6

May 19, 2016

1 Multiplying Large Numbers

Given 2 n-bit numbers A and B, compute AB. Trivially there's a lower bound of $\Omega(N)$. Trivial solution is $O(N^2)$.

Alternatively, use Karatsuba-Ofman's Algorithm: see https://en.wikipedia.org/wiki/Karatsuba_algorithm

If $x = x_1 B^m + x_2, y = y_1 B^m + y_2$, then we exploit the fact that $z = x_1 y_2 + x_2 y_1 = (x_1 + y_1)(x_2 + y_2) - x_1 y_1 - x_2 y_2$ and $xy = (x_1 B^m + x_2)(y_1 B^m + y_2) = x_1 y_1 B^{2m} + z B^m + x_2 y_2$.

```
In [134]: # O(N ^ 2)
          # a,b are lists of {0, 1}
          def multiply_1(a, b):
              assert len(a) <= len(b)
              if len(a) <= 0:
                  return []
              elif len(a) == 1:
                  if a[0] == 0:
                      return [0 for i in b]
                  else:
                      return b[:]
              right = multiply(a[:len(a)/2], b) + [0 for _ in range(len(a)/2, len(a
              left = [0 for _ in range(len(a)/2)] + multiply(a[len(a)/2:], b)
              s = [l+r for l,r in zip(left, right)] #  contains 2's and possibly 3

              for i in reversed(range(len(s))):
                  if s[i] == 2 or s[i] == 3:
                      s[i] = 0 if s[i] == 2 else 1
                      if i == 0:
                          s.insert(0, 1)
                      else:
                          s[i-1] += 1
              return s[s.index(1):]  # filters out 0-padding

          # Karatsuba
          # Analysis: T(n) = 3 T(n/2) + Theta(n)
          # via master theorem: O(n ^ log_2(3))
          def multiply_2(a, b):
              assert len(a) == len(b)
```

```python
n = len(a)
if len(a) <= 3:
    # TODO: do stuff
    pass
A2 = a[:n/2], A1 = a[n/2:]
A2 = a[:n/2], A1 = a[n/2:]
C1 = mult(A1, B1)
C2 = mult(A2, B2)
C3 = mult(add(A1, A2), add(B1, B2))
return add(add(shift(C1, n), shift(sub(sub(C3, C1), C2), n/2)), C2)
```

```python
In [135]: assert multiply_1([1, 1, 0], [1, 1, 1]) == [1, 0, 1, 0, 1, 0]
          assert multiply_1([1, 0], [0, 1]) == [1, 0]
```

Refinement (3-way divide): $T(n) = 5T(n/3) + \Theta(n)$
Schonhage-Strassen '71: $O(n \log n \log \log n)$
Furer '07: $O(n \log n \log^* n)$
Open question: can multiplication be done in $O(n \log n)$?

---

2 Matrix Multiplication

Given $n \times n$ matrices $A, B \in \mathbb{R}^{n \times n}$, compute $n \times n$ matrix $C = AB$.

Trivial solution is $O(n^3)$.

Strassen's Algorithm: We note that we can subdivide each matrix into 4 quadrants, resulting in only needing $8T(n/2) + \Theta(n^2) = O(n^3)$. But we can use matrix substructure to only need 7 multiplications.