Ordenação recursiva

```
1  void merge_sort (int v[], int inicio, int fim, int aux[]) {
2      int n = fim - inicio;
3      if (n < 2) return;
4      else {
5          merge_sort(v, inicio, inicio + n/2, aux);
6          merge_sort(v, inicio + n/2, fim, aux);
7          merge(v, inicio, fim, aux);
8      }
9  }
```

Merge sort

Algoritmo 1 mergeSort (vetor v, índices $\langle inicio, fim \rangle$)

n $\leftarrow$ fim - inicio

Merge sort

Algoritmo 1 mergeSort (vetor v, índices $\langle inicio, fim \rangle$)

```
n ← fim - inicio
if n < 2 then
    return
```

Merge sort

Algoritmo 1 mergeSort (vetor v, índices $\langle inicio, fim \rangle$)

```
n ← fim - inicio
if n < 2 then
    return
else
    mergeSort(v, 0, inicio + ⌊n/2⌋)
    mergeSort(v, inicio + ⌊n/2⌋, n)
```

Merge sort

Algoritmo 1 mergeSort (vetor v, índices $\langle inicio, fim \rangle$)

```
n ← fim - inicio
if n < 2 then
    return
else
    mergeSort(v, 0, inicio + ⌊n/2⌋)
    mergeSort(v, inicio + ⌊n/2⌋, n)
    merge(v, inicio, fim)
end if
```

Merge sort

$$T(n) = \begin{cases} 3, & n < 2 \\ 2 + 2T(\frac{n}{2}) + t_{merge}, & n \geq 2 \end{cases} \tag{1}$$

```
1  void merge (int v[], int inicio, int fim, int aux[]) {
2      int meio = (inicio + fim) / 2;
3      int i = inicio, j = meio, k = inicio;
4      while (i < meio && j < fim)
5          aux[k++] = v[i] < v[j] ? v[i++] : v[j++];
6      while (i < meio || j < fim)
7          aux[k++] = i < meio ? v[i++] : v[j++];
8      i = inicio;
9      while (i < fim) v[i++] = aux[i++];
10 }
```

Algoritmo 2 merge(vetor v, índices $\langle inicio, fim \rangle$)

meio $\leftarrow (inicio + fim)/2$

Algoritmo 2 merge(vetor v, índices $\langle inicio, fim \rangle$)

meio $\leftarrow (inicio + fim)/2$

i $\leftarrow$ inicio, j $\leftarrow$ meio, k $\leftarrow$ inicio

Algoritmo 2 merge(vetor v, índices $\langle inicio, fim \rangle$)

meio $\leftarrow (inicio + fim)/2$

i $\leftarrow$ inicio, j $\leftarrow$ meio, k $\leftarrow$ inicio

while i $<$ meio **and** j $<$ fim **do**

end while

Algoritmo 2 merge(vetor v, índices $\langle inicio, fim \rangle$)

meio $\leftarrow (inicio + fim)/2$

i $\leftarrow$ inicio, j $\leftarrow$ meio, k $\leftarrow$ inicio

while i $<$ meio **and** j $<$ fim **do**

    if v[i] $<$ v[k] **then**

        aux[k] $\leftarrow$ v[i]; i $\leftarrow$ i $+$ 1

end while

Algoritmo 2 merge(vetor v, índices $\langle inicio, fim \rangle$)

```
meio ← (inicio + fim)/2
i ← inicio, j ← meio, k ← inicio
while i < meio and j < fim do
    if v[i] < v[k] then
        aux[k] ← v[i]; i ← i + 1
    else
        aux[k] ← v[j]; j ← j + 1
    end if

end while
```

Algoritmo 2 merge(vetor v, índices $\langle inicio, fim \rangle$)

```
meio ← (inicio + fim)/2
i ← inicio, j ← meio, k ← inicio
while i < meio and j < fim do
    if v[i] < v[k] then
        aux[k] ← v[i]; i ← i + 1
    else
        aux[k] ← v[j]; j ← j + 1
    end if
    k ← k + 1
end while
```

Algoritmo 2 merge(vetor v, índices $\langle inicio, fim \rangle$)

```
meio ← (inicio + fim)/2
i ← inicio, j ← meio, k ← inicio
while i < meio and j < fim do
    if v[i] < v[k] then
        aux[k] ← v[i]; i ← i + 1
    else
        aux[k] ← v[j]; j ← j + 1
    end if
    k ← k + 1
end while
while i < meio or j < fim do




end while
```

