# Zig-Zag-Sort 测试文档

**以下变量均会出现在文档中, 用以说明变量用意**

| 索引 | 相关变量 | 变量说明 |
| --- | --- | --- |
| 1.1, 1.2 | k | 由于测试数组Size都为2^x次方, 用k表示当前测试数组为2的次方数 |
| 1.1, 1.2 | distribution | 原型为uniform_int_distribution， 为使生成所用到的测试数据均匀分布 |
| 1.1, 1.2 | engine | 原型为default_random_engine(time(0))， 为测试使用到的随机数引擎 |
| 1.1 | caseTimes | 表示测试次数 |
| 1.1, 1.2, 2.1 | ArrayLength | 表示当前测试样例的数组长度 |
| 1.1, 1.2 | preArray | 表示当前测试数组排序前的序列 |
| 1.1, 1.2, 2.1 | sortedArray | 表示当前测试数组使用Zig-Zag-Sort后的序列 |
| 1.1, 1.2 | builtInSortedArray | 表示当前测试数组使用内置Sort(快排)生成的序列 |
| 1.1, 1.2 | upperK | 表示为k的上界 |
| 2.1 | FirstArray | 表示全排列测试的首个全排列(有序) |

# 1. 随机测试

## 1.1 完全随机测试

### 测试目的：验证Zig-Zag-Sort排序算法的正确性

### 测试方法：

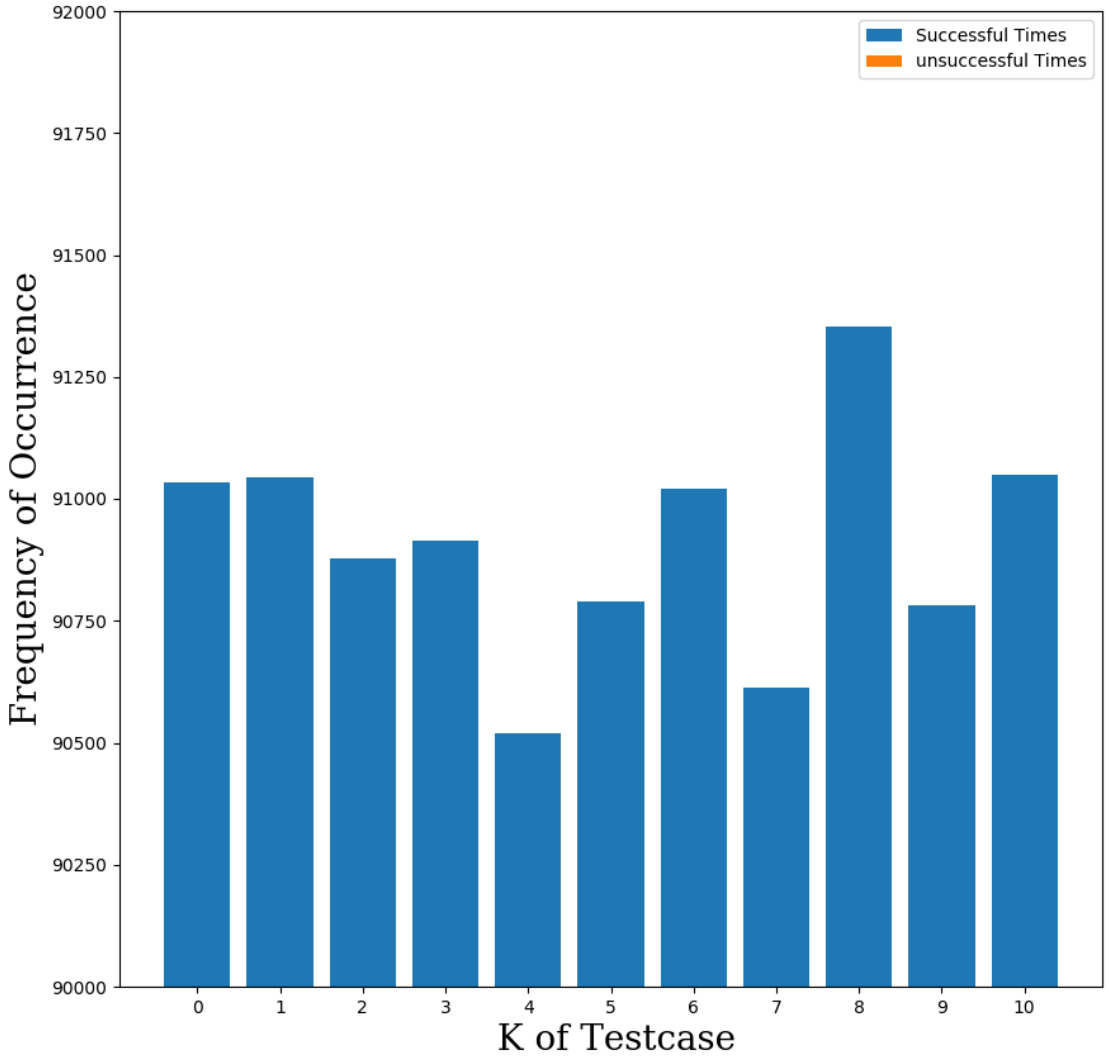
1. 确定Zig-Zag-Sort测试次数caseTimes与k的上限值upperK
2. 进入测试次数循环,直到caseTimes <= 0
3. 首先根据distribution（engine）生成一个当前测试样例的Length
4. 生成preArray, 同时用distribution（engine）preArray填充至Length
5. 分别执行两种排序排序算法得到sortedArray与builtInSortedArray, 比较sortedArray与

builtInSortedArray是否一致, 如果是, 则测试成功, 否则记录下当前样例.

