Code Project 1: Report

PB14204015 张晗

1. **问题背景**

在一个由n个元素组成的集合中，第i个顺序统计量是指该集合第i小的元素。例如，在一个元素集合中，最小值是第一个顺序统计量i=1，最大值是指第n个顺序统计量i=n。

需要设计算法找出第i个顺序统计量

输入：一个包含n个数的集合A和一个整数，1≤i≤n.

输出：元素x∈A，且A 中恰好有i-1个其他元素小于它。

1. **算法思想**
   1. 随机选择算法

随机选择算法利用了快排的思想，只不过这里的partition函数利用随机生成的下标来决定主元，可以减小最坏情况出现的概率。

*step 1*: 随机生成主元，并对数组A进行randomizedPartition, 返回主元的下标q使得q之前的元素都小于A[q]，q之后的元素都大于A[q]。（这里的大于和小于对是否严格大于小于没有要求）。

*step 2*: 判断如果q前面还有i-1个元素，A[q]就是第i小的元素，返回。如果不是，分两种情况，一种是q前面的元素多于i-1个，则递归调用自身RadomSelect(p,q-1,i,A)

如果q前面元素少于i-1个，则递归调用自身在右半区寻找RadomSelect(q+1,r,i-(q-p+1),A)。

* 1. 最坏情况线性复杂度算法

此算法可以在最坏输入下保持O(n)的复杂度，这个保证来源于对随机选择算法的修改。在随机选择算法当中，主元的选取是由随机函数随机选取，这样在某些情况下仍然不能保证partition的均衡性。在最坏输入下，每一次分割仍然出现最大不均衡，这样算法的复杂度将达到O(n2)。

修改后的最坏情况线性复杂度算法( 在程序中我们暂时将它命名为WorstCaseLinearSelect)，通过几步运算计算出主元，以保证数组中一定比例的元素小于主元，一定比例的元素大于主元。这样，无论对于什么样的输入，partition的均衡性都能得到保证。这也是“最坏情况线性复杂度算法”名称的由来。

算法步骤如下：

*step1*: 将输入数组的n个元素划分为⌊n/5⌋组，每组5个元素，且之多只有一组由剩下的n mod 5个元素组成。

*step2* :寻找这⌈n/5⌉组中每一组的中位数；首先对每组元素进行插入排序，然后确定每组有序元素的中位数。

*step3*: 对第2步中找出的⌈n/5⌉个中位数，递归调用SELECT以找出其中位数X(约定是下中位数)。

*step4*: 利用修改过的PARTITION函数，按X对输入数组进行划分让k比低分区的元素多1，因此是第k小的元素，并且有n-k个元素在高分区。

*step5*: 如果i=k, 则返回X，如果i<k, 则在低分区递归调用SELECT来找出第i小的元素。如果i>k，则在高区递归查找第i-k小的元素。

1. **编程实现**

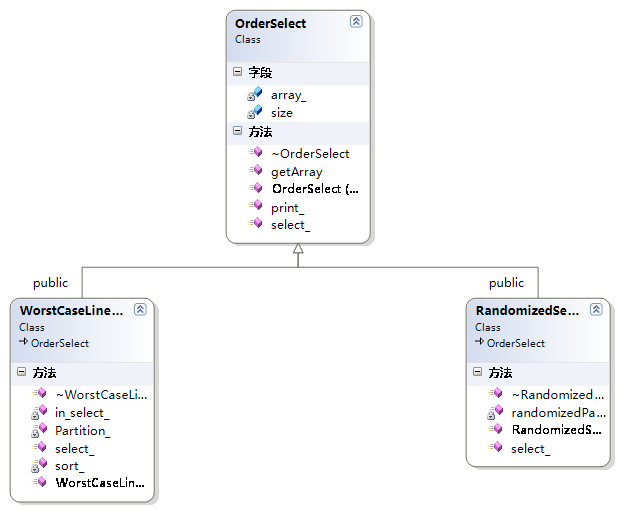
硬件环境: Intel Core i5 4210U

编程环境：Windows 10+ Visual Studio 2010

编程语言：C++

概述：本次实验采用C++面向对象的编程，编写基类OrderSelect负责初始化一个数组，虚函数int select\_(int p, int q, int i)作为公有类方法负责返回第i个顺序统计量。子类RandomizedSelect和WorstCaseLinearSelect继承自OrderSelect，各自的类方法int select\_来源于父类虚函数的多态，具体实现是分别调用上述两种算法来求取顺序统计量。主函数int main()是函数入口，根据用户输入提供两种模式，一种是现场生成满足要求的随机数组，分别利用两种算法返回顺序统计量并且打印这两种算法消耗时间；第二种模式是自动在当前目录下生成一个报告文件(.csv)，内含在不同数组规模k下两种算法的表现报告。

