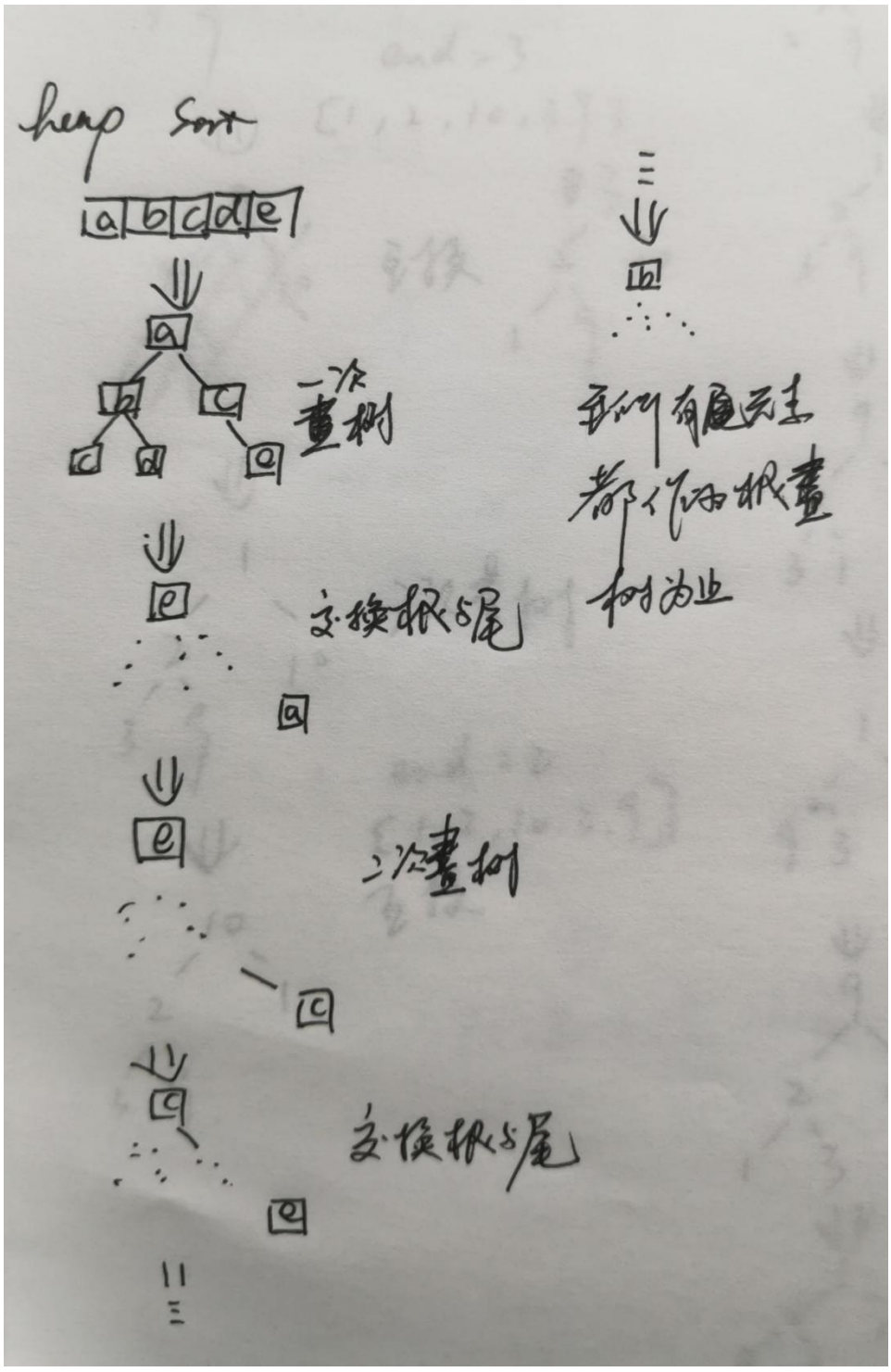
