```
Last login: Wed Dec 9 12:10:00 on ttys001
uts1242015s: 123456789$ cat caution.txt
```

このシケプリは2015年度理科一類24組のアルゴリズム入門過去問解答です。

2013年および2014年のアルゴリズム入門（旧情報科学）の共通問題の解答を掲載しています。以下の解答はあくまで一例ですので他の解答も存在する場合がありますが一つのみの掲載としています。

この解答独自のこととして関数の返り値の文に本授業では教えられていないreturn文を使用しています。これは作者がこちらのほうが慣れているためつけているだけで、実際の答えとしてはreturnを取り除いて構いません。予めご了承のほどよろしくお願いします。

2015年理一24組 touyou(@touyoubuntu)

```
uts1242015s: 123456789$ isrb
Picked up _JAVA_OPTINONS: -Dfile.encoding=UTF-8
isrb(main):001:0> load “2013_exam.rb”
true
isrb(main):002:0> ans(“第一問”)
```

(a)ア from[n-1] イ (n+1)/2 ウ return from[0]

(b)[[7],[2],[8],[3],[1],[4],[6],[5]] → [[2,7],[3,8],[1,4],[5,6]]
→ [[2,3,7,8],[1,4,5,6]] → [[1,2,3,4,5,6,7,8]]

(c)単純整列法は計算量がO(n^2)、空間計算量がO(1)

一方併合整列法が計算量がO(n log n)、空間計算量がO(n)なので単純整列法のほうがメモリ消費が少なく遅い。

```
isrb(main):003:0> ans(“第二問”)
```

(a)i. i=1, a=[1,3,2,5,4] → i=2, a=[1,2,3,5,4] → i=3, a=[1,2,3,5,4]
→ i=4, a=[1,2,3,4,5]

ii. a=[1,2,5,6,3,4]などでは最後にa=[1,2,5,3,4,6]となる

(b)i. a=[2,3,4,5], O(n)

ii. 解説に掲載

```
isrb(main):004:0> ans(“第三問”)
```

(a)解説に掲載

(b)前者は足してから掛け算を、後者は掛けてから足し算をしている。そのため後者に情報落ち誤差が生じて結果の値に差が生まれたと考えられる。

(c)解説に掲載

(d)plus(mul(mul([-1.0,0.0],x),fd(x)),f(x))

```
isrb(main):005:0> ans(“第四問”)
```

(a)解説に掲載

(b)この２つの関数はi,j,kが同じ値になる場合について何度も再帰的に関数を実行してしまうようになっており、それは特にi,j,kの値がお互い近い場合顕著になって最悪計算量がO(3^(n/3))と見積もられるから。

(c)キ 1.0

ク value(a,i-1,j,k)*w(i-1,j,k)+value(a,i,j-1,k)*l(i,j-1,k)
+value(a,i,j,k-1)*d(i,j,k-1)

解説

(なんか思い立った解答デザイン非常に見にくくてすいません…)第一問から解説します。

第一問はソートのアルゴリズムに関する問題です。(a)のようなコードの穴埋め問題に対してはまずそのコードが何をやりたいのか、これをコードを頭のなかで一行一行実行しながら（このような作業をウォークスルーと呼びます。）考えていくことで解答につながります。一例としてこの問題でウォークスルーを試みたいと思います（以後の穴埋め問題に関してはウォークスルーの部分は皆さんにおまかせします。）n=a.l~の行を一行目として説明していきます。

まず関数として受け取った配列aの長さを一行目でnに代入しています。二行目ではそれに対してn個の長さ1の配列をもつ配列fromを定義しています。この時点ではこのfromの正体はわかりません。次に三行目から五行目のforループ内でfromに含まれる配列にaの要素を一個ずつ入れていきます。とりあえずこの時点でfromのi番目の配列には最初aのi番目の要素が入っているのだなということがわかります。続いて六行目から十六行目がこのプログラムのメインのループです。条件文をみるとnが1よりも大きい間中のプログラムを実行するということを把握します。では中身を見ていきましょう。

