华东师范大学期中试卷

2019 — 2020 学年第 一 学期

课程名称：__算法导论________

学生姓名：____________________　　　　学　　号：____________________

专　　业：__软件工程________　　　　年级/班级：__2016 级________

课程性质：专业必修

一	二	三	四	五	总分	**阅卷人签名**

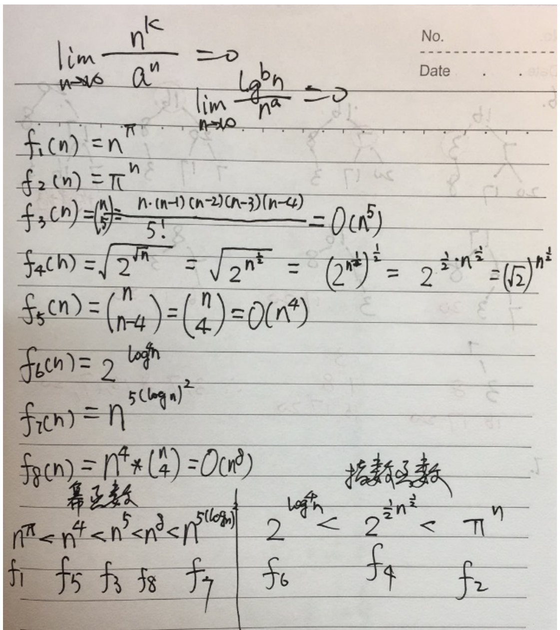
一、渐近分析（共 20 分）。

1、根据增长的阶来排序下列函数，即找到函数的一种排列g_1, g_2, g_3, g_4，g_5, g_6, g_7, g_8 ,使得$g_i = O(g_{i+1})$，i = 1 ⋯ 7。（8 分）

$f_1(n) = n^{\pi}$ ， $f_2(n) = \pi^n$ ， $f_3(n) = \binom{n}{5}$ ， $f_4(n) = \sqrt{2^{\sqrt{n}}}$

$f_5(n) = \binom{n}{n-4}$ ，$f_6(n) = 2^{log^4 n}$ ，$f_7(n) = n^{5(logn)^2}$ ，$f_8(n) = n^4 * \binom{n}{4}$

Solution: $f_1(n), f_5(n), f_3(n), f_8(n), f_7(n), f_6(n), f_4(n), f_2(n)$



No.

Date

$\lim_{n\to\infty} \frac{n^k}{a^n} = 0$

$\lim_{n\to\infty} \frac{lg^b n}{n^a} = 0$

$f_1(n) = n^{\pi}$

$f_2(n) = \pi^n$

$f_3(n) = \binom{n}{5} = \frac{n\cdot(n-1)(n-2)(n-3)(n-4)}{5!} = O(n^5)$

$f_4(n) = \sqrt{2^{\sqrt{n}}} = \sqrt{2^{n^{\frac{1}{2}}}} = (2^{n^{\frac{1}{2}}})^{\frac{1}{2}} = 2^{\frac{1}{2}\cdot n^{\frac{1}{2}}} = (\sqrt{2})^{n^{\frac{1}{2}}}$

$f_5(n) = \binom{n}{n-4} = \binom{n}{4} = O(n^4)$

$f_6(n) = 2^{log^4 n}$

$f_7(n) = n^{5(\log n)^2}$

$f_8(n) = n^4 * \binom{n}{4} = O(n^8)$

幂函数　　　　　　　　指数函数

$n^{\pi} < n^4 < n^5 < n^8 < n^{5(\log n)^2}$ | $2^{log^4 n} < 2^{\frac{1}{2}n^{\frac{1}{2}}} < \pi^n$

f_1 f_5 f_3 f_8 f_7 | f_6 f_4 f_2

2、判断下面每个论断是正确（T）还是错误（F），并给出简单的说明。（12 分）

（1）二分插入排序（在插入排序过程中，利用二分法去找到每一个插入点）需要 *O(n*logn)*的运算量。

答：

Solution: **False.** While binary insertion sorting improves the time it takes to find the right position for the next element being inserted, it may still take $O(n)$ time to perform the swaps necessary to shift it into place. This results in an $O(n^2)$ running time, the same as that of insertion sort.

（2）在合并排序的递归树中，在树的每个层次上运算代价基本相同。

答：

Solution: **True.** At the top level, roughly n work is done to merge all n elements. At the next level, there are two branches, each doing roughly $n/2$ work to merge $n/2$ elements. In total, roughly n work is done on that level. This pattern continues on through to the leaves, where a constant amount of work is done on n leaves, resulting in roughly n work being done on the leaf level, as well.

（3）在最小堆中，每个元素的下一个最大元素，可以在 *O(logn)*时间内找到。

答 ：

Solution: **False.** A min-heap cannot provide the next largest element in $O(\log n)$ time. To find the next largest element, we need to do a linear, $O(n)$, search through the heap's array.

