# 堆排序算法

## 算法

二叉堆的定义：二叉堆是完全二叉树或近似完全二叉树。

二叉堆满足两个特性:

（1）父结点的键值总是大于等于（小于等于）任何一个子节点的键值；

（2）每个结点的左子树和右子树都是一个二叉堆；

当父结点的键值总是大于或者等于任何一个子节点的键值时为大根堆；当父结点的键值总是小于或等于任何一个子节点的键值时为小根堆。

堆的存储：一般都用数组来表示堆，i结点的父结点下标就为（i-1）/2.它的左右子节点的下标分别为2\*i+1和2\*i+2。

堆的插入：每次插入都是将新数据放在数组最后。可以发现从这个新数据的父结点到根结点必然为一个有序的数列，然后将这个新数据插入到这个有序数据中。

初识堆排序：堆排序是指利用堆这种数据结构所设计的一种排序算法。该排序方式主要利用了堆的一些特点：

（1）结构近似于完全二叉树，这样就可以在每层的节点中从左到右遍历。

（2）子结点的值总是小于（或者大于）它的父节点，即是一个大顶堆或小顶堆。

用大根堆排序的基本思想：

先将初始数组建成一个大根堆，此对为初始的无序区；

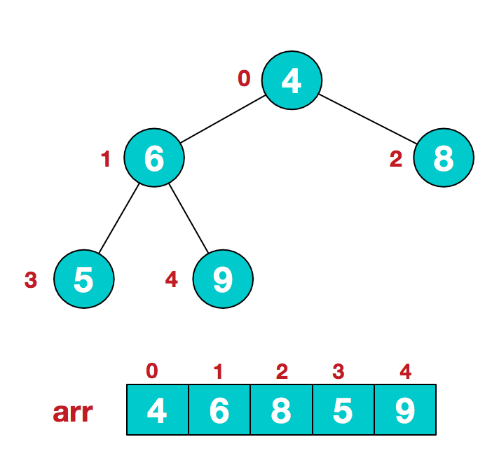
再将最大的元素和无序区的最后一个记录交换，由此得到新的无序区和有序区，且满足<=的值；

由于交换后新的根可能违反堆性质，故将当前无序区调整为堆。然后再次将其中最大的元素和该区间的最后一个记录交换，由此得到新的无序区和有序区，且仍满足关系的值<=的值，同样要将其调整为大根堆..........

