Problem Set 4

Problem 1

$Step1$: 首先将数组中的元素先建立成为一个堆。然后从最底部的叶子节点开始调换位置。将25、15和4看作一个子堆。15和4均小于25，所以位置不变。

$Step2$：将13、25、7、15和4看作一个子堆。将25、7和13进行比较，$25 > 13 > 7$，所以调换25和13的位置；将13、15和4进行比较，得到$15 > 13 > 4$，

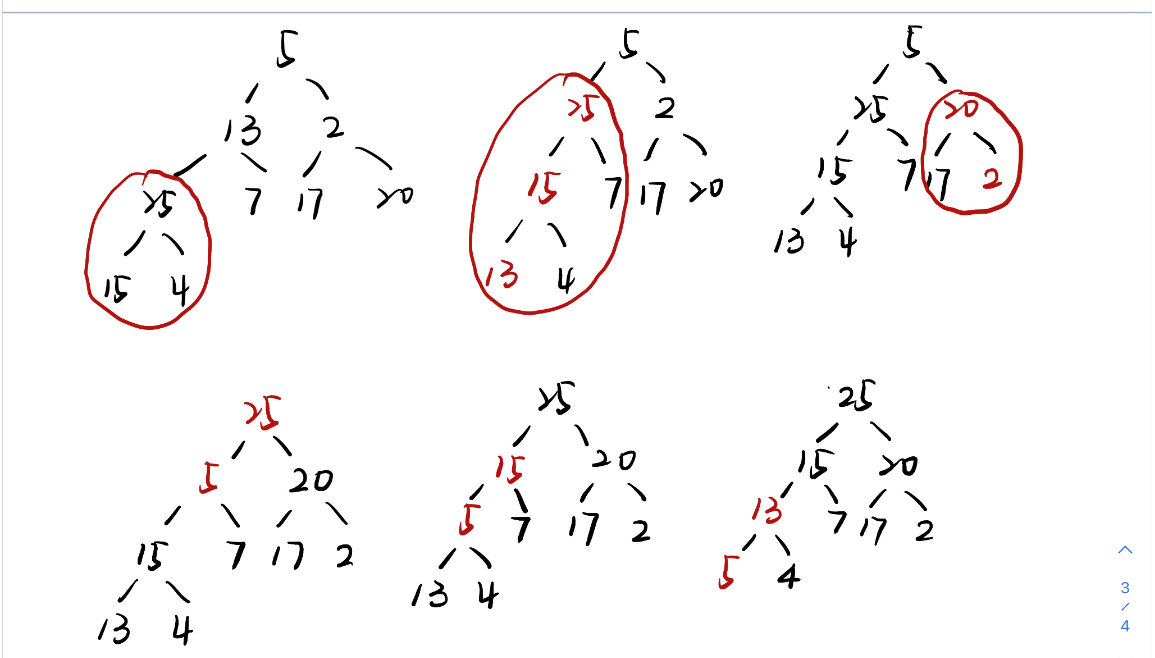
$Step3$：将2、17、20看作一个子堆。比较三个数的大小，得到$2 < 17 < 20$，所以将20和2调换位置。此时子堆应为$20 - 17 - 2$.

