** **

**课程实验报告**

课程名： 数据结构实验

学 院： 数学与计算机学院 系 计算机科学与技术系

专 业： 计算机科学与技术

班 级： 计算机222班

学 号： 5418122020

姓 名： 马星

任课教师： 任 燕

授课学期： 2022 年~~~~ 2023 年 1 学期

**目 录**

[实验一 堆排序 3](#_Toc31771)

[实验二 归并排序 4](#_Toc32327)

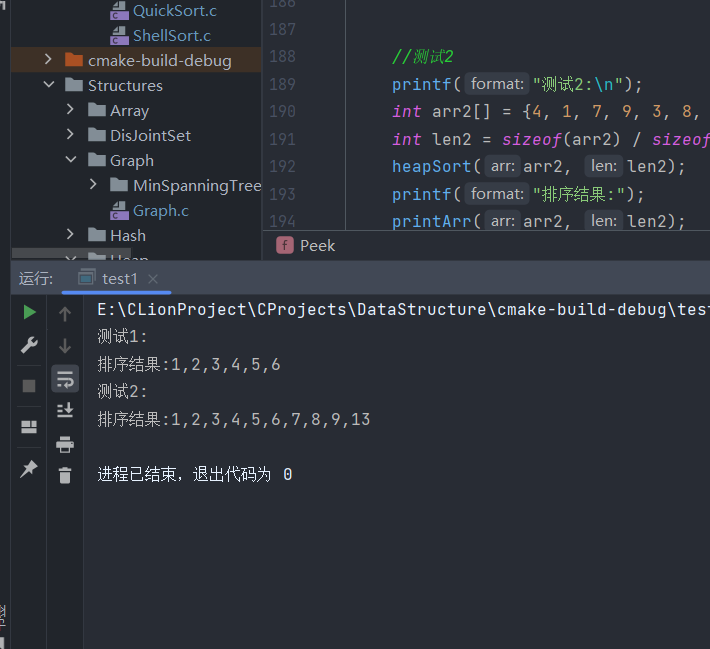
**实验一 堆排序**

学生姓名： 马星 学 号： 5418122020 专业班级： 计算机222班

实验类型：□ 验证 □ 综合 □ 设计 □ 创新 实验日期： 实验成绩：

*#include* <stdio.h>  
*#include* <stdbool.h>  
*#include* <stdlib.h>  
*#include* "../Algorithm/Sort/ArrayUtil.h"  
  
  
*//大顶堆,每个节点都比它的孩子节点大,堆顶始终为最大值  
typedef struct* MinHeap {  
 *int* \*array;  
 *int* size;*//当前元素个数  
 int* capacity;*//最大容量*} MinHeap;  
  
*void* down(MinHeap \*heap, *int* parent);  
  
  
*void* Heapify(MinHeap \*heap);  
  
  
bool IsEmpty(MinHeap heap) {  
 *return* heap.size == 0;  
}  
  
bool IsFull(MinHeap heap) {  
 *return* heap.size == heap.capacity;  
}  
  
  
*void* heapSwap(MinHeap \*heap, *int* i, *int* j) {  
 *int* t = heap->array[i];  
 heap->array[i] = heap->array[j];  
 heap->array[j] = t;  
}  
  
*/\*\*  
 \* 传入一个数组,将其建成大顶堆堆  
 \*/  
void* InitHeap(MinHeap \*heap, *const int* arr[], *int* len, *int* capacity) {  
 heap->size = len;  
 heap->capacity = capacity;  
 heap->array = malloc(*sizeof*(*int*) \* capacity);  
 *for* (*int* i = 0; i < len; i++) {  
 heap->array[i] = arr[i];  
 }  
 *for* (*int* i = len; i < capacity; i++) {  
 heap->array[i] = 0;  
 }  
 Heapify(heap);  
}  
  