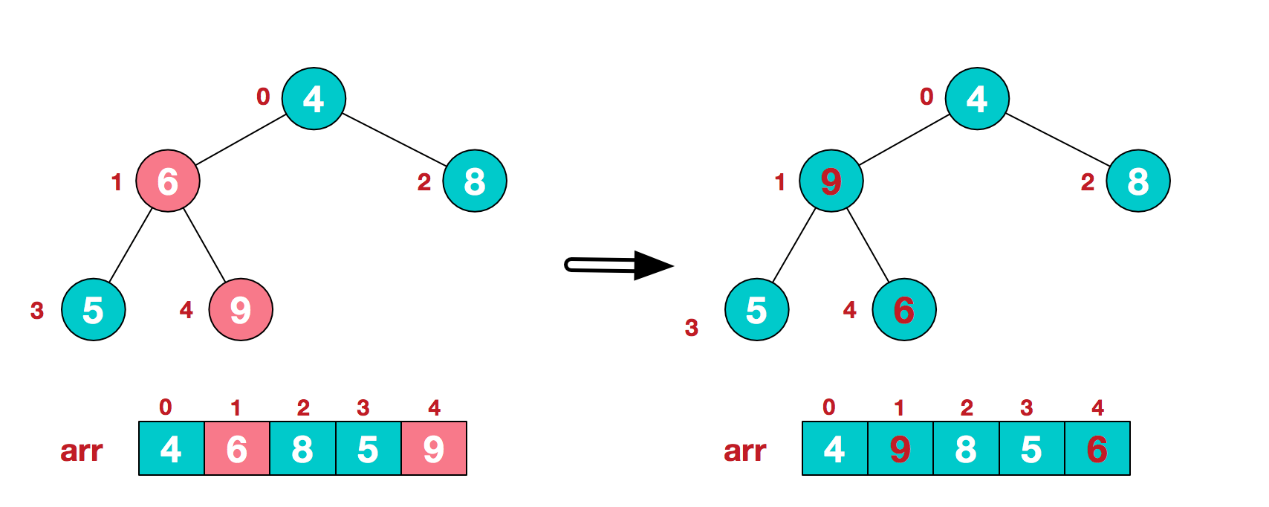
直到无序区只有一个元素为止。

以下通过一个小例子来演示大根堆排序过程：

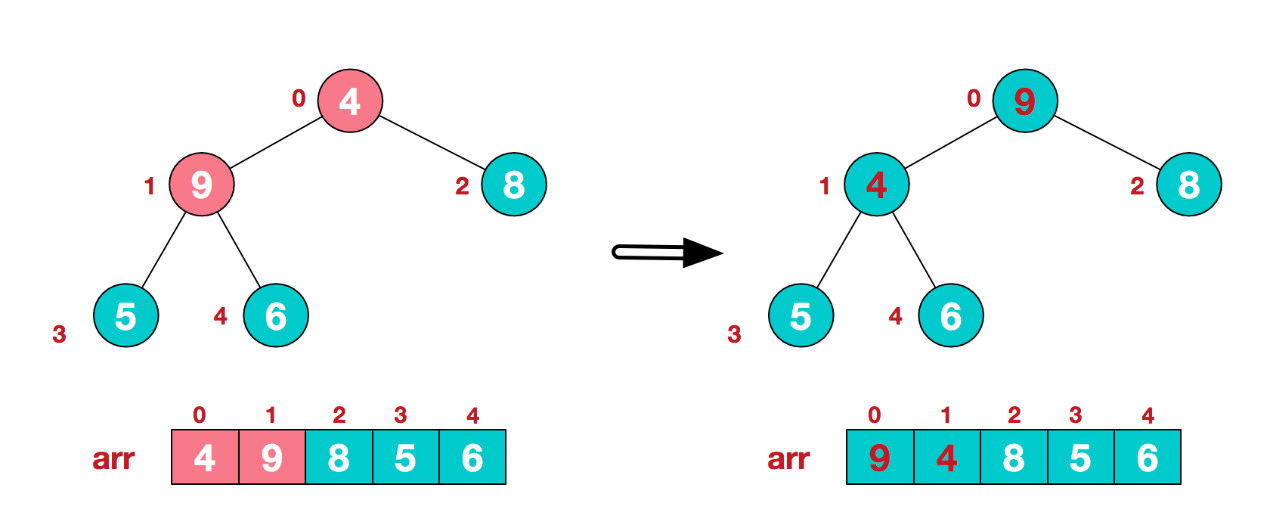
步骤一：构造初始堆。将给定无序序列构造成一个大根堆（一般升序采用大根堆，降序采用小根堆)。