七行目ではnの半分のサイズ（ただしここでの半分とは偶数の場合は単に1/2の値、奇数の場合は1/2の値の小数点を切り上げた値です。）の配列toを定義しています。そして八行目から十行目のループでこのtoのi番目にfromのi*2+1番目とi*2+2番目を併合したものを代入しています。この時点でtoというのが今のfromの中にある隣同士の配列をそれぞれ併合したものを表すということがわかります。すると続く十一行目から十三行目はnが奇数であると困ったことがあるのでなにかをしているということがわかります。一体なんでしょうか？ヒントは配列のアドレスです。代入しようとしているのはtoの最後の要素、そして直前のforループでfromの(n+1)/2番目だけはmergeされていません。よってアにはfrom[(n+1)/2-1]が入ります。十四行目でfromにtoを代入しているのでこれでtoが一回隣同士を併合した後の配列、fromが操作している配列の現在の状態を示すということがわかってきます。また次の行でnにイを代入していますがこれは今までのことを考えると現在のfromのサイズを持っているということがわかるので入れるのは(n+1)/2です。（あるいはもっと簡単にfrom.length()としてしまっても良いかもしれません）以上より最後に返すべき値はfrom[0]なのでこれをウに入れれば(a)の答えです。ちなみに(b)に関しては今だらだらと言葉で書いてきたことを頭の中で繰り返していきながらエの位置に注意して一回一回自分でループでおこなっている操作をすれば自ずと答えはでます。(c)については特に併合整列法に関してはコードがあるので自分で見積もれたほうが良いですが、限られた時間で解答するということを考えると授業で紹介されたソートの計算量と空間計算量を覚えておくほうが良いと思います。

第二問にうつりましょう。(a)はウォークスルーを実際にしてその問題点をあぶり出す問題です。隣としか交換しないということがわかれば解答例も簡単に思いつけるでしょう。(b)も似たような問題です。minの呼び出し回数が最大になるとは条件分岐がすべて下にいってminをあらたに呼び出す場合だというのがわかればそのような例をあげればいいですし、計算量はそのような場合に何回minが呼び出されるかを考えればよいです（ただし定数倍は通常省略するため実際はO(2n)などになってもO(n)と書きます。）

iiに関してのコードは横のようなものが一例です。このように適宜範囲を2分割していくことで計算量を減らすことができます。計算量の見積もりをするにあたっては二分探索法というアルゴリズムを思い出しましょう。このアルゴリズムは二分探索法の形と似ているなということ、そして二分探索法の仕組みがわかっていればこのアルゴリズムの計算量をO(log n)という風に見積もれると思います。逆に言えば二分探索法を覚えていればそこからこの改善案の着想を得ることもできます。

```
def min(a,i,j)
  if i == j
    return a[i]
  else
    l = min(a, i, i+(j-i)/2)
    r = min(a, i+(j-i)/2+1, j)
    if l < r
      return l
    else
      return r
    end
  end
end
```

さて第三問。まず(a)は以下のようです。

```
def div(x, y)
  d = y[0]*y[0]+y[1]*y[1]
  return [(x[0]*y[0]+x[1]*y[1])/d, (-x[0]*y[1]+x[1]*y[0])/d]
end
```

複素数の割り算はそのままではできないので、分母を有理化することを考えるとこのようなコードになります。わからなかったら実際の式を自分で変形してみてもっともプログラムにしやすい形をさぐるのもいいかもしれません。(b)は情報落ち誤差についての問題です。穴になりやすいので（僕だけかな…?笑）勉強しておきましょう。

続いて(c)のコードを見てみましょう。

```
def f(x)
  return plus(plus(mul(x,x),x),[1.0, 0.0])
end

def fd(x)
  return plus(mul([2.0, 0.0], x), [1.0, 0.0])
end
```

これは式をそのまま関数にするだけです。ただしxは複素数として２つの実数を含んだ配列として表されているために単に数を掛けたり足したりすることはできません。問題に与えられた関数を利用することを考えましょう。そして最後の(d)ですが、これは説明から力にはxでの接線とx軸の交点のx座標を計算すればいいということになります。式で書けばこれはf'(x)*(0-x)+f(x)ですのでそれを(c)同様与えられた関数で表現しましょう。

最後に第四問です。まず(a)はひとまず余計なことは考えず問題文の指示通りにプログラムを書いていきましょう。その際次の(b)を見越してpattern(i,j,k)となるべく似た構造のプログラムを書くという方針でいけば自然とpattern(i,j,k)がどのようなことを行っているかがわかりますしおすすめです。コードの一例は次のページ頭に掲載しています。そして(b)はこれを元に最悪計算量を見積もるという問題と言い換えることができます。最悪計算量であるのはその後の問

```
def probability(i, j, k)
  if i < 0 || j < 0 || k < 0
    return 0.0
  else
    if i == 0 && j == 0 && k == 0
      return 1.0
    else
      return probability(i-1, j, k) * w(i-1, j, k) +
             probability(i, j-1, k) * l(i, j-1, k) +
             probability(i, j, k-1) * d(i, j, k-1)
    end
  end
end
```

