**Algorithm 33** Selecao

```
1: procedure SELECAO(vetor S[], int k)
2:     if |S| < 7 then
3:         Ordene S
4:         return S[k]
5:     else
6:         Dividir S em n/5 subsequências
7:         Ordene cada sub sequência e determine sua mediana
8:         Seja M a sub sequência das medianas
9:         m = selecao(M, M/2)
10:        S_1 = {x ∈ S | x < m}
11:        S_2 = {x ∈ S | x = m}
12:        S_2 = {x ∈ S | x > m}
13:        if |S_1| ≥ k then
14:            selecao(S_1, k)
15:        else
16:            if |S_1| + |S_2| ≥ k then
17:                return m
18:            else
19:                return selecao(S_3, k − |S_1| − |S_2|)
```

$$n) = \begin{cases} 1, \text{se } n = 1 \\ 3T(\lfloor n/2 \rfloor) + n, \text{cc} \end{cases}$$

es consomem muito menos
tempo que multiplicação

$$\Theta(n^{\log 3}) = \Theta(n^{1,58})$$

**Algorithm 34** Karatsuba-Ofman

```
1:  procedure KARATSUBA-OFMAN(u, v, n)
2:      if n ≤ 3 then
3:          return uv
4:      else
5:          k ← ⌈n/2⌉
6:          p ← ⌊u/10^k⌋
7:          q ← u mod 10^k
8:          r ← ⌊v/10^k⌋
9:          s ← v mod 10^k
10:         pr ← KARATSUBA-OFMAN(p, r, k)
11:         qs ← KARATSUBA-OFMAN(q, s, k)
12:         y ← KARATSUBA-OFMAN(p + q, r + s, k + 1)
13:         x ← pq10^{2k} + (y − pr − qs)10^k + qs)
14:     return x
```

**Algorithm 35** Pontos mais próximos

```
1:  procedure POINTS-DC(P, n)
2:      if n ≤ lim then
3:          Resolva por força bruta
4:      Ordene os pontos pela coordenada x
5:      Compute a linha ℓ que separa os pontos ao meio, particio-
        nando P em S_1 e S_2
6:      δ_1 ← Points-DC(S_1, |S_1|)
7:      δ_2 ← Points-DC(S_2, |S_2|)
8:      δ ← min(δ_1, δ_2)
9:      Ordene pela coordenada y os pontos na faixa F = {p ∈
        P | p.x ∈ [ℓ − δ, ℓ + δ]}
10:     Seja m ← |F|
11:     for i ← 1 → m do
12:         k ← 1
13:         while i + k ≤ m e p_{i+k}.y < p_i.y + δ  do
14:             δ ← min(δ, dist(p_i, p_{i+k}))
15:             k ← k + 1
16:     return δ
```

**Algorithm 41** Solução Haste PD

```
1: procedure HASTE-PD(vetor p[], inteiro n)
2:     for i ← 1 → n do
3:         r[i] ← −∞
4:     r[0] ← 0
5:     for i ← 1 → n do
6:         for j ← 1 → i do
7:             if r[i] < p[j] + r[i − j] then
8:                 r[i] ← p[j] + r[i − j]
9:                 s[i] ← j                    ▷ Reconstrução da solução
10:    return r[n] e s
```

**Algorithm 42** Programação Dinâmica de Multiplicação de Matrizes

```
1:  procedure ProgDinMatr(vetor P[])
2:      n ← P.size() − 1
3:      for i = 1 → n do
4:          m[i, j] = 0
5:      for l = 2 → n do
6:          for i = 1 → n − l + 1 do
7:              j = i + l − 1
8:              m[i, j] = inf
9:              for k = i → j do
10:                 g = m[i, k] + m[k + 1, j] + p[i − 1].p[k].p[j]
11:                 if g < m[i, j] then
12:                     m[i, j] = g
13:                     s[i, j] = k
```