二、递归分治策略（共 20 分）。

3、找出下面递归式的渐近解，用θ符号表示你的答案，并给出简单的理由。

(1) $T(n)=\log n + T(\sqrt{n})$ (2) $T(n)=4\,T(n/2) + n^2 * \sqrt{n}$

解答：(1)

Solution: $T(n) = \Theta(\log n)$.

To see this, note that if we expand out $T(n)$ by continually replacing $T(n)$ with its formula, we get:

$$\begin{aligned}
T(n) &= \log n + \log\sqrt{n} + \log\sqrt{\sqrt{n}} + \log\sqrt{\sqrt{\sqrt{n}}} + \dots \\
&= \log n + \tfrac{1}{2}\log n + \tfrac{1}{2}\log\sqrt{n} + \tfrac{1}{2}\log\sqrt{\sqrt{n}} + \dots \\
&= \log n + \tfrac{1}{2}\log n + \tfrac{1}{4}\log n + \tfrac{1}{8}\log n + \dots \\
&= \Theta(\log n)
\end{aligned}$$

（2）

We have $f(n) = n^2\sqrt{n} = n^{5/2}$ and $n^{\log_b a} = n^{\log_2 4} = n^{\lg 2}$. Since $n^{5/2} = \Omega(n^{\lg 2+3/2})$, we look at the regularity condition in case 3 of the master theorem. We have $af(n/b) = 4(n/2)^2\sqrt{n/2} = n^{5/2}/\sqrt{2} \le cn^{5/2}$ for $1/\sqrt{2} \le c < 1$. Case 3 applies, and we have $T(n) = \Theta(n^2\sqrt{n})$.

三、理解堆的算法（共 15 分）。

4、堆排序算法中，需要调用 MAX-HEAPIFY 过程，以维护最大堆性质。现有数组 A，对一棵以 i 为根结点、大小为 n 的子树，MAX-HEAPIFY（A，i）主要代价包括（1）调整代价θ（1）；（2）在一棵以 i 的（左/右）孩子为根结点的子树上运行 MAX-HEAPIFY 的时间代价。请说明：

（1）每个孩子的子树的大小至多为 2n/3（最坏情况发生在树的最底层恰好半满的时候）。（10 分）

（2）对一棵树高为 h 的结点来说，MAX-HEAPIFY 的时间复杂度时 O(h)。（5 分）

解：（1）根据二叉树的性质，从根结点开始每次分成两支，每层填满后才开始下一层，所以最坏情况发生在树的最底层（h 层）恰好半满的时候。h 层的叶子结点数是 h-1 层叶子结点数的两倍。对 n 个结点的二叉树而言，有 n/2 个叶子结点，所以 h 层的叶子结点数为 n/2*2/3=n/3。

所以小分支的结点数为 （n-n/3）/2=n/3。

所以，每个孩子的子树大小最多为 n-n/3=2n/3。

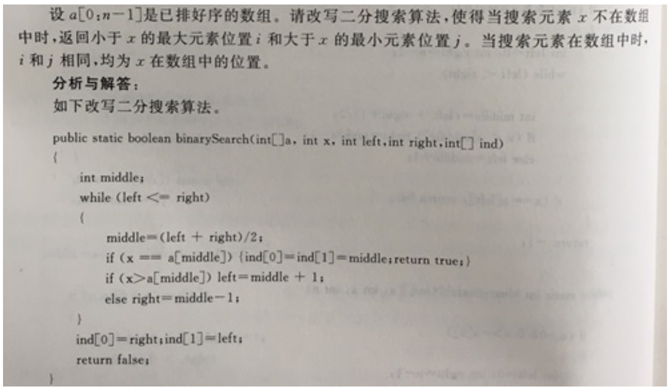
（2）运行时间 T(n) <= T(2n/3)+ θ（1），

由主定理计算，可得 T(n)=O（lgn）。如果树高为 h，则 h=lgn，所以 MAX-HEAPIFY 的时间复杂度时 O(h)。

四、算法设计与实现（共 45 分）。

5、设 a[0:n-1]是已排好序的数组。请改写二分搜索算法，使得当搜索元素 x 不在数组中时，返回小于 x 的最大元素位置 i 和大于 x 的最小元素位置 j。当搜索元素在数组中时，i 和 j 相同，均为 x 在数组中的位置。请

（1）写出算法伪代码；（2）以类 C++或 java 风格写出算法代码。



设 $a[0:n-1]$是已排好序的数组。请改写二分搜索算法，使得当搜索元素 x 不在数组中时，返回小于 x 的最大元素位置 i 和大于 x 的最小元素位置 j。当搜索元素在数组中时，i 和 j 相同，均为 x 在数组中的位置。

