**重 庆 大 学**

**学 生 实 验 报 告**

**实验课程名称 数据结构与算法**

**开课实验室 D1501**

**学 院 软件学院 年级 2016 专业班 软件2班**

**学 生 姓 名 丁子元 学 号 20161616**

**开 课 时 间 2016 至 2017 学年第 1 学期**

|  |  |
| --- | --- |
| **总 成 绩** |  |
| **教师签名** | **文俊浩** |

**软件学院制**

**《数据结构与算法》实验报告**

**开课实验室：DS1501 2017 年 10 月　10 日**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 学院 | | 软件学院 | 年级、专业、班 | | 2016级2班 | 姓名 | 丁子元 | | 成绩 | |  |
| 课程  名称 | | 数据结构与算法 | | 实验项目  名 称 | 排序算法,检索算法 | | | 指导教师 | | **文俊浩** | |
| 教师评语 | 教师签名：  年 月 日 | | | | | | | | | | |
| **一、实验目的**  1 练习各种排序算法的C++实现并比较算法性能  2 练习各种检索算法的C++实现并比较算法性能  **二、实验原理**  1 各种排序算法的算法流程  2 各种检索算法的算法流程 | | | | | | | | | | | |

|  |
| --- |
| **三、使用仪器、材料**  Vs2017  **四、实验步骤**  排序算法   1. 插入排序 2. 参考教材实现代码 3. 添加计时代码 4. 分别添加最差,平均,最佳的测试用例 5. 选择排序   1>同上   1. 冒泡排序   1>同上   1. 希尔排序   1>同上   1. 归并排序   1>同上   1. 快速排序   1>同上   1. 堆排序   1>同上   1. 基数排序 2. 同上   检索算法   1. 顺序检索   1>同上   1. 二分检索   1>同上 |