a.假设给定无序序列结构如下

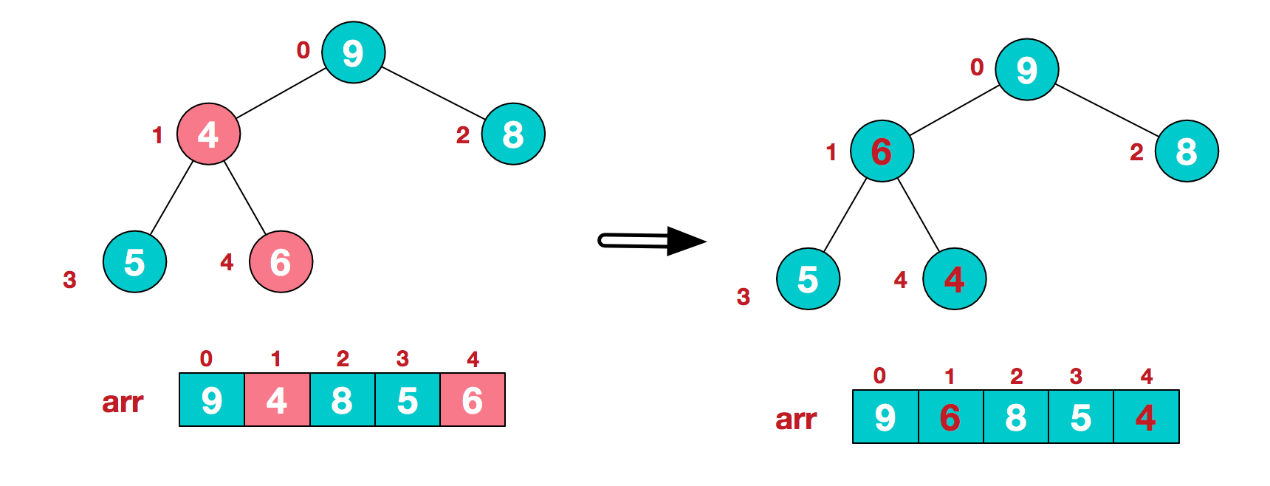
2.此时我们从最后一个非叶子结点开始（叶结点自然不用调整，第一个非叶子结点坐标为len/2-1=5/2-1=1，也就是下面的6结点），从左至右，从下至上进行调整。

此处必须注意，我们把6和9比较交换之后，必须考量9这个节点对于其子节点会不会产生任何影响？因为其是叶子节点，所以不加考虑；但是，一定要熟练这种思维，写代码的时候就比较容易理解为什么会出现一次非常重要的交换了。

4.找到第二个非叶节点4，由于[4,9,8]中9元素最大，4和9交换。

在代码的实现中，这时候4和9交换过后，必须考虑原先9所在的这个节点位置，因为其上的值变为了4，得重新判断以4作为根节点的那棵子树是否还满足大根堆的原则，每一次交换，都必须要循环把子树部分判别清楚。

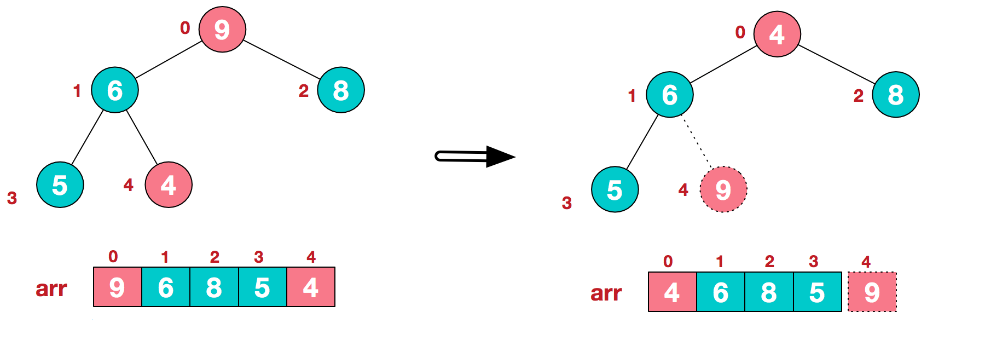
这时，交换导致了子根[4,5,6]结构混乱，继续调整，[4,5,6]中6最大，交换4和6。