*相关类图*：



基类：OrderSelect

创建函数 OrderSelect(int s, int range): 输入元素个数以及随机数最大值range(0~range)创建随机数组，随机数生成取当前时间作为种子。

OrderSelect(int s, int \*ay): 输入元素个数，传入数组指针，从指针头开始逐个把外界传入的数组抄到成员变量array\_中。array\_定义为整形指针，采用malloc函数返回动态数组指针，并把这个指针变量赋给array\_

getArray(): 返回成员变量array\_，这是一个整形指针。

print\_(): 打印当前成员array\_所指向的数组。

virtual int select\_(int p, int r, int i): 虚函数，输入数组下标区间p, r 以及求第i小的元素，返回这个第i小的元素。

子类：RandomizedSelect

*int select\_(int p, int r, int i)*：利用随机选择算法求顺序统计量。

*int randomizedPartition(int p, int r)*: 给定区间p, r, 给出随机分割后的主元的位置q。

子类：WorstCaseLinearSelect

*int select\_(int p, int r, int i)*: 利用最差情况线性选择算法求出顺序统计量。这个函数是通过调用私有类方法 int in\_select\_(int p, int r, int i, int \* A)来实现的。这样做的目的是因为这个方法需要递归调用自身，为了程序的逻辑性，我们在每一步调用之前把求出的n个5元分组的中位数，抄到一个新的动态数组B中，并把这个B传入下一层递归调用。这个时候我们就需要in\_select\_函数，让select\_调用它来统一虚函数select\_的接口(select\_的传参不包含传入一个整形指针int \*A)。

*int in\_select\_(int p, int r, int i, int \*A)*: 先传参检查，如果元素个数少于等于5个，直接采用插入排序并直接返回顺序统计量。如果不是，动态分配一个新的数组B用来存储每一组返回的中位数，并把这个B传入下一层递归调用来求B的中位数。求到的中位数median作为参数传给Partition\_函数作为分割的主元。分割完了之后返回主元的位置，依据这个主元的位置做出和随机选择中select\_函数一样的处理机制，继续递归调用。

*int Partition\_(int p, int r, int \*A, int key)*：key为传入主元的值，依据这个值把区间p, r分成左右两区。注意，扫描过程中同时去寻找key作为主元所在的位置，因为上一部返回的median只是告诉了我们中位数的数值，并没有告诉我们它在数组中位置，所以我们要重新扫过寻找它，一旦发现了它，把它和数组最后一个元素A[r]交换，这样Partition\_函数就和之前的一样了。

主函数：

主函数分两种模式：

第一种按照需求生成随机数组，并且利用同一个数组动态初始化两个实例，一个是RandomizedSelect实例，一个是WorstCaseLinearSelect实例。分别对这两个实例调用select\_类方法，分别记录消耗的时间，最后打印在屏幕上。其中记录时间用的windows系统自带的QueryPerformanceCounter()函数，精度可以达到纳秒级别。