|  |
| --- |
| **五、实验过程原始记录(数据、图表、计算等)**  排序算法   1. 插入排序   1> template<typename E>  void insertsort(E A[], int n)  {  for (int i = 1; i < n; i++)  for (int j = i; (comparetimes++) && (j > 0) && (A[j] < A[j - 1]); j--)  {  swap(A[j], A[j - 1]);  swaptimes++;  }  }   1. 选择排序   1> template <typename E>  void selsort(E A[], int n) { // Selection Sort  for (int i = 0; i<n - 1; i++) { // Select i’th record  int lowindex = i; // Remember its index  for (int j = n - 1; j>i; j--) // Find the least value  if ((comparetimes++)&&(A[j]> A[lowindex]))  lowindex = j; // Put it in place  swap(A[i], A[lowindex]);  swaptimes++;  }  }   1. 冒泡排序   1> template <typename E>  void bubsort(E A[], int n) { // Bubble Sort  for (int i = 0; i<n - 1; i++) // Bubble up i’th record  for (int j = n - 1; j>i; j--)  if( (comparetimes++)&&(A[j] > A[j - 1]))  {  swap(A[j], A[j - 1]);  swaptimes++;  }  }   1. 希尔排序   1> template <typename E>  void inssort2(E A[], int n, int incr) {  for (int i = incr; i<n; i += incr)  for (int j = i; (comparetimes++)&&(j >= incr) &&  ((A[j] > A[j - incr])); j -= incr)  {  swap(A[j], A[j - incr]);  swaptimes++;  }  }  template <typename E>  void shellsort(E A[], int n) { // Shellsort  for (int i = n / 2; i>2; i /= 2) // For each increment  for (int j = 0; j<i; j++) // Sort each sublist  inssort2<E>(&A[j], n - j, i);  inssort2<E>(A, n, 1);  }   1. 归并排序   1>  template <typename E>  void mergesort(E A[], E temp[], int left, int right) {  if (left == right) return; // List of one element  int mid = (left + right) / 2;  mergesort<E>(A, temp, left, mid);  mergesort<E>(A, temp, mid + 1, right);  for (int i = left; i <= right; i++) // Copy subarray to temp  temp[i] = A[i];  // Do the merge operation back to A  int i1 = left; int i2 = mid + 1;  for (int curr = left; curr <= right; curr++) {  if (i1 == mid + 1) // Left sublist exhausted  A[curr] = temp[i2++];  else if (i2 > right) // Right sublist exhausted  A[curr] = temp[i1++];  else if ((comparetimes++)&&(temp[i1]>temp[i2]))  A[curr] = temp[i1++];  else A[curr] = temp[i2++];  }  }   1. 快速排序   1> template <typename E>  void qsort(E A[], int i, int j) { // Quicksort  if (j <= i) return; // Don’t sort 0 or 1 element  int pivotindex = findpivot(A, i, j);  swap(A[pivotindex], A[j]); // Put pivot at end  swaptimes++;  // k will be the first position in the right subarray  int k = partition<E>(A, i - 1, j, A[j]);  swap(A[k], A[j]); // Put pivot in place  swaptimes++;  qsort<E>(A, i, k - 1);  qsort<E>(A, k + 1, j);  }  template <typename E>  inline int findpivot(E A[], int i, int j)  {  return (i + j) / 2;  }  template <typename E>  inline int partition(E A[], int l, int r, E& pivot) {  do { // Move the bounds inward until they meet  while ((A[++l]>pivot)&&(comparetimes++)); // Move l right and  while ((comparetimes++)&&(l < r) && (pivot>A[--r])); // r left  {  swap(A[l], A[r]); // Swap out-of-place values  swaptimes++;  }  } while (l < r); // Stop when they cross  return l; // Return first position in right partition  }   1. 堆排序   1>template <typename E>  void heapsort(E A[], int n) { // Heapsort  E maxval;  heap<E, int> H(A, n, n); // Build the heap  for (int i = 0; i<n; i++) // Now sort  maxval = H.removefirst(); // Place maxval at end  }  在maxheap.h中  void siftdown(int pos) {  while (!isLeaf(pos)) { // Stop if pos is a leaf  int j = leftchild(pos); int rc = rightchild(pos);  if ((comparetimes++)&&(rc < n) && (Heap[rc]> Heap[j]))  j = rc; // Set j to greater child’s value  if ((comparetimes++) && Heap[pos]> Heap[j]) return; // Done  swap(Heap[pos], Heap[j]);  swaptimes++;  pos = j; // Move down  }  }   1. 基数排序   1>  int maxbit(int data[], int n)  {  int d = 1;  int p = 10;  for (int i = 0; i < n; ++i)  {  while (data[i] >= p)  {  p \*= 10;  ++d;  }  }  return d;  }  void radixsort(int data[], int n)  {  int d = maxbit(data, n);  int \*tmp = new int[n];  int \*count = new int[10];  int i, j, k;  int radix = 1;  for (i = 1; i <= d; i++)  {  for (j = 0; j < 10; j++)  count[j] = 0;  for (j = 0; j < n; j++)  {  k = (data[j] / radix) % 10;  count[k]++;  }  for (j = 1; j < 10; j++)  count[j] = count[j - 1] + count[j];  for (j = n - 1; j >= 0; j--)  {  k = (data[j] / radix) % 10;  tmp[count[k] - 1] = data[j];  count[k]--;  }  for (j = 0; j < n; j++)  data[j] = tmp[j];  radix = radix \* 10;  }  delete[]tmp;  delete[]count;  }  检索算法   1. 顺序检索   1> int findK(int A[], int n,int K)  {  for (int i = 0; i<n; i++)  if (abs(A[i]-K)<=0.00001)  {  comparetimes = i+1;  return i;  }  return -1;  }   1. 二分检索   1> int binfindK(int A[], int n, int K)  {  int min = 0;  int max = n- 1;  int mid = (max + min) / 2;  while (A[mid] != K)  {  comparetimes++;  if (K>A[mid]) {  min = mid + 1;  }  else {  max = mid - 1;  }  if (min>max)  return -1;  mid = (max + min) / 2;  }  comparetimes++;  return mid;  } |

|  |
| --- |
| **六、实验结果及分析**  排序算法   1. 插入排序   1>   1. 选择排序   1>   1. 冒泡排序   1>   1. 希尔排序   1>   1. 归并排序   1>   1. 快速排序   1>   1. 堆排序   1>   1. 基数排序   检索算法   1. 顺序检索   1>   1. 二分检索   1> |