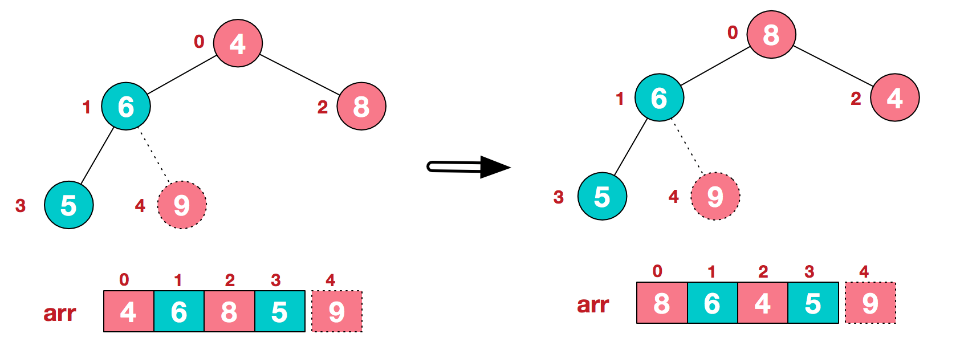
牢记上面说的规则，每次交换都要把改变了的那个节点所在的树重新判定一下，这里就用上了，4和9交换了，变动了的那棵子树就必须重新调整，一直调整到符合大根堆的规则为截。

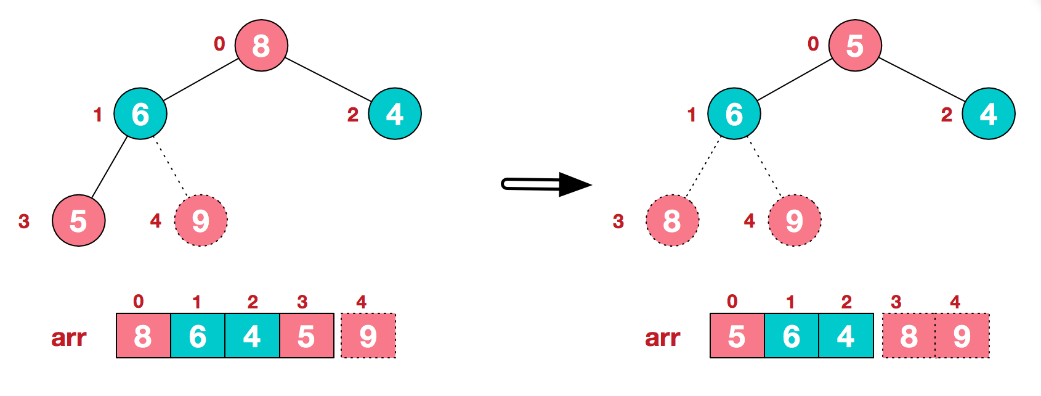
此时，我们就将一个无序序列构造成了一个大顶堆。

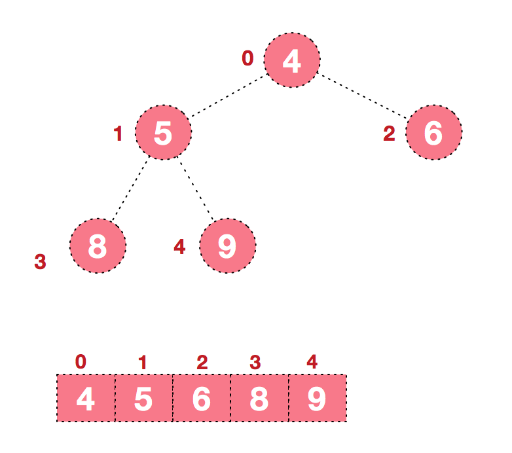
步骤二：将堆顶元素与末尾元素进行交换，使末尾元素最大。然后继续调整堆，再将堆顶元素与末尾元素交换，得到第二大元素。如此反复进行交换、重建、交换。