題で改善を図るためです。最悪計算量は一般に極端な例の計算量になることが多いので極端な例を探します。この場合は「i,j,kのどれかがnでそれ以外0」もしくは「i,j,kがすべて等しい値」のいずれかです。ですが前者に関してはi,j,kが負になる場合に決められた値を返しそれ以上計算しないようにしているので（このような操作を枝刈りと言います。）後者が最悪な場合となることがわかります。ではその計算量はどの程度でしょうか？イメージがつかみにくかったら小さい値で実際に関数が何回よびだされるか実験してみるのも良いでしょう。一般に再帰関数は下に伸びていく木を想像して計算量を見積もるとわかりやすいです。結果、i,j,kが0になるまで3個ずつ枝分かれしていくので合計で約3^(n/3)回関数を呼び出しているとわかります。精密に計算すればもっと違う値になるかもしれませんが計算量は「大体どの程度」を示したものなのでこれでいいです。nが指数の肩にのっているのでこれはあきらかに効率が悪いですね。その改善を図るのが(c)です。配列をつかっていることからわかるかもしれませんがこれは効率の悪い再帰関数を動的計画法（Dynamic ProgrammingあるいはDPとも）によって改善するというよくある手法を再現するものになっています。動的計画法だとわかれば、いやわからなくてもこれは再帰関数のかわりに配列に保存した値を使って計算していくということならわかるはずなのでキやクに入る式は自ずとわかるでしょう。ここで注意しとかなければならないのはvalue(a,i,j,k)の存在です。普通にx,y,zから1を引いた値で配列を参照しようとするとそれが負の値になったときにエラーとなってしまいます。value関数が何をするのかなということに注意を払っておくことでvalue関数を使ってエラーを回避した正解の式を導くことができると思います。

さて随分文字だらけのシケプリになってしまいました。読みにくいですね…とりあえずひとまずは2013年の解説はこれで終わりということで。忘れないうちに言っておきますが、間違ったことを書いている可能性もあるのでそれは見つけ次第教えてくれると嬉しいなと思います。

続いては2014年度共通問題の解答解説ですが、2013年の解説で色々なことを語ったのでこれよりかは幾分少ない解説で済むと思います（というかこっちで書いたことはすべて省略していくので2014年の解説でわからないところがあったらこちらをみるようにしてください。）

```
isrb(main):006:0> load “2014_exam.rb”
true
isrb(main):007:0> ans(“第一問”)
```

1. (A) a.length()==0 (B) 1..a.length()-1 (C) 1..b.length()-1
   (D) a[0]+b[0]
2. (E) a.length()==0 (F) inner(v,a[0]) (G) 1..a.length()-1
3. (H) a.length()==0 (I) vmmult(a[0],b) (J) 1..a.length()-1

```
isrb(main):008:0> ans(“第二問”)
```

1. P(t,a)=p*P(t-1,a-1)+q*P(t-1,a+1)
2. そのまま再帰的に計算しようとすると同じ引数でPが何度も実行されてしまうから。
3. (A) 0 (B) q*prob[i-1][j+1] (C) p*prob[i-1][j-1]
   (D) p*prob[i-1][j-1]+q*prob[i-1][j+1] (E) prob[t][a]
4. 一番繰り返しの多いループは２つの入れ子になったforループでループ内の計算が合計でt*n回実行されるのでもとめる計算量のオーダーはO(t*n)である。
5. 計算量のオーダーとはその関数の中でもっとも時間のかかる繰り返し処理が何回程度繰り返しを行うかを示したもので、調べることでもっとも効率的なアルゴリズムを選んだり、時間のかかりすぎるプログラムの実行を未然に防ぐことができる。

```
isrb(main):009:0> ans(“第三問”)
```

この問題はすべて解説に解答も含めて書きます。

```
isrb(main):010:0> ans(“第四問”)
```

1. (A) C[i][0] (B) i+1 (C) C[j][1] (D) max
2. O(N^3)
3. rcvr[n-1][m-1]+C[n-1][m-1]
4. 解説に掲載

解説

第一問は支持通りヒントをつかってコードを埋めていく問題です。情報が豊富なのでそれを上手く参考にしていきましょう。if文の条件は再帰の終了条件なのでどこで打ち切ればいいのかを考えればわかると思います。