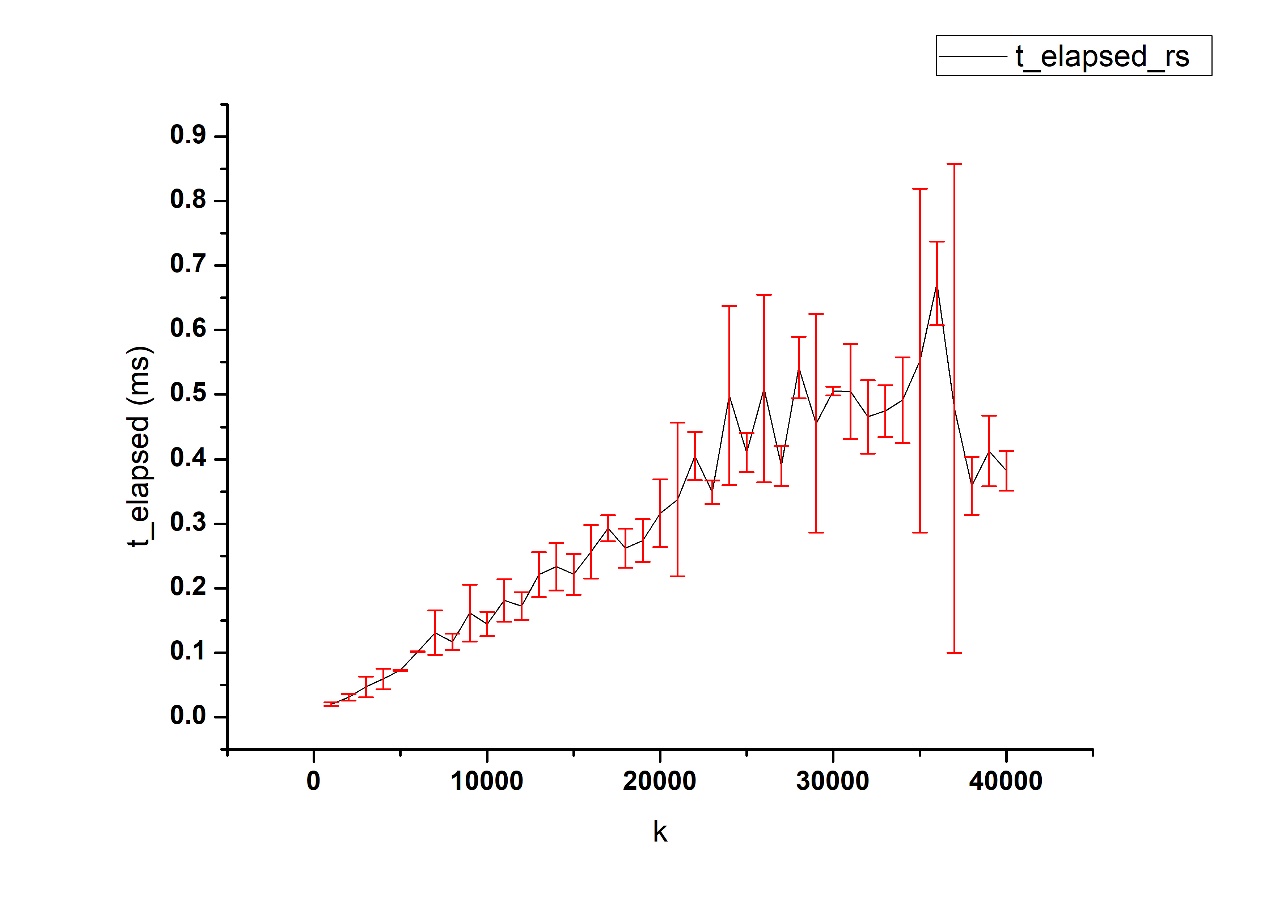
注意，文件中大段注释掉的内容可以用于程序调试，当我们发现两个实例返回的选择结果不一样时，我们可以保存下这个随机数组，下次调试时读取这个数组并且可以单步调试。

第二种模式通过调用createReportFile()函数，在当前目录下生成汇报文档。内容包含两种算法在相同数组输入下，求相同的顺序统计量消耗的时间对比。可以通过修改宏定义来控制循环次数和实验的步长。每种条件下测量5次取平均和极差作为衡量算法耗时和稳定性的依据。