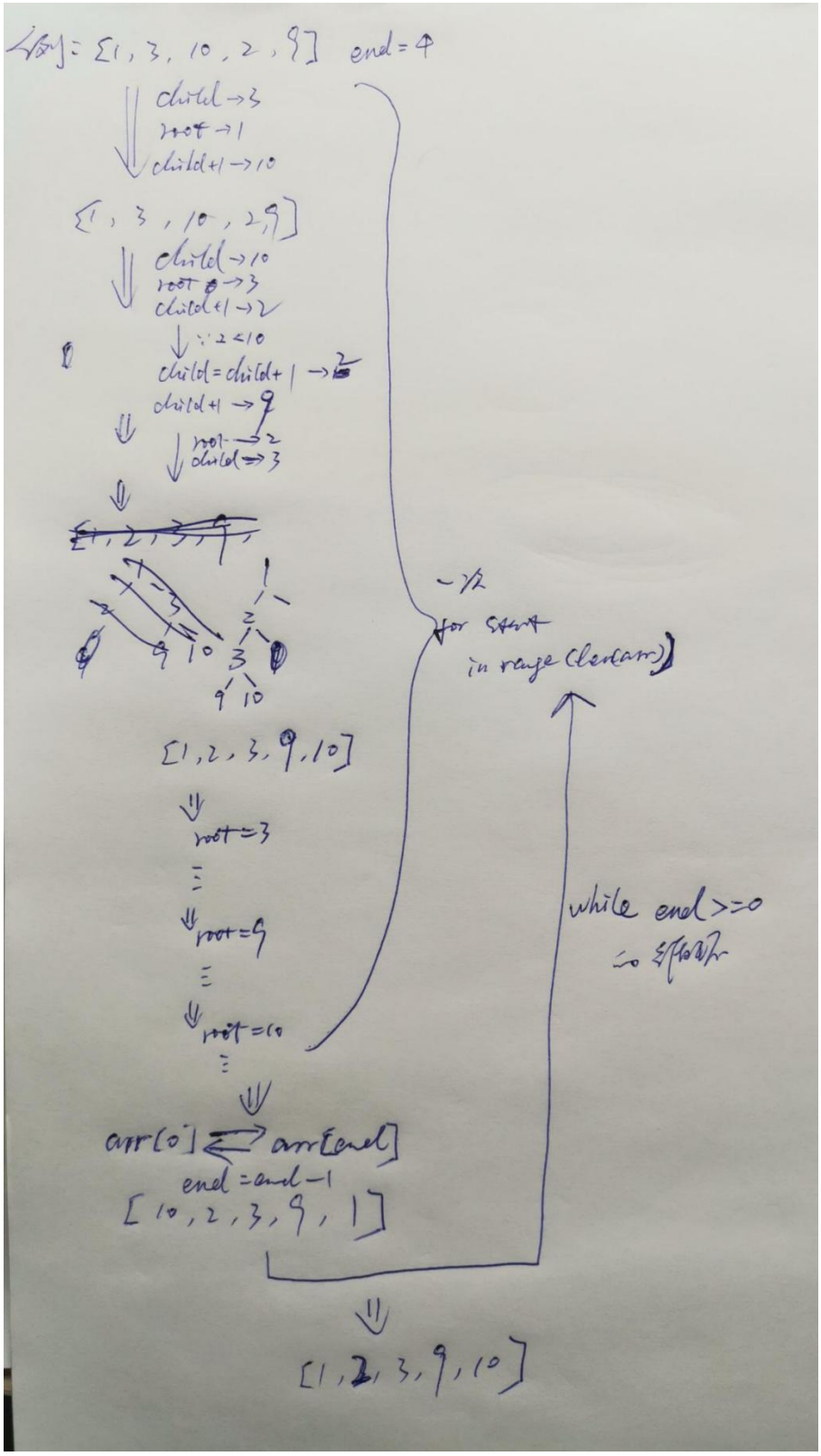
Heap sort