a.将堆顶元素9和末尾元素4进行交换

这里，必须说明一下，所谓的交换，实际上就是把最大值从树里面拿掉了，剩下参与到排序的树，其实只有总结点的个数减去拿掉的节点个数了。所以图中用的是虚线。

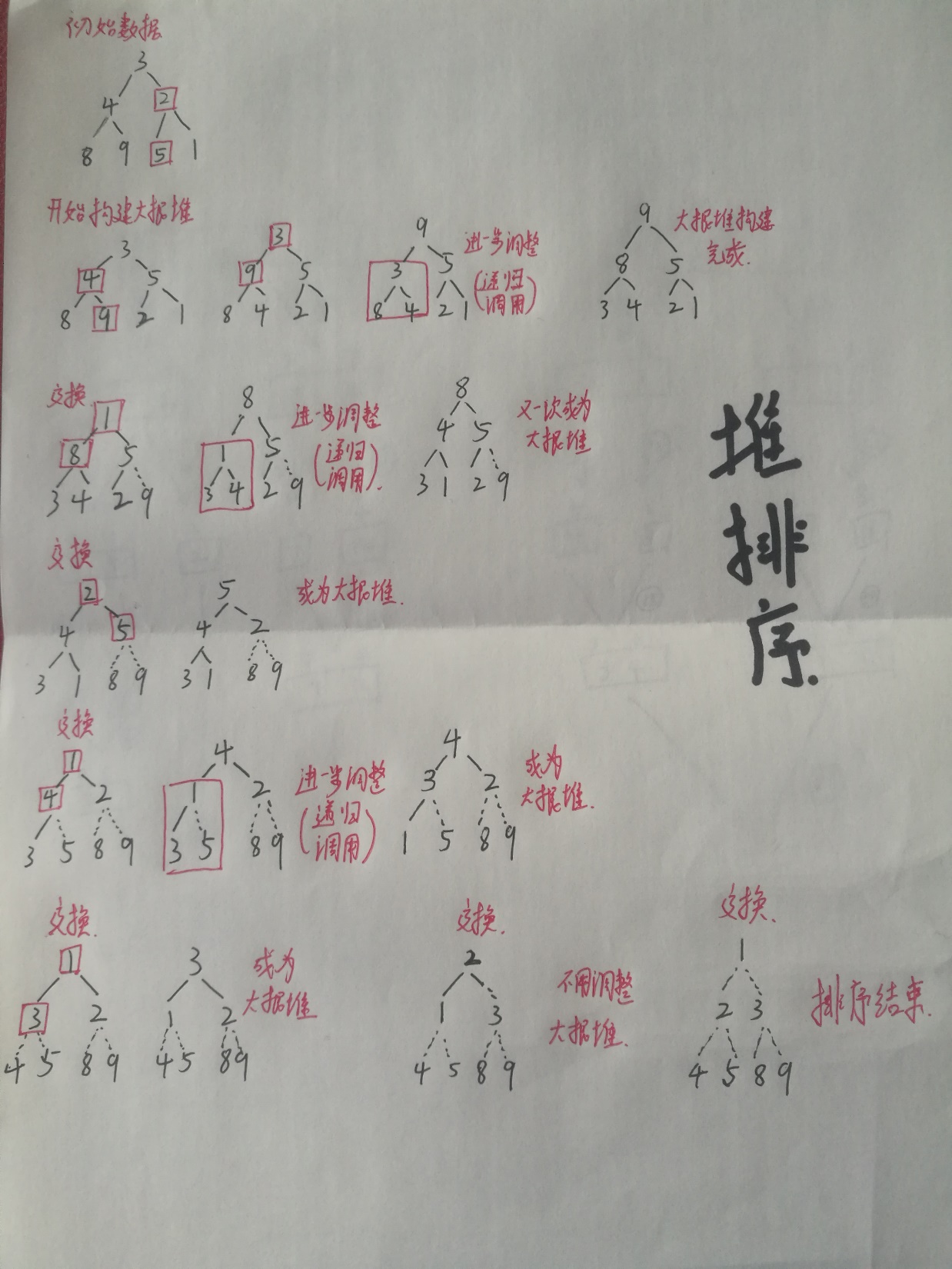
b.重新调整结构，使其继续满足堆定义

c.再将堆顶元素8与末尾元素5进行交换，得到第二大元素8.