第二問は再帰関数と動的計画法をみながら計算量について考える問題のようですね、４番と５番がなぜこの順番になっているのか全くもって理解不能なんですがまぁよいでしょう。解答に書いた通りです。

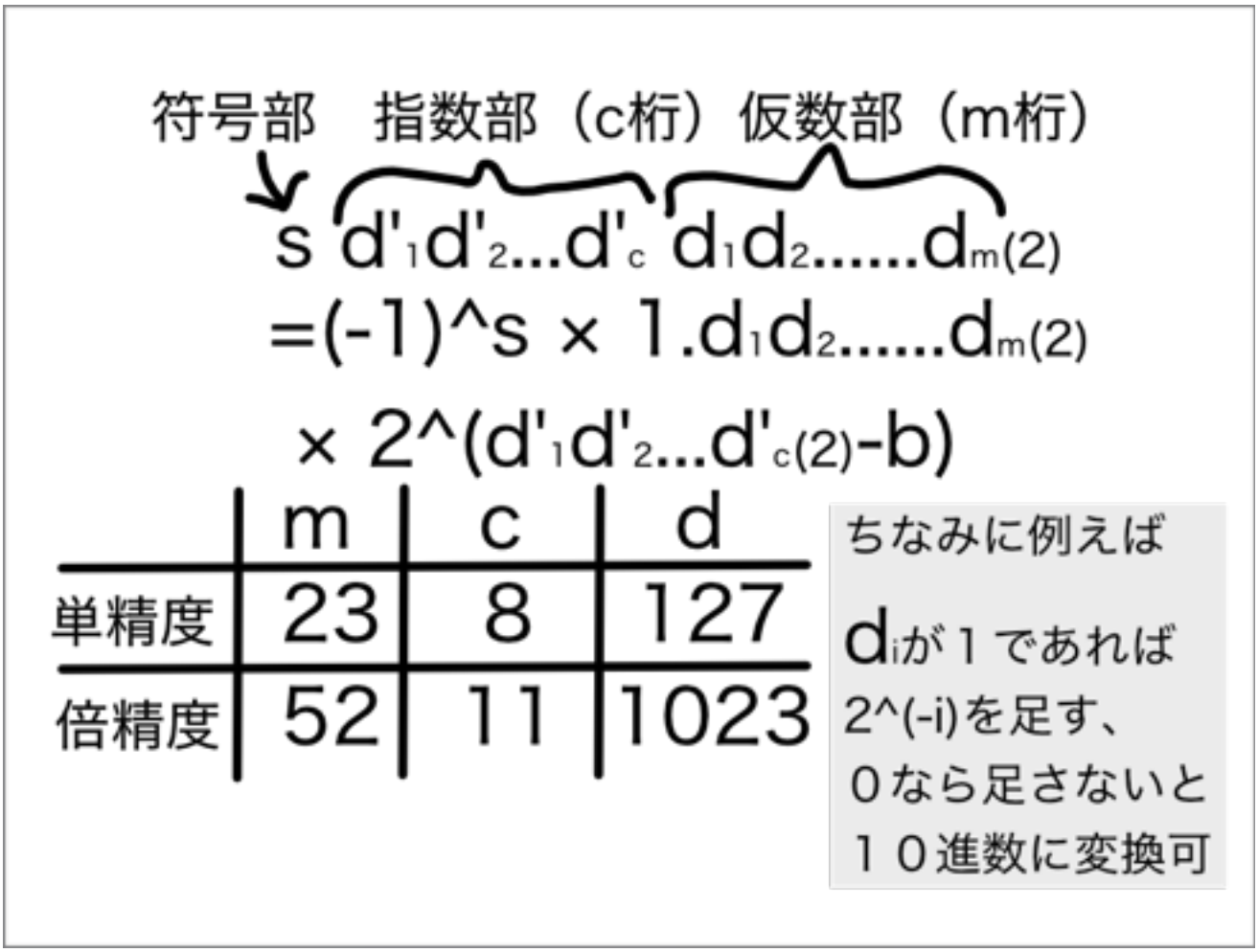
第三問です。誤差の問題です。ただの誤差の問題かと思ったらなんかちゃんと２進数に変換してどこで誤差がでるのかとか計算で議論しなきゃいけないくさいです。実際に誤差どうなるのか計算する問題まであるし…これはまじめに勉強するしかない（とりあえず第四問の解説後に誤差についてつらつら語ることにします。上記のことは嘘言ってるかもしれません。）