6. 如果caseTimes仍 > 0 , 回到步骤2, 否则结束当前程序

### 测试结果

- 由于该测试只为确定Zig-Zag-Sort排序的正确性, 故没有记录相应的排序时间.实际测试中upperK定为10, 即最大只会生成长度为2^10长的测试数组, 在测试的1,000,000样例中通过比较sortedArray与builtInSortedArray, 发现两者匹配程度率确实达到100%
- 相应的测试数组长度分布如下图所示




## 1.2 半随机测试

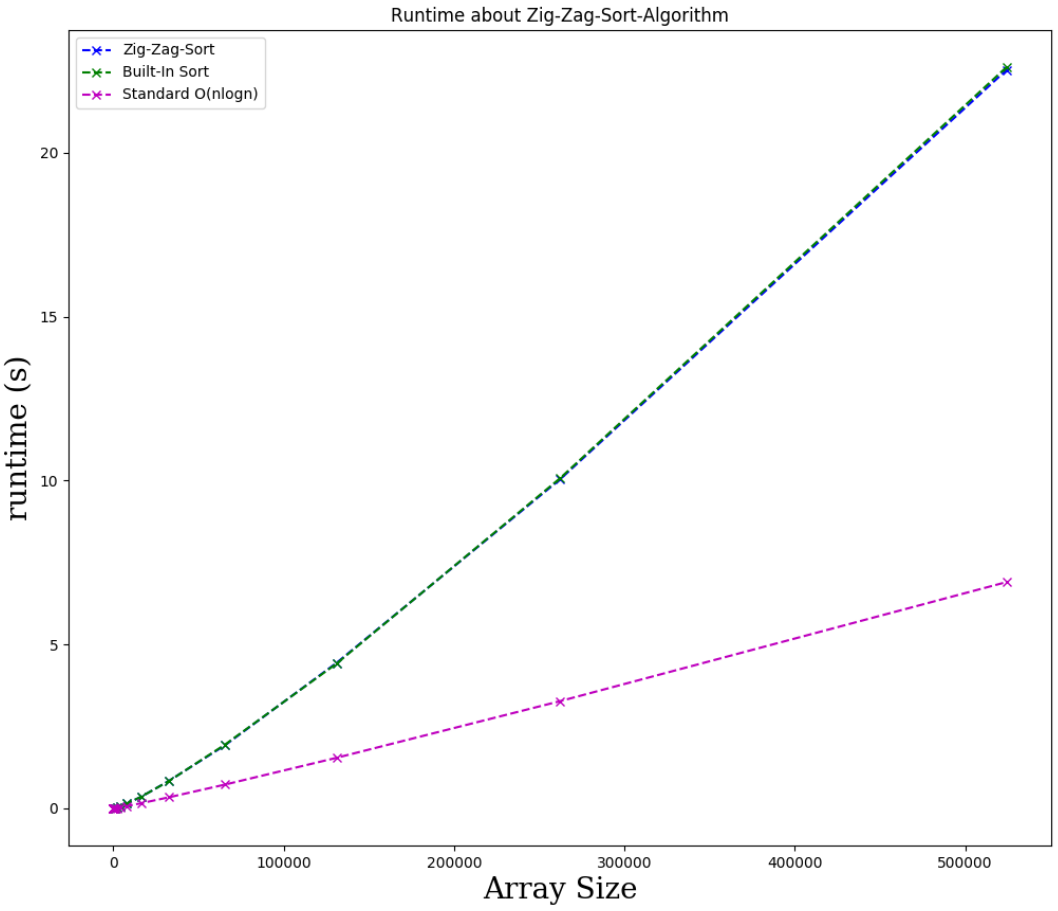
**测试目的： 在验证Zig-Zag-Sort正确性的同时, 记录相应的运行时间, 验证是否为O(nlgn)算法**

**测试方法：**

1. 确定k的上界upperK, 已确定循环次数
2. 循环从k = 1 -> upperK, 每次生成长度为2^k长的数组
3. 生成preArray, 同时用distribution（engine）preArray填充至ArrayLength
4. 分别执行两种排序排序算法得到sortedArray与builtInSortedArray, 比较sortedArray与builtInSortedArray是否一致, 如果是, 则测试成功, 并记录下两种排序分别使用的时间. 否则记录下当前样例.
5. 如果k <= upperK, 回到步骤2, 否则结束当前程序