后续过程，继续进行调整，交换，如此反复进行，最终使得整个序列有序

## C语言设计

在上一节中，我们了解了堆排序的基本思想，以及用简单的示例演示归并排序的过程。在本节中，我们要用C语言实现归并排序。

在下列代码中，我们主要探讨堆排序的两个过程，即建堆（调整）与排序（交换与调整）。以3、4、2、8、9、5、1为例，下图展现了详细的堆排序过程：

在建堆过程中：我们首先要对初始的无序数组不断调整，将其调整为一个大根堆，调整时无序数组长度始终为len，根节点的遍历选取从右至左，从下至上的方式进行调整。

在排序过程的第一阶段：我们先将之前构建的大根堆中首个元素arr[0]（也就是整个堆的最大值）和堆尾元素交换，此时最大值就放在了数组最后的位置，无序数组长度为len-1。

在排序过程的第二阶段：因为上一阶段对堆顶元素进行交换，破坏了大根堆，所以需要从对顶开始向下进行调整，此时需要调整的无需元素为len-1。调整为新的大根堆之后，继续进行排序过程的第一阶段，如此反复直至堆中无序元素只剩一个结束。

|  |
| --- |
| #include"stdio.h"  #define len 7  /\* 调整堆 \*/  // size是去掉已经排序好的节点剩下的节点数  // root是根节点坐标  void adjust\_heap(int \*arr,int root,int size)  {  int l\_leaf = 2 \* root + 1; // 左叶子节点坐标  int r\_leaf = 2 \* root + 2; // 右叶子节点坐标  int max = root; // 最大值坐标 ，初始化为根节点  int change; // 交换辅助变量  if(l\_leaf < size && arr[l\_leaf] > arr[max]) // 如果存在左叶子节点，并且左叶子节点比之前的最大值（根节点）大  max = l\_leaf; // 最大值坐标指向左叶子节点  if(r\_leaf < size && arr[r\_leaf] > arr[max]) // 如果存在右叶子节点，并且右叶子节点比之前的最大值（根节点或左节点）大  max = r\_leaf; // 最大值坐标指向右叶子节点  if(max!=root) // 如果最大值不是根节点  {  change=arr[max]; // 将根节点与最大值节点交换  arr[max]=arr[root];  arr[root]=change;  root = max; // 以之前最大值坐标指向的节点为根节点进行调整  adjust\_heap(arr,root,size);  }  }  void show(int \*arr,int \_long)  {  int i;  for(i=0;i<\_long;i++)  printf("%d ",arr[i]);  printf("\n");  }  int main()  {  // int arr[8] = {3,8,1,5,2,4,6,7};  int arr[len] = {3,4,2,8,9,5,1};  // 显示排序前的序列  show(arr,len);  // 开始构建大根堆  int size,root\_1,change;  size = len;  for(root\_1=len/2-1;root\_1>=0;root\_1--)// 构建大根堆  {  adjust\_heap(arr,root\_1,size); // 构建过程中无排序完成的节点，所以剩余长度永远为len  }    // 显示大根堆构建结果  show(arr,len);    // 开始调整排序  root\_1 = 0;  for(size=len-1;size>=0;size--)  {  change=arr[0]; // 将堆首和堆尾对调，未排序程度i减一  arr[0]=arr[size];  arr[size]=change;  adjust\_heap(arr,root\_1,size); // 开始交换调整堆，构建完毕之后都是从0节点开始  }  // 显示排序后的序列  show(arr,len);  } |