$Step4$：看整个大堆，将5、20、25进行比较，得到$5 < 20 < 25$，所以交换25和5的位置；

将5、15、7进行比较，得到$5 < 7 < 15$，所以交换15和5的位置；

将20、17、2进行比较，符合大顶堆，所以不做改变。

将5、13和4进行比较，得到$13 > 5 > 4$，所以交换13和5的位置。



此数组为$array$。

$Step1$：取出顶部元素25，$array[0] = 25$

同时对取出最大元素的堆进行修复：将最底部元素4放到顶部，与15、20进行比较，调换4和20的位置；与17、2进行比较，调换17和4的位置。

$Step2$：取出最大元素20，$array[1] = 20$；

同时对取出最大元素后的堆进行修复：将底部元素5放到顶部，与15、17进行比较，调换5和17的位置；与4、2进行比较，不改变位置。

$Step3$：取出最大元素17，$array[2] = 17$；

同时对取出最大元素的堆进行修复：将底部元素2放到顶部，与15、5进行比较，调换15和2的位置；与13、7进行比较，调换2和13的位置。

$Step4$：取出最大元素15，$array[3] = 15$；

同时对取出最大元素的堆进行修复：将底部元素4放到顶部，与13、5进行比较，调换4和13的位置；与2和7比较，调换4和7的位置。

$Step5$：取出最大元素13，$array[4] = 13$；

同时对取出最大元素的堆进行修复：将底部元素4放到顶部，与7、5比较，调换4和7的位置；与2比较，不改变位置。

$Step6$：取出最大元素7，$array[5] = 7$；

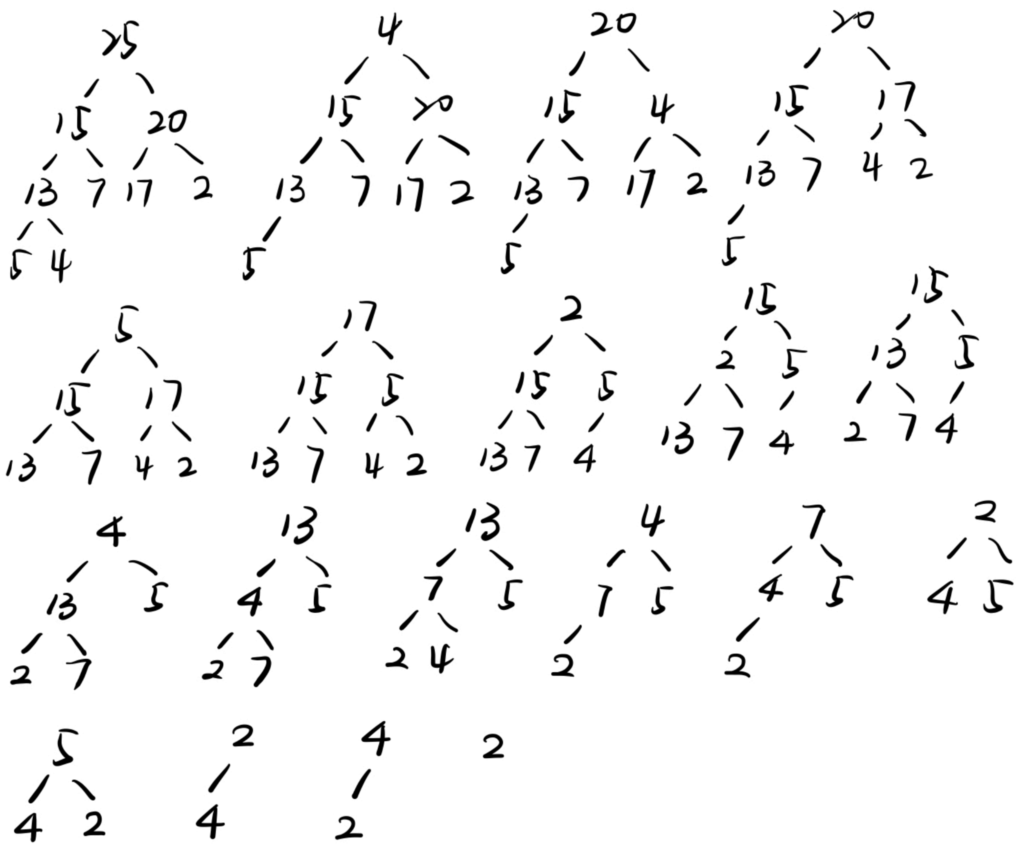
同时对取出最大元素后的堆进行修复：将底部元素2放到顶部，与4、5进行比较，调换5和2的位置。

$Step7$：取出最大元素5，$array[6] = 5$；

同时对取出最大元素后的堆进行修复：将底部元素2放到顶部，与4比较，交换位置。

$Step8$：取出最大元素4，$array[7] = 4$；

同时得到$array[8] = 2$。



Problem 2

Brief description:

假设列表按从大到小的顺序排列，目标也就是将所有的列表合并并且按照从大到小的顺序排列。

先从排列好的列表里依次取出第一个元素，也就是最大的元素，构造一个大顶堆。然后对大顶堆进行$HeapExtractMax$操作，取出的这个元素作为所有列表

Pseudocode:

$MergeAndSorted(A[a_1 ... a_k])$

$New\ array\ S[1 \cdots k], R[1 \cdots nk]$

for $i = 1$ **to** k

  $S.add(a_i[1])$

$HeapSize = k$

for $i = Floor(n/2)$ **down to** 1

  $MaxHeapify(i)$

for $i = 1$ **to** $n - 1$

  $R.add(HeapExtractMax())$

  $HeapInsert(a_j[k])$ **when** $a_j[k] < R[i]\ and\ a_j[k] > others\ in\ a_j$

 return R

Time Analysis:

将每个数组的最大元素提取并合成大顶堆需要的时间是$O(k)$

进行$HeapInsert$和$HeapExtract$操作的时间复杂度均为$O(lgk)$，总共进行了$n-1$次，所以时间复杂度是$O(nlogk)$。

故不难看出$T = O(k + nlogk) = O(nlogk)$

Problem 3

(a) Prove

Basis: 当$n = 2$时，根据$Unusual$内容中$n == 2$的情况，可以实现两个元素从小到大的排序

Inductive Hypothesis:

假设当$n = k$时，算法也可以实现排序，即通过以上算法，这k个元素可以按照从小到大的顺序完成排序。

Inductive Step:

当$n = 2k$时，前k个元素已经完成了排序，所以$Cruel(A[1 \cdots k])$和$Cruel(A[k+1 \cdots 2k])$得到的结果就是前后k个数字已经分别按序排好。在$Unusual$操

正确性得证

(b)

假设没有循环步骤，$Unusual$执行的操作就只有在$n = 2$时交换两个元素的位置，但是当整个数组执行到$Unusual$这一步时，左边一半和右边一半都是已经

(C)

假设先对数组的中间部分进行$Uusual$，再对第二部分数组进行操作的话，还是不能实现将两个数组连接起来的情况。

(d)

Running Time $of\ Unusual$

$T(n) = T(\frac{n}{2}) + O(1 \cdot \frac{n}{2})$

$= O(nlgn)$

Running Time $of\ Cruel$

$T(n) = T(\frac{n}{2}) + O(nlgn)$

$= O(nlg^2n)$

Problem 4

(a)实质上是将原快速排序的一个递归过程更改为迭代过程进行优化。

Basis: 当A数组中只有一个元素时，$p = r$，显然可以实现排序。

Inductive Hypothesis:

假设当$k \in [1, n-1]$时，算法都可以实现排序。

Inductive Step:

则当$k = n$时，根据算法中的$line2$，将中间点赋给q，根据假设，已经能够对左边的子数组$A[p \cdots q-1]$实现排序，接着将q值加一赋给p，操作的范围就变成

(b)要求排序的数组已经按照顺序排好

(c)

Brief Overview:

对递归的范围进行调整，通过比较划分后的子数组，选择较小的数组进行递归，较大的数组进行迭代。

Pseudocode:

$NewSort(A, p, r)$

1 **while** $p < r$

2 $\quad q \leftarrow Partition(A, p, r)$

3 $\quad$ **if** $q > (r-p)/2$

4 $\quad\quad NewSort(A, q+1, r)$

5 $\quad\quad r \leftarrow q-1$

6 $\quad$ **else** $NewSort(A, p, q-1)$

7 $\quad\quad p \leftarrow q+1$

Explanation:

最坏的情况是要求排序的数组是逆序的。每次对较小的数组进行递归，总共需要递归lgn次，每次传入的参数为数组，空间复杂度为$O(1)$，所以空间复杂度为

Problem 5

(a)

Pseudocode:

$Sort(A)$

1 **for** $k = 0$ **to** $n - \sqrt{n}$

2 $\quad SqrtSort(k)$

3 $Sort(A[1 \cdots n - \sqrt{n} + 1])$

Correctness:

类似于冒泡排序。先对第一个到第$\sqrt{n}$个元素进行$SqrtSort$，然后对第二个到第$\sqrt{n}+1$个元素进行$SqrtSort \ldots\ldots$如此循环，循环后可以确定最大的$\sqrt{n} -$

Call-time:

在最坏的情况下需要调用$4n$次$SqrtSort$

Problem 6

(a) 设返回1的概率为p，则返回0的概率为$1-p$。

可以得到等式：$1-p=\frac{1}{2}+\frac{1}{2}p$

解得$p=\frac{1}{3}$

(b)

Problem 7

(a) $lower\ bound$: 999999次(每次说一个数字并问是不是正确答案，问了999999次都不是)

正确性：最低效的方法就是一个一个的问。

$upper\ bound$:

$num=500000; p=1; r=1000000$

$UpperBound(num)$:

if "no" $for\ both$ "$larger\ than\ result?\ and$ "$smaller\ than\ result?$"

return num

else

  if "yes" for "$larger\ than\ result?$"

    $UpperBound(\frac{p+num-1}{2})$

    $r=num-1$

  else "no" for "$larger\ than\ result?$"

    $UpperBound(\frac{num+1+r}{2})$

    $p=num+1$

正确性：通过二分法缩小范围，最多只要询问19次