### 测试结果

- 在实际测试中选取upperK为20, 即测试数组最大长度为2^20, 由下图可以看出两种不同排序算法所需要的时间几乎一致, 并且与标准的O(nlogn)只有常数倍的差距




# 2. permutation测试

## 2.1 AllPermutation测试

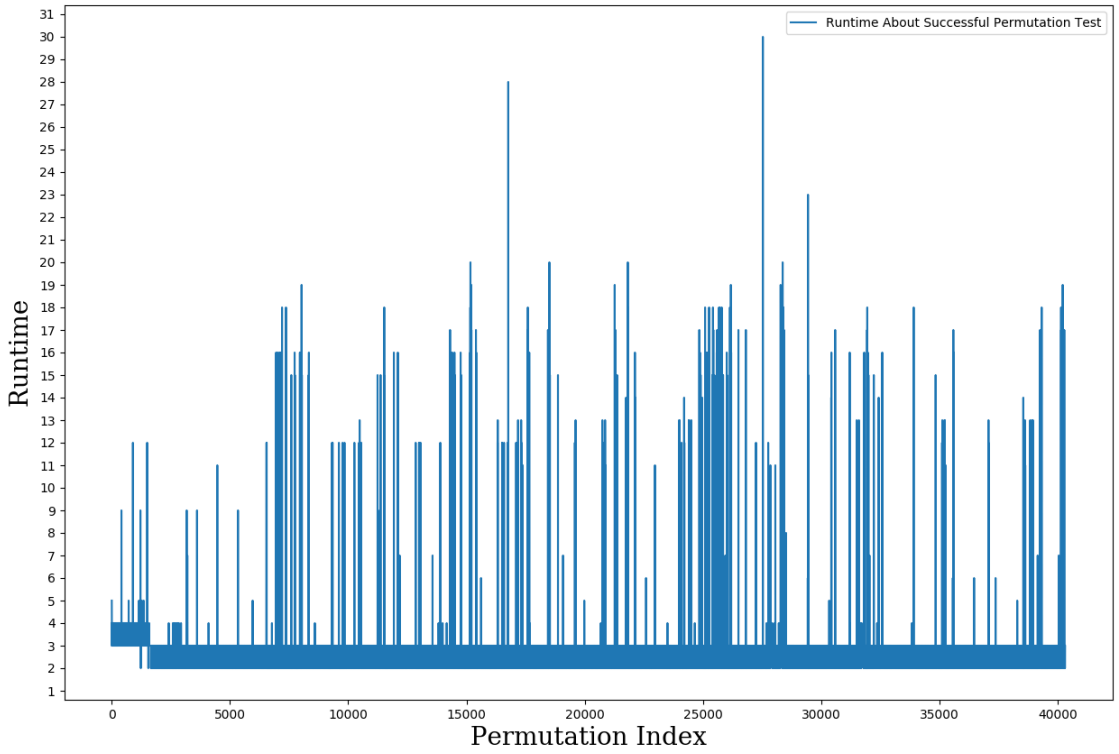
### 测试目的：验证Zig-Zag-Sort在同一序列,不同排列的情况下验证是否得到相同的结果

### 测试方法：

1. 生成长度为ArrayLength的preArray, 同时第一次的preArray定为FirstArray
2. 调用next_permutation生成preArray的下一个排列, 循环至调用next_permutation后 == firstArray时循环终止
3. 对当前生成的排列preArray使用Zig-Zag-Sort, 得到数组sortedArray, 比较sortedArray是否与firstArray相等, 是则排序成功, 记录下运行时间, 否则排序失败.
4. 生成下一个排序, 若生成后preArray == firstArray程序终止, 否则回到2进行下一个排列比较

### 测试结果：

- 在实际中对长度为8的数组生成全排列, 共有8! = 40320种排列, 最终这这40320排列通过Zig-Zag-Sort生成的序列都为相同的序列且有序.分别运行时间统计图如下所示, 大部分排列所需的运行时间基本稳定在2s-4s间,




## 2.2 0-1 测试

### 测试目的：验证Zig-Zag-Sort在较长的长度中仍能保持permutation的稳定性

### 测试方法

1. 由于2.1中的AllPermutation测试在k >=4 由于测试数据量过大而导致需要耗费大量时间, 所以使用等价的0-1测试测试算法保帧
2. 算法优先确定k的lowerBound, 与upperBount, 作为外部循环的条件.

3. 生成长度为ArrayLength为2^k的数组FirstArray, 初始化为全0, 同时生成preSortArray.
4. 不断调用NextZeroOntSeq生成下一个0-1排列.
5. 调用Zig-Zag-Sort对生成的preSortArray进行排序, 看结果是否满足有序
6. 当FirstArray == preSortArray时, 证明当前k的排序已全部生成完 ++ k, 否则回到4
7. 当k > upperBound, 测试结束, 否则回到步骤3

## 测试结果

* 实际测试中k从1 - 10得到的排列均通过测试, 证明算法序列的所有排列中都保持了稳定性