第四問は競技プログラミング（情報オリンピックとかそういうの）に出てくるような典型的な動的計画法の問題をプロセスをおって解いていく問題ですね（競技プログラミングではこれを全部自分で考えて実装します。この情報はどうでもいいか）３番までは支持通りに考えていけばとけると思います。たぶん。漸化式も考えればこのようになるでしょう。では四番の実装です。

```
def g()
  rcvr = make2d(N+1, M+1)
  for n in 0..N
    for m in 0..M
      if m==0 || n==0 || m > n
        rcvr[n][m] = 0
      else
        rcvr[n][m] = [rcvr[n-1][m], rcvr[n-1][m-1]+C[n-1][m-1]].max()
      end
    end
  end
  rcvr[N][M]
end
```

未だにこのRubyのmax関数の仕様がなれませんが…手書き紙面上のインデントって意味わからないですよね…まぁ色々言いたいところはありますが基本的には漸化式をそのまま実装するだけです。

さて、第三問の解答解説および誤差のお話に入りたいと思います。大前提として知っておくべきことにパソコンでの実数データ表現があります。これについてとりあえず画像にまとめてみました。（ページの都合上次のページに載せてあります。）ざっくりとしていますがsが0なら正、1なら負であるということdとd'が全部0の時に0を表すなどをおさえとけばよいでしょう。また実際の誤差を考える際の指標として単精度と倍精度の実数（浮動小数点数という）が10進数でどのくらいをカバーできるのかはおさえておいたほうが後々わかりやすいのでそれも表にまとめてみました。こちらも次のページに掲載しています。さてこれらをおさえた上で教科書に登場する４つの誤差をざっくり説明してみます。



	単精度浮動小数点数	倍精度浮動小数点数
10進数での有効桁数	log10(2^(23+1))≒7桁	log10(2^(52+1))≒16桁
最小の指数	2^(-126)≒1.2*10^(-38)	2^(-1022)
最大の指数	2^127≒1.7*10^38	2^1023
表せる最大値	2*1.7*10^38	2*2^1023

- 丸め誤差･･･有限桁の2進数では正確な値を表せない場合に生じる誤差
- 桁落ち誤差･･･非常に近い２つの値の差を求めた場合に有効桁数が減ることで引き起こる誤差
- 情報落ち誤差･･･大きな数に小さな数を足すと小さな数が大きな数の有効桁の範囲外となり無視されることで生じる誤差
- 打切り誤差･･･無限級数であらわされる値を途中で打ち切って近似することで生じる誤差

このようになっています。ではこれらを踏まえた上で具体的な問題に入っていきましょう。

　まず1番、これは誤差とは関係なく単純にコンピューターが浮動小数点型と明示しないかぎり数字はすべて整数型として扱うという性質の結果です。よって「扱っている数値が全て整数型であるため1/aが0と計算されたから。」のようなことを書いておけばよいでしょう。

　続く2番からが誤差の問題です。すべて一行程度と書かれているので上の4つのどの誤差が原因になっているかを調べればよいのだと思われます。そこでとりあえずまず重要なのがirbが通常浮動小数点数を単精度と倍精度のどちらで扱っているかということです。これは結果の値が16桁で

表されていることに注目すれば倍精度だということがわかります。それを踏まえて考えてみましょう。まずひとつ上の実行結果から1/a + 1という計算まではちゃんと計算できていることがわかります。よってこの時点で(1.001-1)*aという計算に同値であることがわかります。では1.001-1という計算はどうでしょうか？ここで桁落ち誤差の可能性が出てきます。仮にここで誤差が出ないとすると0.001と1が丸め誤差の起こらない数値ということはこれまでの計算でわかっているので矛盾します。ここまでわかれば“とりあえずの”解答として「途中1.001から1を引く計算があり、そこで桁落ち誤差が生まれるから。」と書けるでしょう。

3番、4番はこのような考察の上でさらに考えていくというものです。まず1/a**5ですがこれは10^(-3)^5=10^(-15)となり倍精度の有効桁の範囲内です。よって1+10^(-15)から1を引く操作のところで先ほどと同様桁落ち誤差が生じていると考えられます。a**5をかけるところで誤差が生じることはないでしょう。一方a**6は10^18です。すなわち1/a**6は10^(-18)となります。すると倍精度浮動小数点数としての1の有効桁数が小数点以下16桁までなのでこちらは情報落ち誤差によって1/a**6+1=1となります。よって0.0となるのは自明であり、4番の答えに関しては「cは桁落ち誤差で、dは情報落ち誤差により理論値からずれており情報落ち誤差においては1/a**6が無視されてしまうから。」などと書いておけばよいでしょう。