*註：引用資料採用藍色，我的說明採用紅色

1、流程圖





2、學習歷程

我參考了 https://baike.baidu.com/item/堆排序/2840151?fr=aladdin%EF%BC%89 其中關於堆的操作的說明，如下：

在堆的數據結構中，堆中的最大值總是位於根節點（在優先佇列中使用堆的話堆中的最小值位於根節點）。堆中定義以下幾種操作：

- 最大堆調整（Max Heapify）：將堆的末端子節點作調整，使得子節點永遠小於父節點
- 創建最大堆（Build Max Heap）：將堆中的所有數據重新排序
- 堆排序（HeapSort）：移除位在第一個數據的根節點，並做最大堆調整的遞歸運算

與以下兩段代碼：

```
#include <iostream>
#include <algorithm>
using namespace std;

void max_heapify(int arr[], int start, int end)
{
  //建立父節點指標和子節點指標
  int dad = start;
  int son = dad * 2 + 1;
  while (son <= end)  //若子節點指標在範圍內才做比較
  {
    if (son + 1 <= end && arr[son] < arr[son + 1]) //先比較兩個子節點大小，選擇最大的
      son++;
    if (arr[dad] > arr[son]) //如果父節點大於子節點代表調整完畢，直接跳出函數
      return;
    else  //否則交換父子內容再繼續子節點和孫節點比較
    {
      swap(arr[dad], arr[son]);
      dad = son;
      son = dad * 2 + 1;
    }
  }
}

void heap_sort(int arr[], int len)
{
  //初始化，i 從最後一個父節點開始調整
  for (int i = len / 2 - 1; i >= 0; i--)
    max_heapify(arr, i, len - 1);
  //先將第一個元素和已經排好的元素前一位做交換，再從新調整(剛調整的元素之前的元素)，直到排序完畢
  for (int i = len - 1; i > 0; i--)
  {
    swap(arr[0], arr[i]);
    max_heapify(arr, 0, i - 1);
  }
}

void main()
{
```

```
    int arr[] = { 3, 5, 3, 0, 8, 6, 1, 5, 8, 6, 2, 4, 9, 4, 7, 0, 1, 8, 9, 7, 3, 1, 2, 5, 9, 7, 4, 0, 2, 6 };
    int len = (int) sizeof(arr) / sizeof(*arr);
    heap_sort(arr, len);
    for (int i = 0; i < len; i++)
        cout << arr[i] << ' ';
    cout << endl;
    system("pause");
}
```

```
def big_endian(arr,start,end):
    root=start
    child=root*2+1 #左孩子 *這行的原因看不懂，所以改為了 child=root+1，邏輯是一樣的
    while child<=end:
    #孩子比最後一個節點還大，也就意味著最後一個葉子節點了，就得跳出去一次迴圈，已經調整完畢
        if child+1<=end and arr[child]<arr[child+1]:
        #為了始終讓其跟子元素的較大值比較，如果右邊大就左換右，左邊大的話就默認
            child+=1
        if arr[root]<arr[child]:
        #父節點小於子節點直接交換位置，同時座標也得換，這樣下次迴圈可以準確判斷：是否為最底層，
        #是不是調整完畢
            arr[root],arr[child]=arr[child],arr[root]
            root=child
            child=root*2+1
        else:
        break

def heap_sort(arr): #無序區大根堆排序
    first=len(arr)//2 - 1
    for start in range(first,-1,-1):*range（）中的第三個數字為步數，可以省略，下面也一樣
    #從下到上，從左到右對每個節點進行調整，迴圈得到非葉子節點
        big_endian(arr,start,len(arr)-1) #去調整所有的節點
    for end in range(len(arr)-1,0,-1):
        arr[0],arr[end]=arr[end],arr[0] #頂部尾部互換位置
        big_endian(arr,0,end-1) #重新調整子節點的順序，從頂開始調整*堅持了重新調整的目標，做了些許調整
    return arr

def main():   *以下似與 heap sort 無關，故省略
    l=[3,1,4,9,6,7,5,8,2,10]
    print(heap_sort(l))

if __name__=="__main__":
     main()
```

此種排序的目標是使父節點大於或小於所有子節點，所以首先做了如下嘗試：

```
In [5]: def heapsort(arr):
            root=0
            child=root+1
            end=len(arr)-1
            while child<=end:
            #僅在子節點的位置在arr範圍之內時進行
                if child+1<=end and arr[child+1]<arr[child]:
                #若子節點的下一位小於子節點則互換；使root始終與較小值比較
                    child+=1
                if arr[root]>arr[child]:
                #若root大於子節點，則互換位置與坐標，以保證root始終為最小值
                    arr[root],arr[child]=arr[child],arr[root]
                    root=child
                    child=root+1
                else:break
            return arr
```

出現如下結果：

```
In [6]: arr1=[7,5,6,2,0,10,22]
        heapsort(arr1)

Out[6]: [5, 2, 6, 0, 7, 10, 22]
```

可見，一次這樣的循環，是將第一個數值放到該放的位置，所以需要每一個數值都運行一次，如下：

```
In [1]: def good(arr,start,end):
            root=start
            child=start+1
            while child<=end:
            #僅在子節點的位置在arr範圍之內時進行
                if child+1<=end and arr[child+1]<arr[child]:
                #若子節點的下一位小於子節點則互換；使root始終與較小值比較
                    child+=1
                if arr[root]>arr[child]:
                #若root大於子節點，則互換位置與坐標，以保證root始終為最小值
                    arr[root],arr[child]=arr[child],arr[root]
                    root=child
                    child=root+1
                else:break


        def heapsort(arr):
            for start in range(len(arr)):
                good(arr,start,len(arr)-1)#開始一輪排序
            return arr
```

出現如下結果：

```
In [2]: arr1=[7,5,6,2,0,10,22]
        heapsort(arr1)

Out[2]: [5, 0, 2, 6, 7, 10, 22]
```

