Análise de Algoritmos Apresentação #1

Lucas Borges Fernandes
Eduardo Costa de Paiva
Gustavo Silveira Marques

Menor distância entre dois pontos: Abordagem **Divide and Conquer**

Pseudocódigo - Força Bruta $\Theta(n^2)$

PARMAISPROXIMO(X, Y, n)

```
d ← +∞
para i ← 1 até n faça
    para j ← i+1 até n faça
        se DIST(X[i], Y[i], X[j], Y[j]) < d
            então d ← DIST(X[i], Y[i], X[j], Y[j])
devolva d
```

Pseudocódigo - Divide and Conquer Θ(n logn)

```
PARMAISPROXIMO_DC(XOrd, YOrd,  esquerda, direita)
    1 se |direita - esquerda| = 2
    2       então retorna (YOrd[esquerda], YOrd[direita-1], DIST(YOrd[esquerda], YOrd[direita - 1]))
    3 se |direita - esquerda| = 1
    4       então retorna (YOrd[esquerda], YOrd[esquerda], +∞)
    5  meio = (esquerda + direita) / 2
    6 (pontosEsq,distanciaEsq) = PARMAISPROXIMO_DC(XOrd, YOrd, esquerda, meio)
    7 (pontosDir,distanciaDir) = PARMAISPROXIMO_DC(XOrd, YOrd, meio, direita)
```

Pseudocódigo - Divide and Conquer Θ(n logn)

```
minDist = min(distanciaEsq, distanciaDir)
minPontos =  se (minDist = distanciaEsq) pontosEsq senao pontosDir
pontosMeioList = RETORNALISTMEIO(YOrd, esquerda, direita, XOrd[meio]], minDist)
(pontosMeio, minMeio) = RETORNAMINPONTOSMEIO(pontosMeioList, minDist)
minPontos = se (min(minDist, minMeio) == minDist) minPontos senao pontosMeio
retorna (minPontos, min(minDist, minMeio))
```

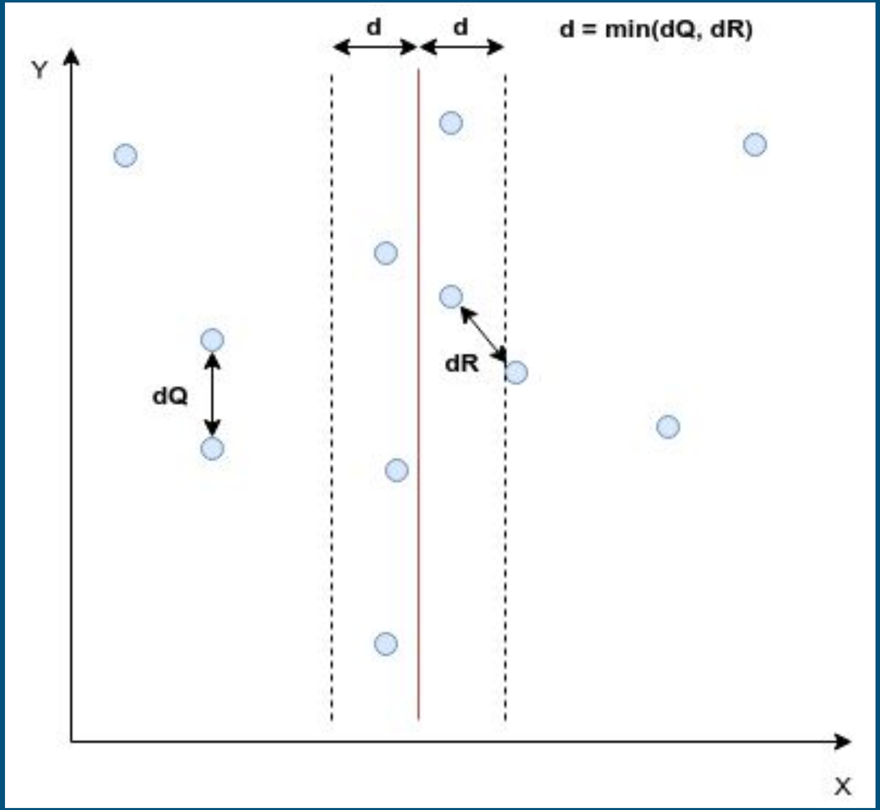
Pseudocódigo - Divide and Conquer $\Theta(n \log n)$

```
RETORNALISTMEIO(YOrd, esquerda, direita, pontoMeio, minDist)
    para i ← esquerda até direita - 1 faça
        se |YOrd[i].x - pontoMeio.x| < minDist faça
            listaPontosMeio.append(YOrd[i])
    retorna listaPontosMeio
```

Pseudocódigo - Divide and Conquer Θ(n logn)

