哈尔滨工程大学

实 验 报 告

实 验 名 称：­­

班 级：

学 号：

姓 名：

实 验 时 间：

成 绩：

指 导 教 师：

实验室名称：

**哈尔滨工程大学实验室与资产管理处 制**

|  |
| --- |
| 实验名称：查找最高分和次高分 |
| 一、问题描述  有512人参与玩某游戏，从1~512给每个人分配一个编号，每个人的游戏得分在0~999之间，现要用不同方法查找出游戏参与者的最高分和次高分。  使用三种实现方法：顺序查找法、锦标赛法、堆法。  二、数据结构设计  定义一个长度为512的数组，并随机生成0~999的整数填入每一个数组元素中去  vector<int> array(n);  srand(static\_cast<unsigned int>(time(0)));  // 为数组赋随机整数值  for (int i = 0; i < n; ++i) {  array[i] = rand() % 1000; // 生成0~999之间的随机数  }  三、算法设计    1.随机生成512组数据并输出。  （输出时每打印10个人的分数就换行一次）  void print(vector<int>& array,int n){  int flag = 0;  cout << "随机赋值后的数组：" << endl;  for(int i = 0; i < n; ++i) {  flag++;  if(flag%10==0) {cout <<"第"<<setw(3)<<i+1<<"个人的得分为:"<< setw(3) << array[i]<<endl;}  else cout <<"第"<<setw(3)<<i+1<<"个人的得分为:"<< setw(3) << array[i]<<" ";  if(flag==n) cout <<endl;  }  }  2.顺序查找方法  ①寻找最大值：  记录当前所遇到的最大分数max和产生这个最大分数的人的编号Pmax。在遍历前，将max和Pmax分别初始化成array[0]和0。  从第1个人遍历到第512个人，每遍历到一个人，便将其分数与max作比较，如果大于max，则更新max和Pmax的值。直到遍历到最后，max即为这512个人中的最大值，Pmax即为产生这个最大值的人的编号。  int Pmax = 0;  for (int i = 0; i < n; ++i) {  if (array[i] > array[Pmax]) {  Pmax = i;  }  }  ②寻找次大值：  类似寻找最大值，但要加一个限制条件：i != Pmax来避免冲突  2.锦标赛法  锦标赛法是先比较1和2、3和4、5和6……在这256对比较中会产生256个胜者，然后这256个胜者再进行成对比较产生128个胜者，以此类推。由于512是2的偶数次幂，最终一定会产生一对，并角逐出胜者。这一对人中，分数较大的是最大值，较小的人的分数就是次大值。  构建一棵线段树，从最大的1~512区间开始，分治初始化左右节点。当前初始化的节点所代表的区间如果只包含一个元素，那么这个节点是叶子节点，可以不再进行分治向下递归了。当前节点如果左右子节点均完成初始化后，即可对当前节点的max、m2、mi、mi2信息进行计算。计算时需要用到子节点的信息：  选取的最大值为左右子节点的最大值中的最大值，并取对应的下标。  如果左右子节点中的最大值中的较小值大于左右子节点中的次小值中的较大值，则当前区间的次小值取前者，否则取后者，并取得对应下标。  最后，整个区间的最大值和次大值即为1~512对应的区间的最大值和次大值。  3.建堆与堆调整法  核心代码：void heapify(vector<int>& array,int n,int i) {  int largest = i; // 假设根节点最大  int left = 2 \* i + 1; // 左子节点  int right = 2 \* i + 2; // 右子节点  // 如果左子节点比根节点大，则更新最大值索引  if (left < n && array[left] > array[largest])  largest = left;  // 如果右子节点比当前最大值大，则更新最大值索引  if (right < n && array[right] > array[largest])  largest = right;  // 如果最大值不是根节点，则交换并继续堆化  if (largest != i) {  int temp = array[i];  array[i] = array[largest];  array[largest] = temp;  heapify(array, n, largest);  }  }  void buildHeap(vector<int>& array,int n) {  // 从最后一个非叶子节点开始向上构建堆  for (int i = n / 2 - 1; i >= 0; i--) {  heapify(array, n, i);  }  }  最后，512个数中的最大值和次大值分别为max和maxx。  四、界面设计、运行测试  ①输出随机生成的512组数据。    ②输出三种排序所获得的结果，以及所用到的时间    五、分析  对三种查找方法的时间复杂度进行分析。  ①顺序查找  不难看出，顺序查找方法只执行了两次从1到512的循环，是一个O(n)级别的方法。  ②锦标赛法  锦标赛法使用的是线段树。虽然线段树本身进行修改和查询是一个log级别的复杂度，但是在本实验的查找中，我们只对建出来的线段树的2n-1的节点进行遍历，对每一个点的信息更新时间复杂度是O(1)的，所以整体的时间复杂度是O(n)的。  ③建堆与堆调整  由于实现过程中我们用一维数组来模拟堆，所以我们在建堆（每次插入节点时）的时间复杂度是O(n)的，但是在堆调整时，我们要对每一个点进行调整，从而将原本不符合大根堆性质的地方恢复。最坏情况为，如果我们按照从小到大的方式插入，那么我们每次插入后，都要进行log n次的堆调整。所以整体的时间复杂度为O(nlog n)级别的。  六、实验收获与思考  在本次实验中，通过学习，我学会了使用顺序查找法、锦标赛法、建堆及堆排序法来获得一段序列的最大值。  本次实验的要求只是获取最大值和次大值，所以顺序查找法无论从时间复杂度方面和实现难易度方面，都是比较优秀的。锦标赛法要进行至少n次比较，不过整体也是O(n)的，不过要专门建一棵树来实现锦标赛法，实现起来比较有难度。这两种方式都不受到数据的限制，所以属于时间复杂度相对稳定的方法。  而第三种的建堆与堆调整，时间复杂度不稳定。最好情况是：如果插入堆中的数的顺序是从大到小，那么时间复杂度可以达到O(n)，此时级别都是O(n)，不过就常数而言优于前两种方法；最坏情况是：如果插入堆中的数的顺序是从小到大，意味着每次插入后都要进行堆调整的操作，并且都要将这个元素从堆的叶子节点一端移动到根节点的位置，进行log n级别次数的比较，导致整体时间复杂度退化到了O(nlog n)级别。不过从实现方法来看，堆的方法仅次于顺序查找法。 |
|  |