No:
Date:

1-3: $2,\ \log n,\ n^{\frac{2}{3}},\ 20n,\ 4n^2,\ 3^n,\ n!$

1-4: (1) 设第一台、第二台计算机 t 秒内可解决输入规模为 $n_1$、$n_2$ 的问题

$$\frac{3\times 2^{n_1}}{3\times 2^{n_2}} = \frac{1}{64}$$

$$n_2 = n_1 + 6$$

(2) $$\frac{n_1^2}{n_2^2} = \frac{1}{64}$$

$$n_2 = 8n_1$$

(3) 由于 $T(n)=8$，8为常数，所以计算机可处理任意规模的问题。

1-7: 由 Stirling's approximation

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)$$

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \cdot e^{\alpha_n} \quad \left(\frac{1}{12n+1} < \alpha_n < \frac{1}{12n}\right)$$

$$\lim_{n\to\infty}\frac{n!}{n^n} = \lim_{n\to\infty}\frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\cdot e^{\alpha_n}}{n^n} = \lim_{n\to\infty}\sqrt{2\pi n}\left(\frac{1}{e}\right)^n\cdot e^{\alpha_n} = 0$$

$\therefore n! = o(n^n)$.

No:
Date:

1-8: $$T_{avg}(n) = \sum_{size(I)=n} p(I)\,T(I) = \Theta(f(n))$$

即 $\exists c_1, c_2 > 0, n_0$ 使 $\forall n > n_0$ 有：

$$c_1 f(n) \le \sum_{size(I)=n} p(I)\,T(I) \le c_2 f(n)$$

于是有 $\exists c > 0, n_0 > 0$，使 $\forall n > n_0$，有

$$0 \le c f(n) \le \sum_{size(I)=n} p(I)\,T(I)$$

又 $\because \sum_{size(I)=n} p(I)\,T(I) \le T_{max}(n)$

即 $0 \le c f(n) \le T_{max}(n)$

故 $T_{max}(n) = \Omega(f(n))$

1-11: $3n^2+10n = O(n^2)$

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

$21 + \frac{1}{n} = O(1)$

$\log n^3 = O(\log n)$

$10\log 3^n = O(n)$

1-12: 由 O 定义知 $O(1) = O(2)$

二者表达同一函数时，常数不同.

No:
Date:

2-3
```
template <class Type>
int binarySearch(Type a[], const Type& x, int left,
    int right, int &i, int &j){
 int middle;
 while(left <= right){
     middle = (left + right)/2;
     if(x == a[middle];
     return i;
     i=j=middle;
     return 1;
     }
     if(x > a[middle])
         left = middle+1;
     else
         right = middle-1;
     }
    i=right;
    j=left;
    return 0;
}
```

No:
Date:

2-4. 将 n 分成 $\frac{n}{m}$ 段，每段 m 位。
计算一次 m 位乘法，耗时 $O(m^{\log 3})$
(采用分治法)
计算结果为各结果拼接的结果
因此此，耗时为 $O(\frac{n}{m} \cdot m^{\log 3})$，即 $O(nm^{\log(\frac{3}{2})})$

2-6.
将 n 阶矩阵分块为 $m \times m$ 的矩阵。用传统方法求两个 m 阶矩阵的乘积需要计算 $O(m^3)$ 次 2 个 $2^k \times 2^k$ 矩阵的乘积。用 Strassen 矩阵乘法计算两个 $2^k \times 2^k$ 矩阵的乘积需要的计算时间为 $O(7^k)$，因此算法的计算时间为 $O(7^k m^3)$

2-7 $P(x) = \prod_{i=1}^{d}(x-n_i) = \prod_{i=1}^{d/2}(x-n_i)\prod_{i=d/2+1}^{d}(x-n_i)$

$= P_1(x)P_2(x)$

d 次多项式转化为两个 $\frac{d}{2}$ 次多项式的乘积.
计所需时间为 $T(d)$.

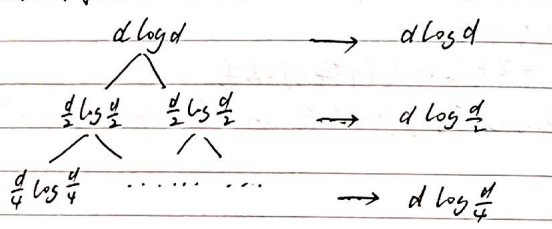
$$T(d) = \begin{cases} O(1) & d=1 \\ 2T(d/2) + O(d\log d) \end{cases}$$

$2T(d/2)$ 为两个 $\frac{d}{2}$ 次多项式乘积的计算时间

No:
Date:

$O(d\log d)$ 为两个 $\frac{d}{2}$ 次多项式的乘积所需时间

画出递归树：




树长为 $\log d$

$$T(d) = d\log d + d\log\frac{d}{2} + \cdots + d\log\frac{d}{2^{\log d-1}} + O(d)$$

$$= d\left(\log\frac{d^{\log d}}{2^{(\log d(\log d-1))/2}}\right) + O(d)$$

$$= d\cdot\log d\cdot\left(\log\frac{d}{2^{\frac{\log d-1}{2}}}\right) + O(d)$$

$$= d\cdot\log d\cdot\left(\log d - \frac{\log d-1}{2}(\log 2)\right) + O(d)$$

$$= O(d\log^2 d)$$

2-13 将算法 Partition 的不等号反向

```
template<class Type>
int Partition(Type a[], int p, int r){
    int i=p, j=r+1;
    Type x=a[p];
    while(true){
        while(a[++i] > x && i<r);
        while(a[--j] < x);
        if(i>=j)
            break;
        Swap(a[i], a[j]);
    Swap(a[j], a[p]);
    return j;
}
```

算法实现题 众数问题

问题描述:

给定含有 n 个元素的多重集合 S,每个元素在 S 中出现的次数称为该元素的重数。多重集 S 中重数最大的元素称为众数。例如,S={1,2,2,2,3,5}。多重集 S 的众数是 2,其重数为 3。

算法设计:

对于给定的由 n 个自然数组成的多重集 S,编程计算 S 的众数及其重数。

数据输入:

输人数据由文件名为 input txt 的文本文件提供。

文件的第 1 行为多重集 S 中元素个数 n;在接下来的 n 行中,每行有一个自然数。

结果输出:

程序运行结束时,将计算结果输出到文件 output.txt 中。输出文件有 2 行,第 1 行是众数。第 2 行是重数。

解答：核心代码：

```
public static void searchMode(int[] arr, int first, int last) {
//        等于中间值元素的元素个数
    int midRepeatTimes = midRepeatTimes(arr, first, last);
//        等于中间值元素的第一个元素下标
    int midFirstIndex = midFirstIndex(arr, first, last);
    if (midRepeatTimes > repeatTimes) {
        mode = arr[midFirstIndex];
        repeatTimes = midRepeatTimes;
    }
    if (midFirstIndex > repeatTimes) {
        searchMode(arr, first, midFirstIndex);
    }
    if ((last - midFirstIndex - midRepeatTimes) > repeatTimes) {
        searchMode(arr, first: midFirstIndex + midRepeatTimes, last);
    }
}
```

运行结果：

```
D:\Javaaaa\ruanjian\bin\java.exe "-
 .jar=50365:D:\Javaaaa\ruanjian\ide
 D:\study\大三上\算法设计与分析\homewo
6
1 2 3 3 5 6
3
2

Process finished with exit code 0
```

算法实现题 半数集问题

问题描述:

给定一个自然数 n,由 n 开始可以依次产生半数集 set(n)中的数如下:

(1)n 属于 set(n);

(2)在 n 的左边加上一个自然数,但该自然数不能超过最近添加的数的一半;

(3)按此规则进行处理,直到不能再添加自然数为止。

例如,set(6)={6,16,26,126,36,136}。半数集 set(6)中有 6 个元素。注意,该半数集是多重集。

算法设计:

对于给定的自然数 n,编程计算半数集 set(m)中的元素个数。

数据输入:

输入数据由文件名为 input.txt 的文本文件提供。每个文件只有一行,给出整数 n(0<n<1000)。

结果输出:

程序运行结束时,将计算结果输出到文件 output.txt 中。输出文件只有一行,给出半数集 set(n)中的元素个数。

解答：核心代码：

直接递归：

```
static long halfSetNumber01(int n) {
    long result = 1L;
    for (int i = 1; i <= n / 2; i++) {
        result += halfSetNumber01(i);
    }
    return result;
}
```

数组存储中间数据：

```
static long halfSetNumber02(int n) {
    long[] halfSet02 = new long[n];
    halfSet02[0] = 1L;
    return halfSetNumber02(n, halfSet02);
}

static long halfSetNumber02(int n, long[] halfSet02) {
    if (halfSet02[n-1]>0){
        return halfSet02[n-1];
    }
    long result = 1L;
    for (int i = 1; i <= n / 2; i++) {
        result += halfSetNumber02(i, halfSet02);
    }
    halfSet02[n - 1] = result;
    return result;
}
```

迭代：

```
static long halfSetNumber03(int n) {
    long[] halfSet03 = new long[n+1];
    for (int i = 0; i <= n/2; i++) {
        halfSet03[i] = 1L;
    }
    for (int i = 1; i <= n / 2; i++) {
        for (int j = i * 2; j <= n; j++) {
            halfSet03[j]+=halfSet03[i];
        }
    }
    return halfSet03[n]+1;
}
```