*/\*\*  
 <h1>弗洛伊德 建堆算法</h1>  
 1. 找到最后一个非叶子节点<br>  
 2. 从后往前依次找非叶子节点进行下潜<br>  
 <h2>时间复杂度:</h2>  
 总交换次数:最底层交换0次,倒数第二层交换1次,...,第i层交换i-1次<br>  
 [i=1 ~ h] ∑ ( (2^h / 2^i) \* (i - 1) ) = 2^h - h - 1 <br>  
 由于2^h ≈ n, h≈ log\_2\_ n ,所以时间复杂度为O(n)  
 \*/  
void* Heapify(MinHeap \*heap) {  
 *for* (*int* i = heap->size / 2 - 1; i >= 0; i--) {  
 down(heap, i);  
  
 }  
}  
  
*/\*\*  
 \* 大元素下潜  
 \* @param heap Heap  
 \* @param parent 父节点索引  
 \*/  
void* down(MinHeap \*heap, *int* parent) {  
 *int* left = 2 \* parent + 1;  
 *int* right = left + 1;  
 *int* min = parent;*//寻找 父,左,右 三者最小的元素  
 if* (left < heap->size && heap->array[left] < heap->array[min]) {  
 min = left;  
 }  
 *if* (right < heap->size && heap->array[right] < heap->array[min]) {  
 min = right;  
 }  
 *if* (min != parent) {*//如果子节点比父节点优先级大* heapSwap(heap, parent, min);*// 交换* down(heap, min);*//递归,直到max==parent(父节点大于左右子节点)时停止* }  
}  
  
  
*/\*\*  
 \* 添加元素到堆底,然后开始上浮  
 \*/  
void* up(MinHeap \*heap, *int* offered) {  
 *int* child = heap->size;  
 *int* parent = (child - 1) / 2;  
 *while* (child > 0 && offered < heap->array[parent]) {  
 heap->array[child] = heap->array[parent];*//offered比父节点大,将offered上浮* child = parent;  
 parent = (parent - 1) / 2;  
 }  
 heap->array[child] = offered;*//插入*}  
*/\*\*  
 \* 向堆中添加元素  
 \* @param heap  
 \* @param offered  
 \* @return  
 \*/*bool Offer(MinHeap \*heap, *int* offered) {  
 *if* (IsFull(\*heap)) {  
 *return* false;  
 }  
 up(heap, offered);  
 heap->size++;  
 *return* true;  
}  
  
*/\*\*  
 \* 获取堆顶元素(堆内的最大值)  
 \*/  
int* Peek(MinHeap heap) {  
 *if* (IsEmpty(heap)) {  
 exit(-1);  
 }  
 *return* heap.array[0];*//返回堆顶元素*}  
  
*/\*\*  
 \* 抛出堆顶元素(堆内的最小值)  
 \*/  
int* Poll(MinHeap \*heap) {  
 *if* (IsEmpty(\*heap)) {  
 exit(-1);  
 }  
 heapSwap(heap, 0, heap->size - 1);*//交换头元素与尾元素再将尾元素抛出* heap->size--;  
 *int* value = heap->array[heap->size];  
 down(heap, 0);*//下潜  
 return* value;  
}  
  
*/\*\*  
 \* 堆的打印方法  
 \* @param heap  
 \*/  
void* printHeap(MinHeap heap) {  
 MinHeap heapClone;  
 InitHeap(&heapClone, heap.array, heap.size, heap.capacity);  
 *while* (!IsEmpty(heapClone)) {  
 printf("%d,", Poll(&heapClone));  
 }  
 printf("\b\n");  
}  
  
*/\*\*  
 \* 拷贝堆中有序数组到数组中  
 \*/  
void* copyHeapToArray(MinHeap \*heap, *int* \*arr) {  
 *int* k = 0;  
 *while* (!IsEmpty(\*heap)) {  
 arr[k++] = Poll(heap);  
 }  
}  
*/\*\*  
 \* 堆排序  
 \* @param arr 数组  
 \* @param len 数组长度  
 \*/  
void* heapSort(*int* \*arr, *int* len) {  
 MinHeap heap;  
 InitHeap(&heap, arr, len, len);  
 copyHeapToArray(&heap, arr);  
}  
  