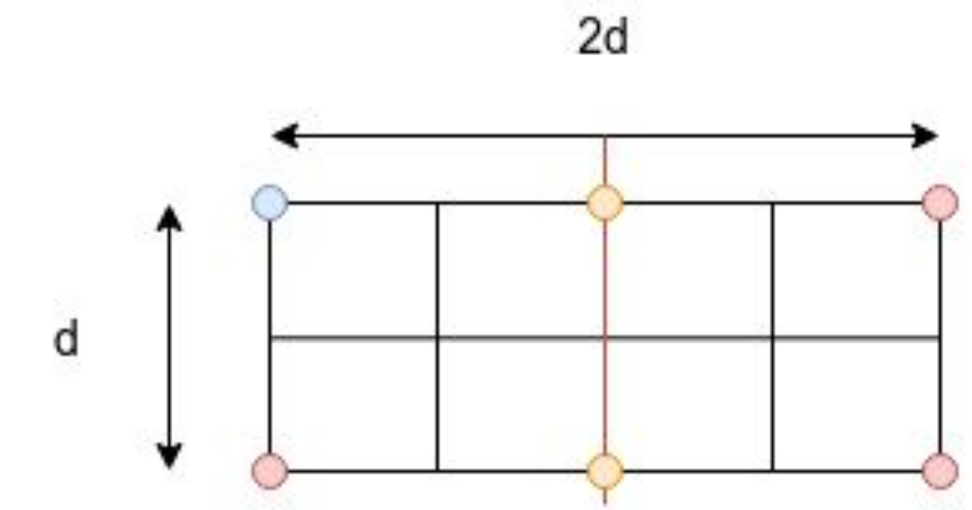
```
RETORNAMINPONTOSMEIO(listaPontosMeio, minDist)
    para i ← 1 até n faça
        para j ← i+1 até n faça
            se (listaPontosMeio[j].y - listaPontosMeio[i].y) >= minDist
                break;
            senao se DIST(listaPontosMeio[i], listaPontosMeio[j]) < minDist
                minPontos = (listaPontosMeio[i], listaPontosMeio[j])
                minDist = DIST(listaPontosMeio[i], listaPontosMeio[j])
    retorna (minPontos, minDist)
```

Recorrência - Ideia



Recorrência - Ideia

- **Comparamos no máximo 7 pontos, isto é, o tempo de combinar será O(7n)**



Recorrência

Desta forma, o tempo de combinar é $O(7n)$ e o tempo de dividir é $\Theta(1)$.

Então, $f(n) = \Theta(1) + O(7n)$

Podemos considerar que o tempo de combinar é $\Theta(n)$, pois os limites superior e inferior da função de complexidade estão na ordem de *n*.

Em cada passo geramos dois subproblemas, *que serão resolvidos separadamente*, de tamanho *n/2* cada.

Sendo assim, a recorrência é expressa por:

$$T(n) = 2 * T(n / 2) + \Theta(n)$$

Análise de complexidade

Sendo a recorrência expressa por:

$T(n) = 2 * T(n / 2) + \Theta(n)$

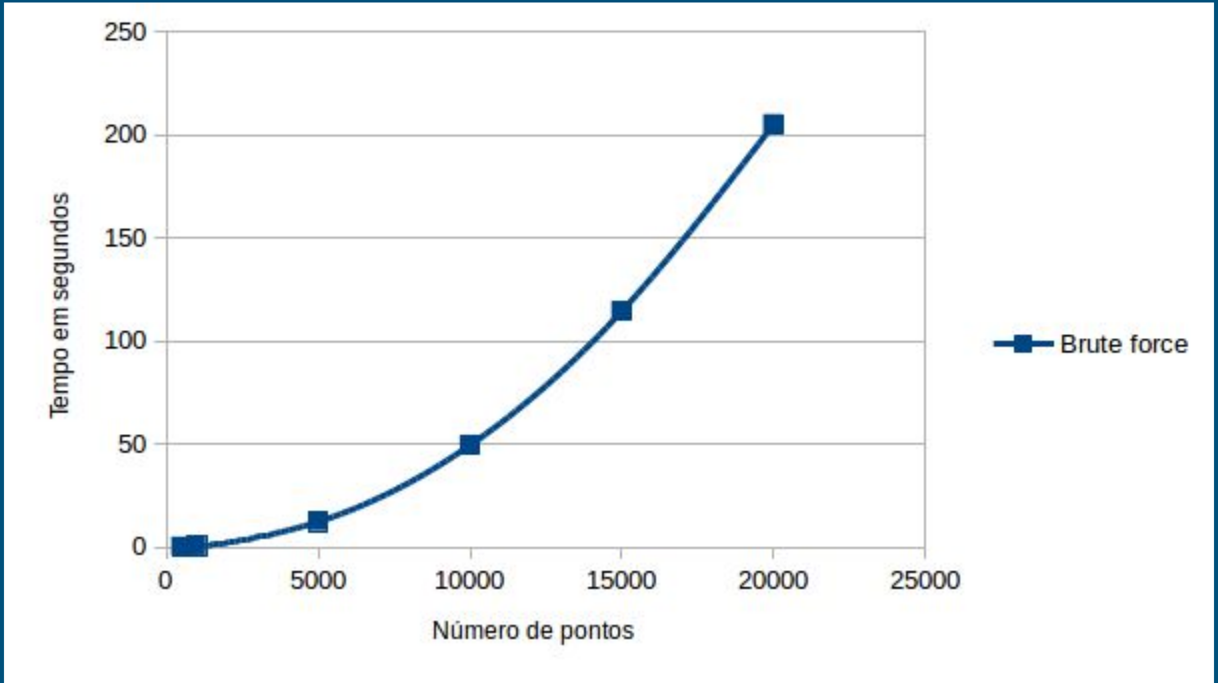
Podemos analisar a complexidade de tempo do algoritmo utilizando o Teorema Master.

Sejam a = 2 e b = 2. Então, $n^{\log_b a} = n^{\log_2 2} = n$

Desta forma, caímos no segundo caso do Teorema Master, uma vez que $f(n) = \Theta(n^{\log_b a})$

e concluímos, portanto, que a complexidade do algoritmo é $\Theta(n * \log n)$

Curvas de tempo de execução - Força bruta



Curvas de tempo de execução - Divide n Conquer