实验一：验证 List 的按索引取值确实是 O(1)

代码：

```
import timeit
x=list(range(1000001))#创建列表
for i in range(10000,1000001,10000):
    n1 = timeit.Timer("x[%d]"%i, "from __main__ import x")#按[n]索引
    t=n1.timeit(number=1000)#重复 1000 次计时
    print("%d %.9f"%(i,t))#输出 i 值及对应时间
```

实验结果：

如图，当 n 值变化时，list 按索引取值的运行时间（1000 次）基本在 0.00006s 上下浮动，其复杂度应为 O(1)。

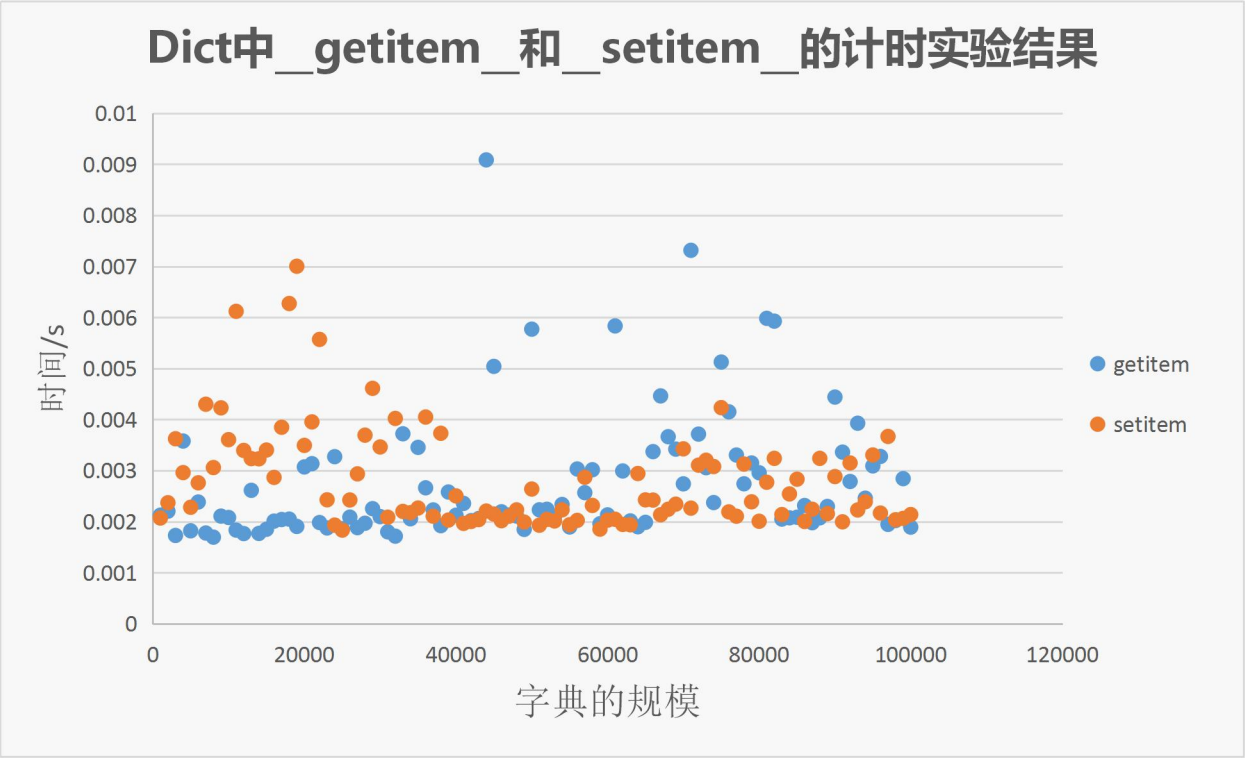
实验二：验证 Dict 的 get item 和 set item 都是 O(1)的

