Karatsuba 空间复杂度优化

Karatsuba 的时间复杂度无法优化了：

$$T(n) = 3T(n/2) + \Theta(n) = \Theta(n^{\log_2 3})$$

但如果使用下面这样最朴素的写法，会分配大量的空间：

```
def Karatsuba(A, B): # n is the length of A and B
    separate A as AR, AR
    separate B as BR, BR
    ansL <- Karatsuba(AL, BL)
    ansR <- Karatsuba(AR, BR)
    ansS <- Karatsuba(AL+AR, BL+BR)
    ans  <- new array of length 2n-1
    ans[0:] += ansL
    ans[n/2:] += ansS-ansL-ansR
    ans[n:] += ansR
    return ans
```

其中 `AL+AR`，`BL+BR`，`ans <- new array of length 2n-1` 的操作都会分配 $\Theta(n)$ 的空间，导致算法的空间复杂度也是

$$S(n) = 3S(n/2) + \Theta(n) = \Theta(n^{\log_2 3})$$

实际上其中有大量的空间被浪费掉了！

优化 1

对于 `AL+AR`，`BL+BR` 因相加而创建的额外空间，比较好优化，只需要重复利用 `AL`，`BL` 的空间，进入子问题时加上 `AR, BR` ，退出子问题时再减掉 `AR`，`BR` 就行了。

```
def Karatsuba(A, B): # n is the length of A and B
    separate A as AR, AR
    separate B as BR, BR
    ansL <- Karatsuba(AL, BL)
    ansR <- Karatsuba(AR, BR)
    AL += AR
    BL += BR
    ansS <- Karatsuba(AL, BL)
    AL -= AR
    BL -= BR
    ans <- new array of length 2n-1
    ans[0:] += ansL
    ans[n/2:] += ansS-ansL-ansR
    ans[n:] += ansR
    return ans
```

优化 2

另一个显著浪费空间的地方在于，`ansL`, `ansR`, `ansS` 这三个数组在计算完 `ans` 后 `return ans` 之前就已经没用了。如何把这块空间利用起来呢？

我们需要换一种空间管理模式，不再是用一次丢一次的堆模式，而是栈模式，在解决完三个子问题、计算完 `ans` 之后，就把 `ansL`, `ansR`, `ansS` 三个数组从栈空间弹出，再把 `ans` 往栈里压。

需要定义一个全局数组 `tmp` ，用来临时保存一下 `ans` ，等 `ansL`, `ansR`, `ansS` 三个数组从栈空间弹出之后，再把 `tmp` 往栈里压。

```
tmp <- global array of length 2n-1
def Karatsuba(A, B): # n is the length of A and B
    separate A as AR, AR
    separate B as BR, BR
    ansL <- Karatsuba(AL, BL)
    ansR <- Karatsuba(AR, BR)
    AL += AR
    BL += BR
    ansS <- Karatsuba(AL, BL)
    AL -= AR
    BL -= BR
    tmp.clear()
    tmp[0:] += ansL
    tmp[n/2:] += ansS-ansL-ansR
    tmp[n:] += ansR
    stack.pop(ansS)
    stack.pop(ansR)
    stack.pop(ansL)
    ans <- stack.push(tmp)
    return ans
```

如何分析这样做最多占用多少空间？设 $S(n)$ 为解决一个大小为 n 的子问题，额外占用栈空间大小的峰值。

我们分析这个函数运行到下面几行的时候，额外占用栈空间大小的变化情况（为方便假设 n 为 2 的幂）：

- 初始时是 0 ；
  - 运行 `ansL <- Karatsuba(AL, BL)` 的过程中，额外占用栈空间大小的峰值是 $S(\frac{n}{2})$ ；
- 运行完这行之后，栈空间相比初始时额外多了 $n-1$ ，因为多存了个 `ansL` ；
  - 运行 `ansR <- Karatsuba(AR, BR)` 的过程中，额外占用栈空间大小的峰值是 $n-1+S(\frac{n}{2})$ ；
- 运行完这行之后，栈空间相比初始时额外多了 $2n-2$ ，因为多存了个 `ansR` ；
  - 运行 `ansS <- Karatsuba(AL, BL)` 的过程中，额外占用栈空间大小的峰值是 $2n-2+S(\frac{n}{2})$ ；
- 运行完这行之后，栈空间相比初始时额外多了 $3n-3$ ，因为多存了个 `ansS` ；
- 依次pop掉 `ansS`, `ansR`, `ansL` ，再 push `ans` 的之后，栈空间相比初始时额外多了 $2n-1$ ，反而没有之前多了。

那什么时候额外占用栈空间大小达到峰值呢？显然最大值 $S(n)$ 只能在 $2n-2+S(n/2)$ 和 $3n-3$ 中取，因为别的时候至少都比这俩之一小。展开计算，不难发现取前者更大，但仍然不超过 $4n$ ，

$$S(n)=2n-2+S(n/2) \implies S(n)=\Theta(n).$$

另外，全局数组 `tmp` 也需要 $\Theta(n)$ 。相比之下， $\Theta(\log n)$ z的函数递归栈空间就微不足道了。

最终我们得到了空间复杂度 $\Theta(n)$ 的做法，是一个渐进意义下的巨大优化。

具体实现可以参考代码。