*int* main() {  
 *//测试1* printf("测试1:\n");  
 *int* arr1[] = {1, 3, 4, 2, 5, 6};  
 *int* len1 = *sizeof*(arr1) / *sizeof*(*int*);  
 heapSort(arr1, len1);  
 printf("排序结果:");  
 printArr(arr1, len1);  
  
  
 *//测试2* printf("测试2:\n");  
 *int* arr2[] = {4, 1, 7, 9, 3, 8, 13, 2, 5, 6};  
 *int* len2 = *sizeof*(arr2) / *sizeof*(*int*);  
 heapSort(arr2, len2);  
 printf("排序结果:");  
 printArr(arr2, len2);  
 *return* 0;  
}

测试结果:



**实验二 归并排序**

学生姓名： 马星 学 号： 5418122020 专业班级： 计算机222班

实验类型：□ 验证 □ 综合 □ 设计 □ 创新 实验日期： 实验成绩：

*#include* <stdio.h>  
*#include* "../Algorithm/Sort/ArrayUtil.h"  
  
*/\*\*  
 合并两个有序数组  
  
 @param a1 数据源  
 @param i 第一个数组的起始索引  
 @param iEnd 第一个数组的结束索引  
 @param j 第二个数组的起始索引  
 @param jEnd 第二个数组的结束索引  
 @param a2 临时数组  
 \*/  
void* merge(*int* \*a1, *int* i, *int* iEnd, *int* j, *int* jEnd, *int* \*a2) {  
 *int* k = 0;  
 *while* (i <= iEnd && j <= jEnd) {  
 *if* (a1[i] < a1[j]) {  
 a2[k] = a1[i];  
 i++;  
 } *else* {  
 a2[k] = a1[j];  
 j++;  
 }  
 k++;  
 }  
 *//处理剩余没放入a2的部分  
 if* (i > iEnd) {  
 *for* (*int* l = j; l <= jEnd; l++) {  
 a2[k++] = a1[l];  
 }  
  
 }  
 *if* (j > jEnd) {  
 *for* (*int* l = i; l <= iEnd; l++) {  
 a2[k++] = a1[l];  
 }  
 }  
  
}  
  
*/\*\*  
 \* 分治  
 \*/  
void* split(*int* \*a1, *int* left, *int* right, *int* \*a2, *int* len) {  
  
 *// 2.治  
 if* (left == right) {  
 *return*;  
 }  
 *// 1.分  
 int* m = (left + right) >> 1;*//从中间分隔* split(a1, left, m, a2, len);  
 split(a1, m + 1, right, a2, len);  
  
 *// 3.合* merge(a1, left, m, m + 1, right, a2);*//合并两个有序区域到a2  
 for* (*int* i = 0; i < right - left + 1; i++) {  
 a1[i + left] = a2[i];*//将合并后的再存入a1* }  
 printArr(a1, len);  
}  
  
*void* mergeSort(*int* \*a1, *int* len) {  
 *int* \*a2 = malloc(*sizeof*(*int*) \* len);*//临时数组,在合并时将数据放入* split(a1, 0, len - 1, a2, len);  
 free(a2);  
}  
  
  
*int* main() {  
 printf("测试1:\n");  
 *int* arr1[] = {3, 5, 4, 1, 6, 8, 2};  
 *int* len1 = *sizeof*(arr1) / *sizeof*(*int*);  
 printf("开始排序:\n");  
 mergeSort(arr1, len1);  
  
 printf("测试2:\n");  
 *int* arr2[] = {51, 3, 5, 10, 16, 20, 4, 1, 72, 6, 8, 2};  
 *int* len2 = *sizeof*(arr2) / *sizeof*(*int*);  
 printf("开始排序:\n");  
 mergeSort(arr2, len2);  
}

测试结果:

