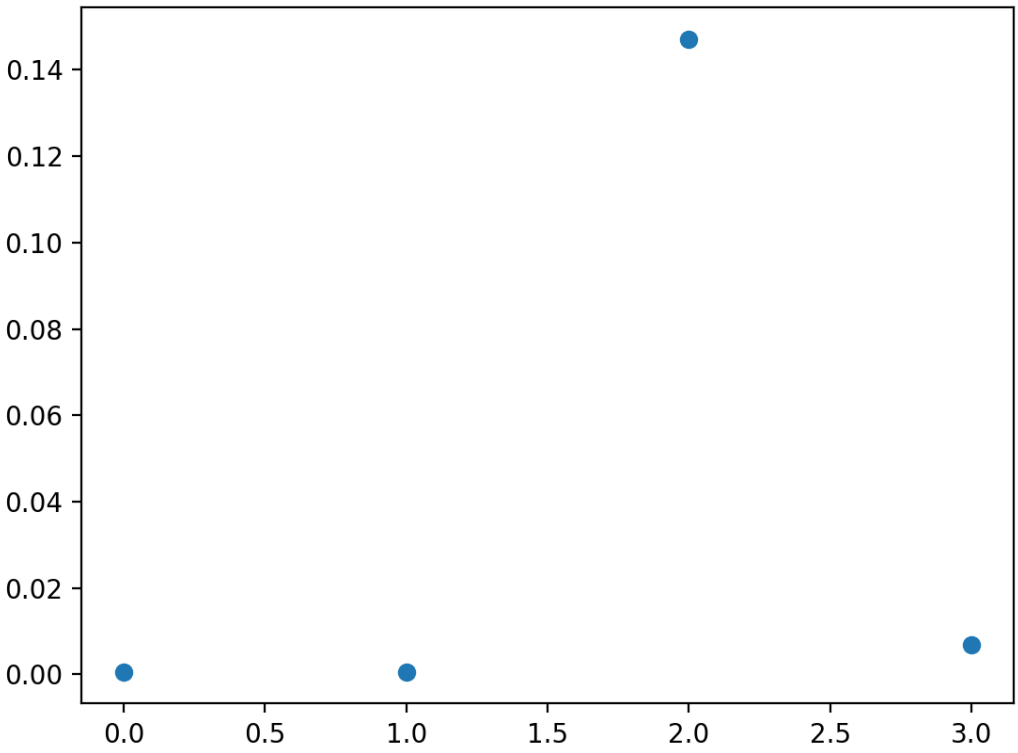
Do M=9000
Array: 0.00045418739318847656
Sorted Array: 0.00045108795166015625
Linear Probing: 0.37448883056640625
Quadratic Probing: 0.009560823440551758

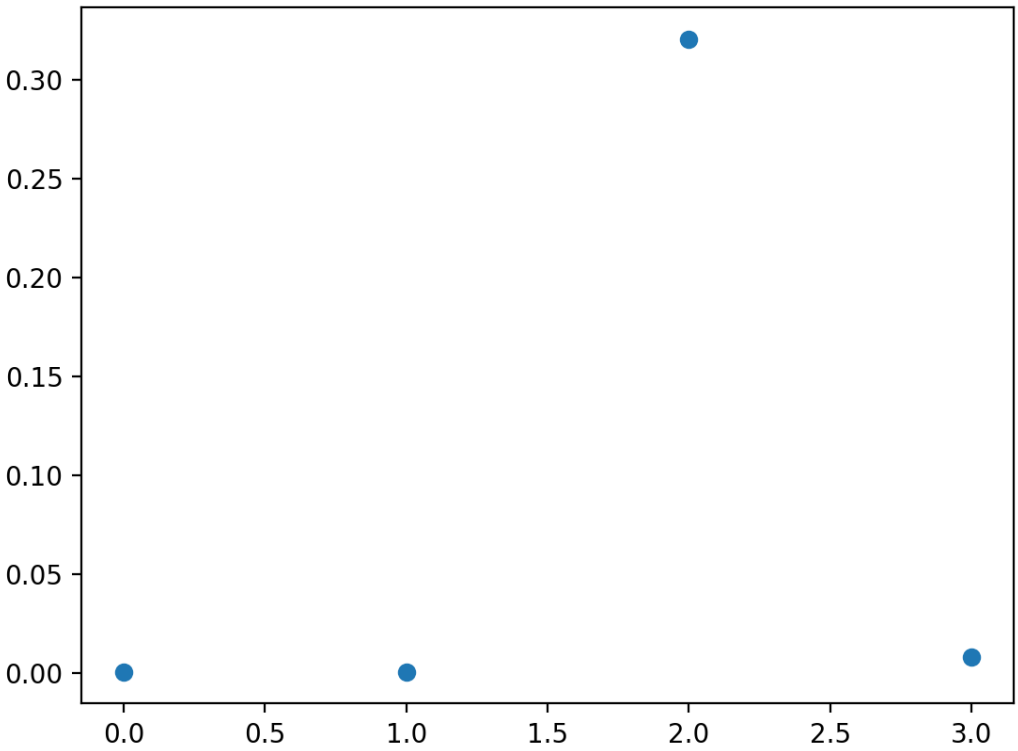

0.08
0.06
0.04
0.02
0.00
0.0
0.5
1.0
1.5
2.0
2.5
3.0

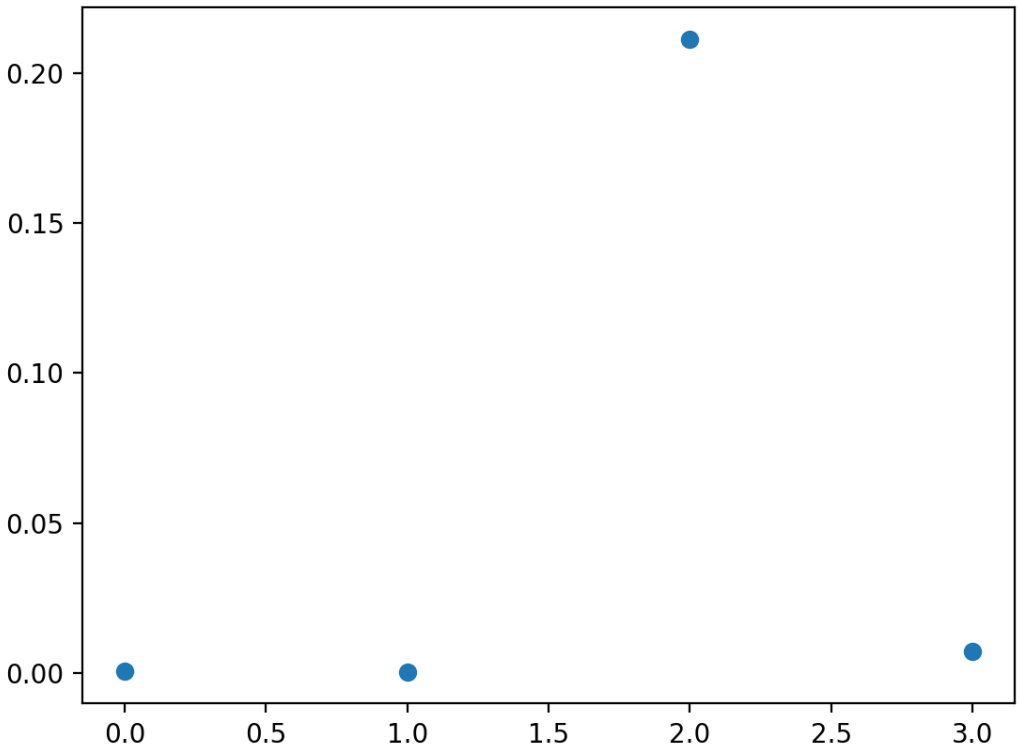

0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0.0
0.5
1.0
1.5
2.0
2.5
3.0

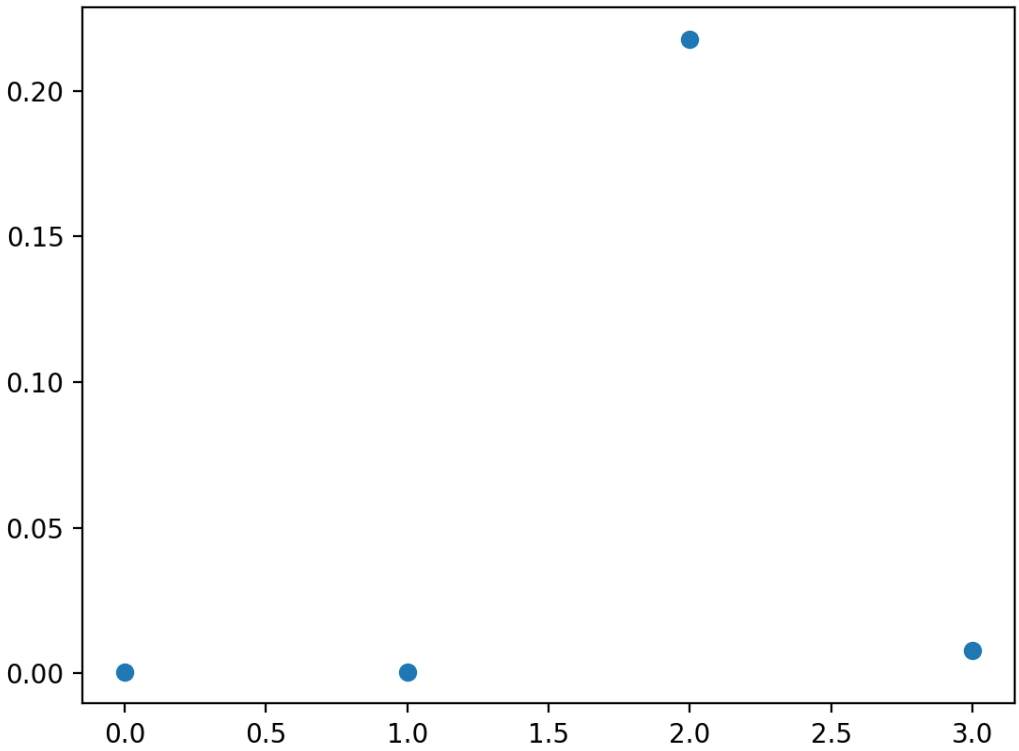

0.06
0.05
0.04
0.03
0.02
0.01
0.00
0.0
0.5
1.0
1.5
2.0
2.5
3.0

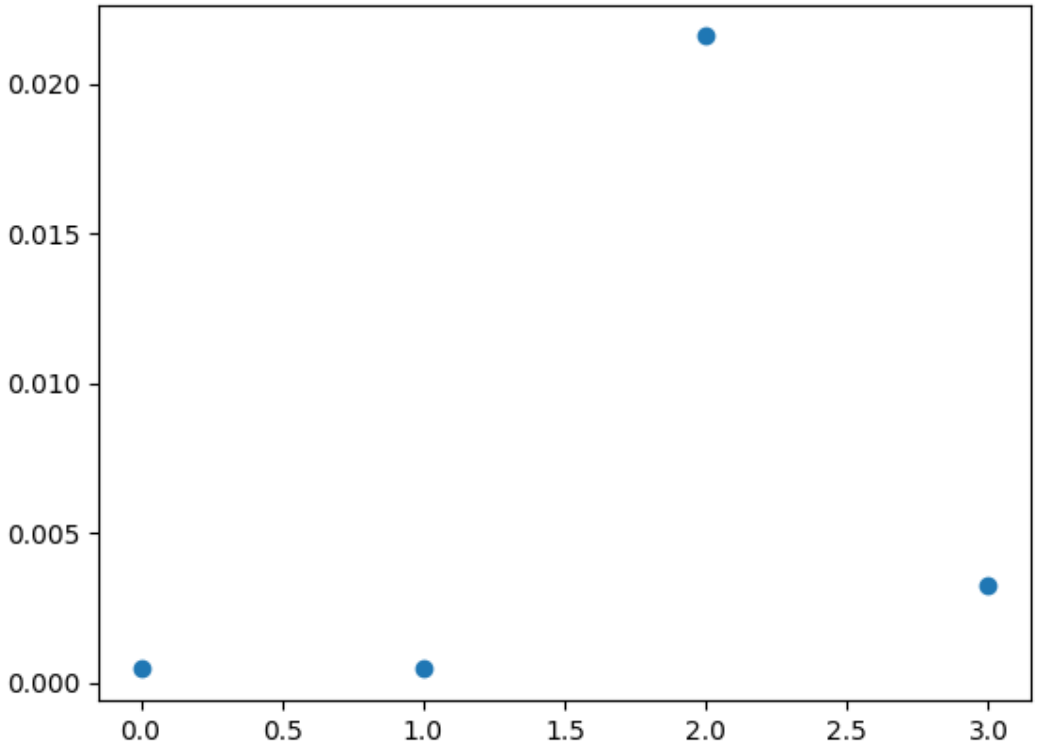

0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0.0
0.5
1.0
1.5
2.0
2.5
3.0


0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0.0
0.5
1.0
1.5
2.0
2.5
3.0


0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.0
0.5
1.0
1.5
2.0
2.5
3.0


0.20
0.15
0.10
0.05
0.00
0.0
0.5
1.0
1.5
2.0
2.5
3.0


0.20
0.15
0.10
0.05
0.00
0.0
0.5
1.0
1.5
2.0
2.5
3.0



맨 왼쪽부터 정렬되지 않은 배열, 정렬된 배열, 선형 탐색, 제곱 탐색 을 활용한 검색시간이다.

[선형 탐색에 사용된 해싱 함수]

```
def hash(self, key):
    index = key % self.size

    if self.hashtable[index] == None:
        return index
    else:
        while self.hashtable[index] != None:
            index = (index + 1) % self.size

        return index
```

충돌이 일어날 경우에 인덱스를 1씩 증가시키며 탐색한다.

[제곱 탐색에 사용된 해싱 함수]

```
def hash(self, key):
    index = key % self.size

    if self.hashtable[index] == None:
        return index
    else:
        i = 1
        while self.hashtable[(index + (i * i)) % self.size] != None:
            i += 1

        return (index + (i * i)) % self.size
```

충돌이 일어날 경우에 인덱스를 제곱씩 증가시키며 탐색한다.

[성능에 대한 평가]

우선 위에서 측정한 데이터는 삽입과 탐색에서 해싱시간을 포함한 결과이다.
해싱 테이블에서 값을 삽입하고 검색하는데 소요되는 시간 복잡도는 O(1)이지만, 해싱 함수를 통한 key를 생성하는데 소요되는 시간은 최악의 경우 전체 해시를 탐색해야하므로 배열과 같다.