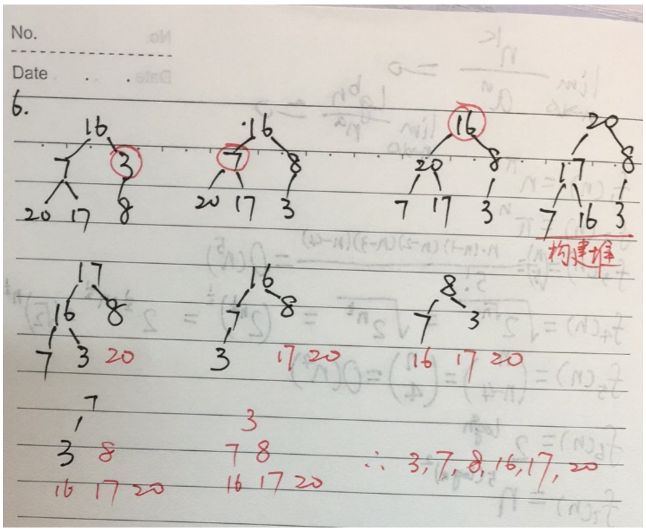
分析与解答：

如下改写二分搜索算法。

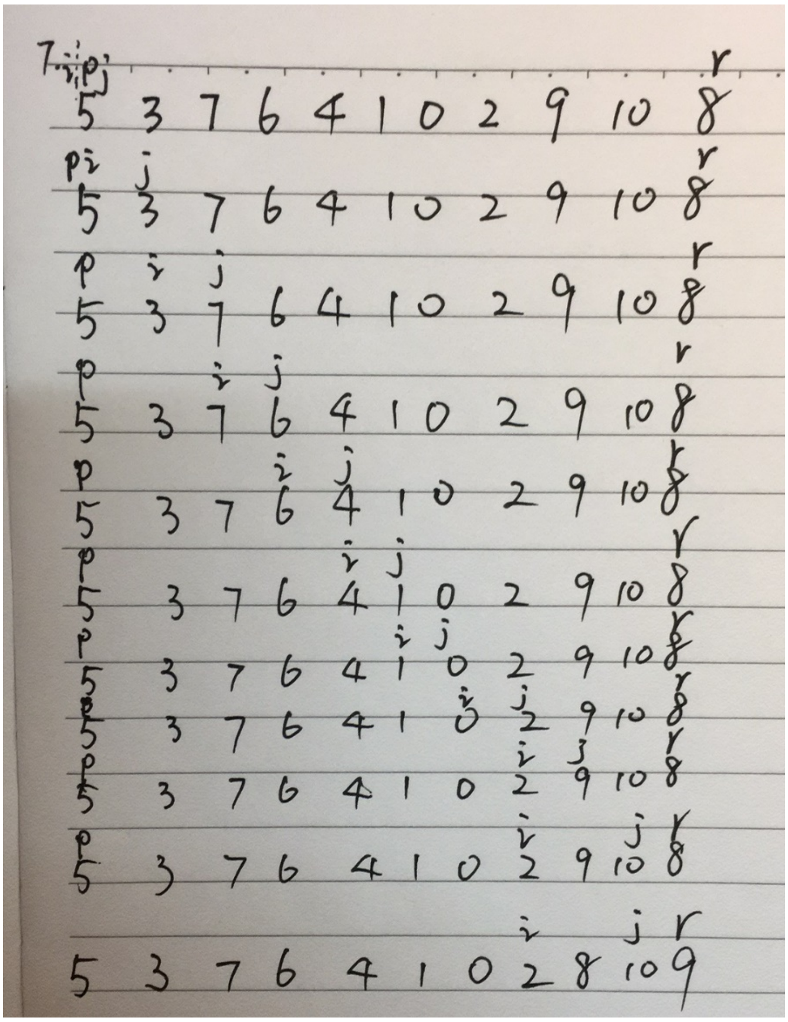
```
public static boolean binarySearch(int[]a, int x, int left,int right,int[] ind)
{
    int middle;
    while (left <= right)
    {
        middle=(left + right)/2;
        if (x == a[middle]) {ind[0]=ind[1]=middle;return true;}
        if (x>a[middle]) left=middle + 1;
        else right=middle-1;
    }
    ind[0]=right;ind[1]=left;
    return false;
}
```

返回的 ind[0]是小于 x 的最大元素位置，ind[1]是大于 x 的最小元素位置。

6、给定一个整型数组 a[6]={16,7,3,20,17,8}，对其进行堆排序。请图示堆排序的全过程。



7、以数组 a[11]={ 5，3，7，6，4，1，0，2，9，10，8 }为例，说明快速排序算法过程中，PARTITION 过程第一次运行的过程。请以 a[11]为主元，分步图示运算结果。