代码：

```
import random
import timeit
for i in range(1000,100001,1000):
    x = {j: j for j in range(i)}#创建字典
    getitem = timeit.Timer("x.__getitem__(random.randrange(%d))" % i, "from
__main__ import random,x")
    setitem = timeit.Timer("x.__setitem__(random.randrange(%d),1)" % i, "from
__main__ import random,x")
    gt = getitem.timeit(number=1000)
```

```
    st = setitem.timeit(number=1000)#分别计时
    print ("%d %5f %5f"%(i,gt,st))#依次输出 i 值和两个时间
```

实验结果：



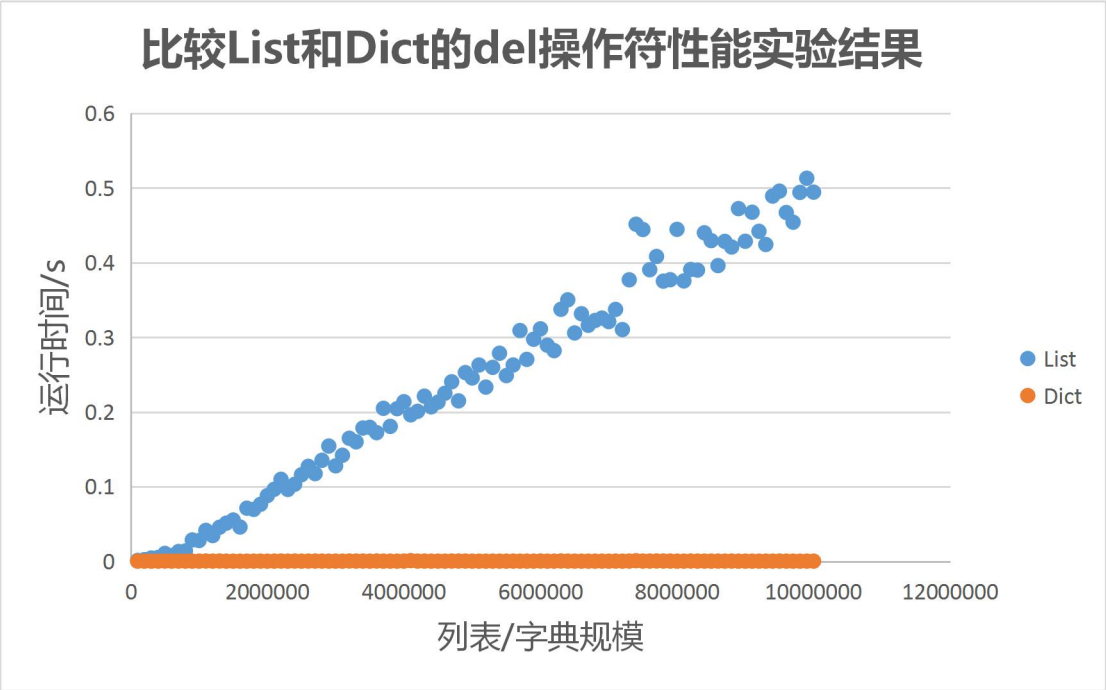
如图，当字典规模变化时，__getitem__和__setitem__的运行时间（1000 次）基本在 0.002s 上下浮动，其复杂度应为 O(1)。

实验三：比较 List 和 Dict 的 del 操作符性能

代码：

```
import random
import timeit
for i in range(100000,10000001,100000):
    l=(list(range(i)))#创建列表
    d = {j: j for j in range(i)}#创建字典
    delist = timeit.Timer("del l[random.randrange(%d)]" % i, "from __main__ import random,l")
    deldic = timeit.Timer("del d[random.randrange(%d)]" % i, "from __main__ import random,d")
    dl = delist.timeit(number=100)
    dd = deldic.timeit(number=100)#分别计时，注意 number 与 i 的比例适中，避免删重
    print ("%d %5f %5f"%(i,dl,dd))#依次输出 i 值和两个时间
```

实验结果：



如图，当 i 递增时，Dict 的 del 运行时间（100 次）基本在 0.00025s 上下浮动，其复杂度应为 O(1)；而 List 的 del 运行时间线性增加，其复杂度应为 O(n)，当 i 较大时，时间远远高于 Dict 的 del。综上，Dict 的 del 操作符性能要优于 List 的 del 操作符，且数据规模越大，优越性越显著。