运行结果：

```
D:\Javaaaa\ruanjian\bin\java.exe "-ja
 .jar=49776:D:\Javaaaa\ruanjian\ideal
 D:\study\大三上\算法设计与分析\homework
6
6 程序运行时间：0ms
6 程序运行时间：0ms
6 程序运行时间：0ms
1000
1981471878 程序运行时间：3185ms
1981471878 程序运行时间：1ms
1981471878 程序运行时间：2ms
2000
264830889564 程序运行时间：428416ms
264830889564 程序运行时间：0ms
264830889564 程序运行时间：1ms
```

代码附录：

众数：

```
import java.util.Scanner;

public class HW0102 {
   public static int mode; //众数
   public static int repeatTimes; //重数

   //找到等于中间值元素的元素个数
   public static int midRepeatTimes(int[] arr, int left, int right) {
      int sum = 0;
      int mid = (left + right) >>> 1;
      int n = arr[mid];
      for (int i = left; i < right; i++) {
         if (arr[i] == n) {
            sum++;
         }
      }
      return sum;
   }

   //找出等于中间值元素的第一个元素下标
   public static int midFirstIndex(int[] arr, int left, int right) {
      int x = 0;
      int mid = (left + right) >>> 1;
      for (int i = left; i < right; i++) {
         if (arr[mid] == arr[i]) {
            x = i;
            break;
         }
      }
      return x;
   }

   public static void searchMode(int[] arr, int first, int last) {
//       等于中间值元素的元素个数
      int midRepeatTimes = midRepeatTimes(arr, first, last);
//       等于中间值元素的第一个元素下标
      int midFirstIndex = midFirstIndex(arr, first, last);
      if (midRepeatTimes > repeatTimes) {
         mode = arr[midFirstIndex];
         repeatTimes = midRepeatTimes;
      }
      if (midFirstIndex > repeatTimes) {
         searchMode(arr, first, midFirstIndex);
      }
      if ((last - midFirstIndex - midRepeatTimes) > repeatTimes) {
         searchMode(arr, midFirstIndex + midRepeatTimes, last);
      }
   }

   public static void main(String[] args) {
      Scanner input = new Scanner(System.in);
      int n = input.nextInt();
      int[] arr = new int[n];
      for (int i = 0; i < arr.length; i++) {
         arr[i] = input.nextInt();
      }
      searchMode(arr, 0, n);
      System.out.println(mode);
      System.out.println(repeatTimes);
   }
}
```

半数集：

```
import java.util.Scanner;

public class HalfSet {
   public static Long[] halfSet02 = new Long[10000];
   int k = 0;

   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      int n = scanner.nextInt();
      long startTime = System.currentTimeMillis();
      long endTime = System.currentTimeMillis();
      while (n != 0) {
         startTime = System.currentTimeMillis();   //获取开始时间
         System.out.print(halfSetNumber01(n));
         endTime = System.currentTimeMillis();   //获取结束时间
         System.out.println(" 程序运行时间: " + (endTime - startTime) + "ms");   //输出程序运行时间

         startTime = System.currentTimeMillis();
         System.out.print(halfSetNumber02(n));
         endTime = System.currentTimeMillis();
         System.out.println(" 程序运行时间: " + (endTime - startTime) + "ms");

         startTime = System.currentTimeMillis();
         System.out.print(halfSetNumber03(n));
         endTime = System.currentTimeMillis();
         System.out.println(" 程序运行时间: " + (endTime - startTime) + "ms");

         n=scanner.nextInt();
      }
   }

   static long halfSetNumber01(int n) {
      long result = 1L;
      for (int i = 1; i <= n / 2; i++) {
```

```
            result += halfSetNumber01(i);
        }
        return result;
    }


    static long halfSetNumber02(int n) {
        long[] halfSet02 = new long[n];
        halfSet02[0] = 1L;
        return halfSetNumber02(n, halfSet02);
    }

    static long halfSetNumber02(int n, long[] halfSet02) {
        if (halfSet02[n-1]>0){
            return halfSet02[n-1];
        }
        long result = 1L;
        for (int i = 1; i <= n / 2; i++) {
            result += halfSetNumber01(i);
        }
        halfSet02[n - 1] = result;
        return result;
    }

    static long halfSetNumber03(int n) {
        long[] halfSet03 = new long[n+1];
        for (int i = 0; i <= n/2; i++) {
            halfSet03[i] = 1L;
        }
        for (int i = 1; i <= n / 2; i++) {
            for (int j = i * 2; j <= n; j++) {
                halfSet03[j]+=halfSet03[i];
            }
        }
        return halfSet03[n]+1;
    }
}
```