問題は3番です。おなじ桁落ち誤差ですが生まれてしまっている差をどう考えれば良いのでしょうか？こう考えてみるとなんだか2番が本当に桁落ち誤差だったのか怪しくなります。もしかして打ち切り誤差や丸め誤差かも？こういった不安を解消するためにも手計算で小数の10進数を2進数に変換する方法を知っておいたほうがよいかもしれません。http://www.it-license.com/cardinal_number/DecimalToBinary.htmlに図解付きで書かれているので見ておきましょう。実際少し手を動かしてみるとどうやら0.001が2進数では無限小数となりそうな数字であるということがわかります。これはサイトなどで計算させても確かめられますし2の累乗で下三桁が0になることがないのは明らかなのでそのことからもわかると思います。では丸め誤差なのでしょうか？これははいとは言い切れないと思います。そのことは1/a+1まではちゃんと計算できているからです。おそらくより正確なことを言えば丸め誤差、打切り誤差、桁落ち誤差の全てを含んだ誤差ということになるのでしょう。どれが一番影響が大きいかなどのことは正直教科書に書いてあることだけでは判断できないと思います。よって3番があることを踏まえてより無難な解答を作るとすれば2番が「２進数で0.001は無限小数になるため打切り誤差が生じたから。」で3番が「bと違いcでは桁落ち誤差の影響が大きく異なった誤差になるため。」とするのがよいのでしょう。もちろんここまでの判断を限られた時間内でこれだけの情報量でさせるのは無意味なところもあると思います。なので先ほど最初に書いた僕の解答のようなものを2番で書き、3番では誤差が値によって異なりうることを説明すれば正解だと思います。

さて5番6番にうつります。結論から言えばAが1.0、Bが0.0です。これは実際に実行してみても確かめられます。これらのようになる理由を答えるにあたっては今度のa=1024が2^10に等しいことを考えればよいでしょう。つまりこれらは途中計算もあわせて全ての結果が単純な2進数であらわすことができます。よってAでは誤差が生じず（二転三転してしまいますがこのことを考えるとbやcはもしかしたら桁落ち誤差というより丸め誤差としておいたほうが無難かもしれません。）、またBではdと同様情報落ち誤差が生じて同じ結果になったのだと考えられます。

ようやく誤差の範囲を抜けました（数値誤差はないものとしてよいの文字に安堵の表情）これからはお馴染みの手法で計算していけばよいでしょう。7番はコード通りに確かめていくだけでs=3となるので3.0です。8番はこれがモンテカルロ法であることを見ぬいてコードの意味を考えます。するとこれはx=[0.0,1.0),y=[0.0,1.0)となる正方形内の点を一つランダムに選んで、その点が正方形の四頂点から0.5以上離れているか、離れていたらsを増やすという操作を行っていることが見て取れます。つまり左の図においてエメラルドグリーンの部分だとsに3を足し、オレンジならば4を足していることになります。エメラルドグリーンの場所の面積がS1=0.25π、オレンジの場所の面積がS2=1-0.25πでありこのように文字を置くとnをどんどん増やしていくと4*S2+3*S1に近づいていきそうです。実際このプログラムにどれでもいいのでif文一個だけを残しそれ以外を無視してしまえば教科書や第九回の授業スライドにある1/4円の面積を求めるプログラムに酷似していることがわかります。すなわち好意的に解釈をすればこれは一辺が2の正方形から直径1の円を切り抜いて残った部分の面積を求めるプログラムということができてその答えは4-0.25πです。これは4*S2+3*S1とも一致します。よって8番はこの答えを書けばいいということになります。

以上になります。結局最後までreturnとか書いてきましたが、みなさんは授業通り返り値はreturn無しで素直に書くのがいいと思います。基本的にはプログラミングの最低限の文法を覚えといてあとは頭を使えばほとんどの問題を解くことができます。（実際競技プログラミングはこのようなアルゴリズムの知識を使って2014年第四問のような問題をどんどん解いていくというものですがみんながみんなプログラミングができるわけでなく、むしろプログラミング能力が高い人よりも数学強者など地頭のいい人のほうが強かったりします。）

がんばってくださいね。ちなみにすごい余談ですがRubyについてもっと知りたいって人はhttp://qiita.com/advent-calendar/2015/rubyとか見るといいと思います。