可見一旦數值被放到前面，root 改變，前面的位置將不會變動
因此，嘗試讓其從頭再來一遍

```
In [7]: def good(arr,start,end):
            root=start
            child=start+1
            while child<=end:
            #僅在子節點的位置在arr範圍之內時進行
                if child+1<=end and arr[child+1]<arr[child]:
                #若子節點的下一位小於子節點則互換；使root始終與較小值比較
                    child+=1
                if arr[root]>arr[child]:
                #若root大於子節點，則互換位置與坐標，以保證root始終為最小值
                    arr[root],arr[child]=arr[child],arr[root]
                    root=child
                    child=root+1
                else:break


        def heapsort(arr):
            for start in range(len(arr)):
                for start in range(len(arr)):
                    good(arr,start,len(arr)-1)#開始一輪排序
            return arr
```

結果如下

```
In [8]: arr1=[7,5,6,2,0,10,22]
        heapsort(arr1)

Out[8]: [0, 2, 5, 6, 7, 10, 22]
```

但如此的次数为 n^2，並不是 logn

因此思考，是否能不再管後面的數值（已經排序好的緣故）的情況下，進行再次調整：

```
def good(arr,start,end):
    root=start
    child=start+1
    while child<=end:
    #僅在子節點的位置在arr範圍之內時進行
        if child+1<=end and arr[child+1]<arr[child]:
        #若子節點的下一位小於子節點則互換；使root始終與較小值比較
            child+=1
        if arr[root]>arr[child]:
        #若root大於子節點，則互換位置與坐標，以保證root始終為最小值
            arr[root],arr[child]=arr[child],arr[root]
            root=child
            child=root+1
        else:break


def heapsort(arr):
    end=len(arr)-1
    while end>=0:#運行直到所有節點都被抽出
        for start in range(len(arr)):
            good(arr,start,len(arr)-1)#開始一輪排序
        arr[0],arr[end]=arr[end],arr[0]#將末尾值與根互換，使最末值成為新根
        end-=1
    return arr
```

此時，則不斷的將重排範圍縮短，同時保留了前面需重新調整的數值位子，此時的運行 good 的次數依然為 n^2

所以，再次做如下調整：

In [27]:
```
def good(arr,start,end):
    root=start
    child=start+1
    while child<=end:
    #僅在子節點的位置在arr範圍之內時進行
        if child+1<=end and arr[child+1]<arr[child]:
        #若子節點的下一位小於子節點則互換；使root始終與較小值比較
            child+=1
        if arr[root]>arr[child]:
        #若root大於子節點，則互換位置與坐標，以保證root始終為最小值
            arr[root],arr[child]=arr[child],arr[root]
            root=child
            child=root+1
        else:break


def heapsort(arr):
    for start in range(len(arr)-1,0):
        good(arr,start,len(arr)-1)#開始一輪排序
    for end in range(len(arr)-1,0):
        arr[0],arr[end]=a[end],arr[0]
        good(arr,0,end)
    return arr
```

出現如下錯誤：

以下為檢驗1:

In [28]:
```
arr1=[7,5,6,2,0,10,22]
heapsort(arr1)
```

Out[28]: [7, 5, 6, 2, 0, 10, 22]

如此看來似乎還是要在 for end 下再加入 for start，運行 good 的次數會更多
所以暫時仍採用，“文字說明”中的程式。

3、文字說明

In [8]:
```
def good(arr,start,end):
    root=start
    child=start+1
    while child<=end:
    #僅在子節點的位置在arr範圍之內時進行
        if child+1<=end and arr[child+1]<arr[child]:
        #若子節點的下一位小於子節點則互換；使root始終與較小值比較
            child+=1
        if arr[root]>arr[child]:
        #若root大於子節點，則互換位置與坐標，以保證root始終為最小值
            arr[root],arr[child]=arr[child],arr[root]
            root=child
            child=root+1
        else:break


def heapsort(arr):
    end=len(arr)-1
    while end>=0:#運行直到所有節點都被抽出
        for start in range(len(arr)):
            good(arr,start,len(arr)-1)#開始一輪排序
        arr[0],arr[end]=arr[end],arr[0]#將末尾值與根互換，使最末值成為新根
        end-=1
    return arr
```

以下為檢驗1:

```
In [10]: arr1=[7,5,6,2,0,10,22]
         heapsort(arr1)
Out[10]: [0, 2, 5, 6, 7, 10, 22]
```

以下為檢驗2:

```
In [11]: arr2=[22,1,9,2,10,100,0,12,11,14,16,10]
         heapsort(arr2)
Out[11]: [0, 1, 2, 9, 10, 10, 11, 12, 14, 16, 22, 100]
```

heap sort 之基本目標為將所以節點都作為根構建一遍二叉樹，故採用 while 使得每一輪的末尾值都能成為新的根； 在將新的 root 最為 start 構建二叉樹，如此反復，最終形成一個有順序的排列。

3、參考網址

（1）https://baike.baidu.com/item/堆排序/2840151?fr=aladdin

（2）https://blog.csdn.net/qq_37653144/article/details/78449021