## 汇编语言设计

用LoongIDE新建一个工程。

移除core文件夹下的bsp\_start.c源文件。

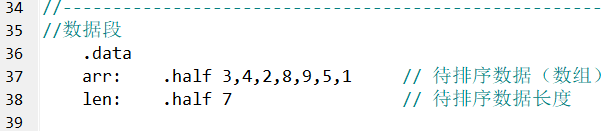
移除main.c源文件。

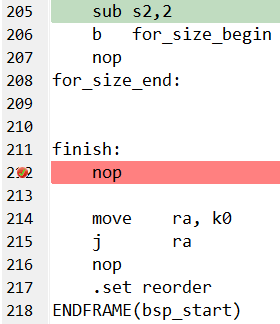
添加bsp\_start.S源文件。

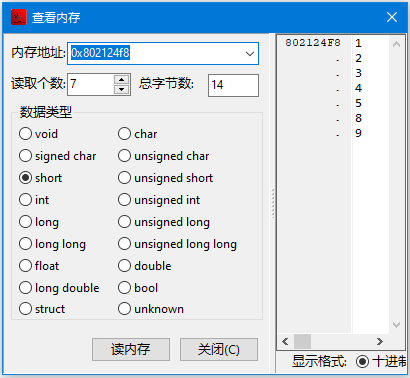
在bsp\_start.S下添加如下代码：

|  |
| --- |
| /\*  \* bsp\_start.S  \*  \* created: 2022/2/1  \* author: Li TianLing  \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*  \* v0 存储器寻址辅助寄存器  \* v1 跳转辅助寄存器  \* s0 待排序数据arr基地址 （常量）  \* s1 参数len （常量）  \* s2 参数size  \* s3 参数root  \* s4 参数l\_leaf  \* s5 参数r\_leaf  \* s6 参数max  \* s7 参数root\_1 函数调用时root辅助参数  \* t1 arr[len]  \* t2 arr[size]  \* t3 arr[root]  \* t4 arr[l\_leaf]  \* t5 arr[r\_leaf]  \* t6 arr[max]  \*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include "regdef.h"  #include "cpu.h"  #include "asm.h"  //-----------------------------------------------------------------------------  //数据段  .data  arr: .half 3,4,2,8,9,5,1 // 待排序数据（数组）  len: .half 7 // 待排序数据长度  //-----------------------------------------------------------------------------  //代码段  .text  FRAME(bsp\_start,sp,0,ra)  .set noreorder  move k0, ra /\* 返回地址 \*/  la s0,arr // 将arr基地址加载到s0（常量）  la v0,len // 将长度len加载到s1（常量）  lh s1,(v0)  sll s1,1    move s2,s1 // size = len  div s1,4 // root\_1 = (len/4)\*2-2  mflo s7  mul s7,2  sub s7,2  for\_root\_begin: // 构建大根堆  bltz s7,for\_root\_end// root\_1<0结束此循环  nop    Adjust\_heap\_1\_begin:  move s3,s7 // root=root\_1  Digui\_1\_begin:  mul s4,s3,2 // l\_leaf=root\*2+2  add s4,2  add s5,s4,2 // r\_leaf=l\_leaf+2  move s6,s3 // max=root    add v0,s0,s3 // 将arr[root]加载到t3  lh t3,(v0)  add v0,s0,s4 // 将arr[l\_leaf]加载到t4  lh t4,(v0)  add v0,s0,s5 // 将arr[r\_leaf]加载到t5  lh t5,(v0)  add v0,s0,s6 // 将arr[max]加载到t6  lh t6,(v0)  if\_l\_leaf\_1\_begin:  sub v1,s4,s2 // l\_leaf>=size不执行if  bgez v1,if\_l\_leaf\_1\_end  nop  sub v1,t4,t6  blez v1,if\_l\_leaf\_1\_end // arr[l\_leaf]<=arr[max]不执行if  nop    move s6,s4 // max=l\_leaf  if\_l\_leaf\_1\_end:  if\_r\_leaf\_1\_begin:  sub v1,s5,s2 // r\_leaf>=size不执行if  bgez v1,if\_r\_leaf\_1\_end  nop  add v0,s0,s6 // max可能改变，重新将arr[max]加载到t6  lh t6,(v0)  sub v1,t5,t6  blez v1,if\_r\_leaf\_1\_end // arr[r\_leaf]<=arr[max]不执行if  nop  move s6,s5 // max=r\_leaf  if\_r\_leaf\_1\_end:  if\_not\_equal\_1\_begin:  beq s3,s6,if\_not\_equal\_1\_end // max==root不执行此if  nop  add v0,s0,s6 // max改变，重新将arr[max]加载到t6  lh t6,(v0)  add v0,s0,s3 // 将t6保存到arr[root]  sh t6,(v0)  add v0,s0,s6 // 将t3保存到arr[max]  sh t3,(v0)    move s3,s6 // root = max  b Digui\_1\_begin  nop  if\_not\_equal\_1\_end:  Adjust\_heap\_1\_end:    sub s7,2 // root\_1-2  b for\_root\_begin  nop  for\_root\_end:  nop  nop  la s7,0 // root\_1 = 0  la v0,len // 重新加载长度len加载到s1（第53行len被改）  lh s1,(v0)  sll s1,1  sub s2,s1,2 // size=len-2  for\_size\_begin:  bltz s2,for\_size\_end // size<0结束  nop    add v0,s0,zero // 将arr[0]加载到t1  lh t1,(v0)  add v0,s0,s2 // 将arr[size]加载到t2  lh t2,(v0)  add v0,s0,zero // 将t2保存到arr[0]  sh t2,(v0)  add v0,s0,s2 // 将t1保存到arr[size]  sh t1,(v0)    Adjust\_heap\_2\_begin:  move s3,s7 // root=root\_1  Digui\_2\_begin:  mul s4,s3,2 // l\_leaf=root\*2+2  add s4,2  add s5,s4,2 // r\_leaf=l\_leaf+2  move s6,s3 // max=root  add v0,s0,s3 // 将arr[root]加载到t3  lh t3,(v0)  add v0,s0,s4 // 将arr[l\_leaf]加载到t4  lh t4,(v0)  add v0,s0,s5 // 将arr[r\_leaf]加载到t5  lh t5,(v0)  add v0,s0,s6 // 将arr[max]加载到t6  lh t6,(v0)  if\_l\_leaf\_2\_begin:  sub v1,s4,s2 // l\_leaf>=size不执行if  bgez v1,if\_l\_leaf\_2\_end  nop  sub v1,t4,t6  blez v1,if\_l\_leaf\_2\_end // arr[l\_leaf]<=arr[max]不执行if  nop  move s6,s4 // max=l\_leaf  if\_l\_leaf\_2\_end:  if\_r\_leaf\_2\_begin:  sub v1,s5,s2 // r\_leaf>=size不执行if  bgez v1,if\_r\_leaf\_2\_end  nop  add v0,s0,s6 // max可能改变，重新将arr[max]加载到t6  lh t6,(v0)  sub v1,t5,t6  blez v1,if\_r\_leaf\_2\_end // arr[r\_leaf]<=arr[max]不执行if  nop  move s6,s5 // max=r\_leaf  if\_r\_leaf\_2\_end:  if\_not\_equal\_2\_begin:  beq s3,s6,if\_not\_equal\_2\_end // max==root不执行此if  nop  add v0,s0,s6 // max改变，重新将arr[max]加载到t6  lh t6,(v0)  add v0,s0,s3 // 将t6保存到arr[root]  sh t6,(v0)  add v0,s0,s6 // 将t3保存到arr[max]  sh t3,(v0)  move s3,s6 // root = max  b Digui\_2\_begin  nop  if\_not\_equal\_2\_end:  Adjust\_heap\_2\_end:  sub s2,2 // size-2  b for\_size\_begin  nop  for\_size\_end:  finish:  nop  move ra, k0  j ra  nop  .set reorder  ENDFRAME(bsp\_start) |

## 调试运行

初始数据如下所示：

如下图所示，在finish后打上断点：

联合调试后，双击s0寄存器，查看对应存储器数据：

如上图所示，排序成功。