7. i p j r
5 3 7 6 4 1 0 2 9 10 8
p i j r
5 3 7 6 4 1 0 2 9 10 8
p i j r
5 3 7 6 4 1 0 2 9 10 8
p i j r
5 3 7 6 4 1 0 2 9 10 8
p i j r
5 3 7 6 4 1 0 2 9 10 8
p i j r
5 3 7 6 4 1 0 2 9 10 8
p i j r
5 3 7 6 4 1 0 2 9 10 8
p i j r
5 3 7 6 4 1 0 2 9 10 8
p i j r
5 3 7 6 4 1 0 2 9 10 8
p i j r
5 3 7 6 4 1 0 2 9 10 8
i j r
5 3 7 6 4 1 0 2 8 10 9

五、拓展题。算法分析（共 20 分）。

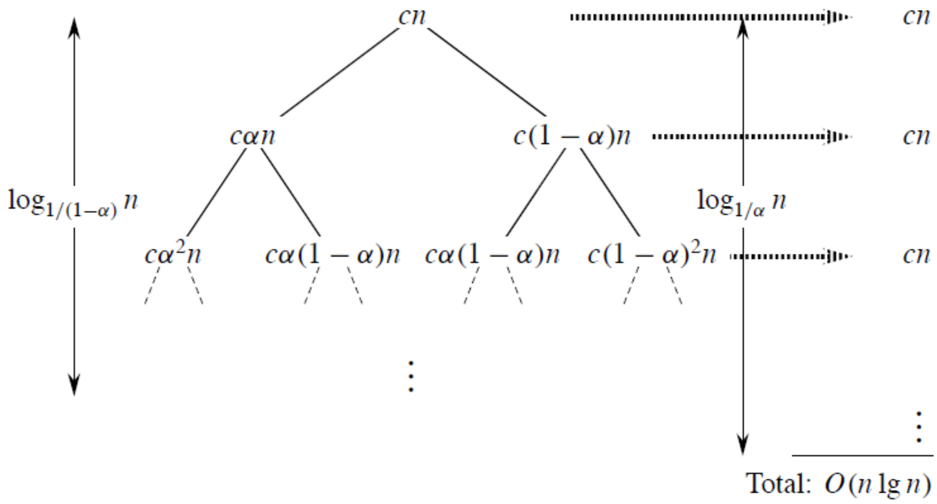
8、利用递归树给出下列递归式的渐近紧确解，

$$T(n)=T(\alpha n)+T((1-\alpha)n)+cn,$$

其中α 是一个常数，0< α <1; c 是常数，c>0。

答：

Without loss of generality, let $\alpha \geq 1-\alpha$, so that $0 < 1-\alpha \leq 1/2$ and $1/2 \leq \alpha < 1$.



The recursion tree is full for $\log_{1/(1-\alpha)} n$ levels, each contributing cn, so we guess $\Omega(n \log_{1/(1-\alpha)} n) = \Omega(n \lg n)$. It has $\log_{1/\alpha} n$ levels, each contributing $\leq cn$, so we guess $O(n \log_{1/\alpha} n) = O(n \lg n)$.

Now we show that $T(n) = \Theta(n \lg n)$ by substitution. To prove the upper bound, we need to show that $T(n) \le dn \lg n$ for a suitable constant $d > 0$.

$$\begin{aligned}
T(n) &= T(\alpha n) + T((1-\alpha)n) + cn \\
&\le d\alpha n \lg(\alpha n) + d(1-\alpha)n \lg((1-\alpha)n) + cn \\
&= d\alpha n \lg \alpha + d\alpha n \lg n + d(1-\alpha)n \lg(1-\alpha) + d(1-\alpha)n \lg n + cn \\
&= dn \lg n + dn(\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)) + cn \\
&\le dn \lg n \,,
\end{aligned}$$

if $dn(\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)) + cn \le 0$. This condition is equivalent to

$$d(\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)) \le -c \,.$$

Since $1/2 \le \alpha < 1$ and $0 < 1-\alpha \le 1/2$, we have that $\lg \alpha < 0$ and $\lg(1-\alpha) < 0$. Thus, $\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha) < 0$, so that when we multiply both sides of the inequality by this factor, we need to reverse the inequality:

$$d \ge \frac{-c}{\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)}$$

or

$$d \ge \frac{c}{-\alpha \lg \alpha + -(1-\alpha)\lg(1-\alpha)} \,.$$

The fraction on the right-hand side is a positive constant, and so it suffices to pick any value of d that is greater than or equal to this fraction.

To prove the lower bound, we need to show that $T(n) \ge dn \lg n$ for a suitable constant $d > 0$. We can use the same proof as for the upper bound, substituting $\ge$ for $\le$, and we get the requirement that

$$0 < d \le \frac{c}{-\alpha \lg \alpha - (1-\alpha)\lg(1-\alpha)} \,.$$

Therefore, $T(n) = \Theta(n \lg n)$.