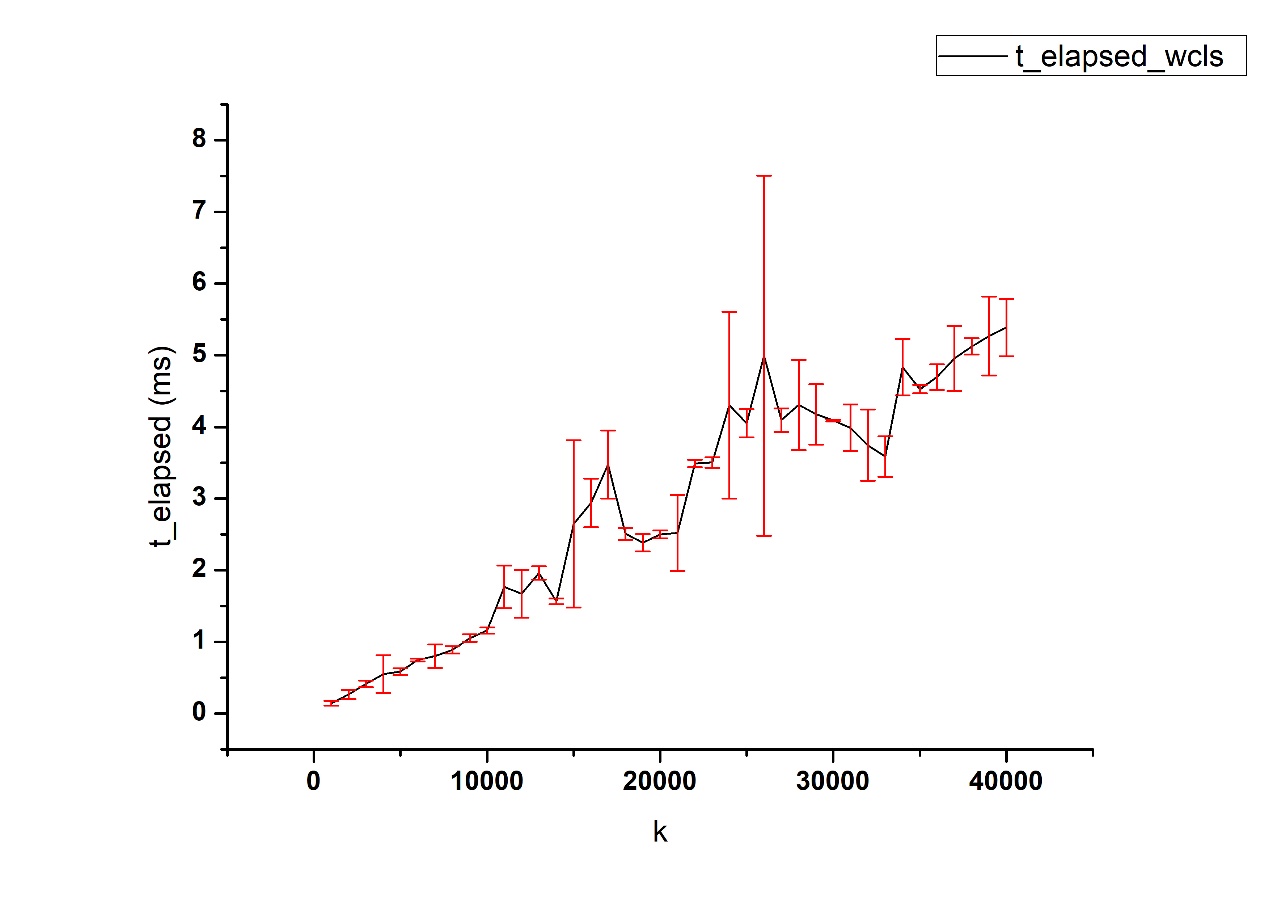
1. **实验结果**

数组规模1000~40000，步长1000，共40组数据，求第6小的元素，结果保存为.csv格式，利用Excel和Origin软件来处理数据，作图。

随机选择：RandomizedSelect (rs)



最坏情况线性选择： WorstCaseLinearSelect (wcls)



1. **结果分析和总结**

可以看出，这两种算法对k近似成线性，但是最坏情况线性算法为了保证最坏情况下的线性，比随机选择算法多了一步寻找中位数的过程，这个代价是线性时间常数C增大，导致这个算法下的时间消耗普遍大于随机选择算法。所以，随机选择算法平均来说要优于最坏情况线性算法，并且从图上看抖动并不大。而最坏情况线性选择算法的理论意义大于实践意义，不推荐作为真正的选择算法使用。