Algoritmo 2 merge(vetor v, índices $\langle inicio, fim \rangle$)

```
meio ← (inicio + fim)/2
i ← inicio, j ← meio, k ← inicio
while i < meio and j < fim do
    if v[i] < v[k] then
        aux[k] ← v[i]; i ← i + 1
    else
        aux[k] ← v[j]; j ← j + 1
    end if
    k ← k + 1
end while
while i < meio or j < fim do
    if i < meio then
        aux[k] ← v[i]; i ← i + 1









end while
```

Algoritmo 2 merge(vetor v, índices $\langle inicio, fim \rangle$)

```
meio ← (inicio + fim)/2
i ← inicio, j ← meio, k ← inicio
while i < meio and j < fim do
    if v[i] < v[k] then
        aux[k] ← v[i]; i ← i + 1
    else
        aux[k] ← v[j]; j ← j + 1
    end if
    k ← k + 1
end while
while i < meio or j < fim do
    if i < meio then
        aux[k] ← v[i]; i ← i + 1
    else
        aux[k] ← v[j]; j ← j + 1
    end if

end while
```

Algoritmo 2 merge(vetor v, índices $\langle inicio, fim \rangle$)

```
meio ← (inicio + fim)/2
i ← inicio, j ← meio, k ← inicio
while i < meio and j < fim do
    if v[i] < v[k] then
        aux[k] ← v[i]; i ← i + 1
    else
        aux[k] ← v[j]; j ← j + 1
    end if
    k ← k + 1
end while
while i < meio or j < fim do
    if i < meio then
        aux[k] ← v[i]; i ← i + 1
    else
        aux[k] ← v[j]; j ← j + 1
    end if
    k ← k + 1
end while
```

Algoritmo 2 merge(vetor v, índices $\langle inicio, fim \rangle$)

```
meio ← (inicio + fim)/2
i ← inicio, j ← meio, k ← inicio
while i < meio and j < fim do
    if v[i] < v[k] then
        aux[k] ← v[i]; i ← i + 1
    else
        aux[k] ← v[j]; j ← j + 1
    end if
    k ← k + 1
end while
while i < meio or j < fim do
    if i < meio then
        aux[k] ← v[i]; i ← i + 1
    else
        aux[k] ← v[j]; j ← j + 1
    end if
    k ← k + 1
end while
i ← inicio; k ← 0
```

Algoritmo 2 merge(vetor v, índices $\langle inicio, fim \rangle$)

```
meio ← (inicio + fim)/2
i ← inicio, j ← meio, k ← inicio
while i < meio and j < fim do
    if v[i] < v[k] then
        aux[k] ← v[i]; i ← i + 1
    else
        aux[k] ← v[j]; j ← j + 1
    end if
    k ← k + 1
end while
while i < meio or j < fim do
    if i < meio then
        aux[k] ← v[i]; i ← i + 1
    else
        aux[k] ← v[j]; j ← j + 1
    end if
    k ← k + 1
end while
i ← inicio; k ← 0
while i < fim do

end while
```

Algoritmo 2 merge(vetor v, índices $\langle inicio, fim \rangle$)

```
meio ← (inicio + fim)/2
i ← inicio, j ← meio, k ← inicio
while i < meio and j < fim do
    if v[i] < v[k] then
        aux[k] ← v[i]; i ← i + 1
    else
        aux[k] ← v[j]; j ← j + 1
    end if
    k ← k + 1
end while
while i < meio or j < fim do
    if i < meio then
        aux[k] ← v[i]; i ← i + 1
    else
        aux[k] ← v[j]; j ← j + 1
    end if
    k ← k + 1
end while
i ← inicio; k ← 0
while i < fim do
    v[i] ← aux[k]; i ← i + 1; k ← k + 1
end while
```

Procedimento merge

- Melhor e pior casos:
  - Laços de cima: $O(n)$

Procedimento merge

- Melhor e pior casos:
  - Laços de cima: $O(n)$
  - Laço de baixo: $O(n)$

Procedimento merge

- Melhor e pior casos:
  - Laços de cima: $O(n)$
  - Laço de baixo: $O(n)$
  - Demais operações: $O(1)$

Procedimento merge

- Melhor e pior casos:
  - Laços de cima: $O(n)$
  - Laço de baixo: $O(n)$
  - Demais operações: $O(1)$
- $t_{\text{merge}}(n) \in O(n)$

Merge sort

Algoritmo 1 mergeSort (vetor v, índices $\langle inicio, fim \rangle$, vetor aux)

```
n ← fim - inicio
if n < 2 then
    return
else
    mergeSort(v, 0, inicio + ⌊n/2⌋, aux)
    mergeSort(v, inicio + ⌊n/2⌋, n, aux)
    merge(v, inicio, fim, aux)
end if
```

Merge sort

$$T(n) = \begin{cases} 3, & n < 2 \\ 2 + 2T(\frac{n}{2}) + t_{merge}, & n \geq 2 \end{cases} \tag{2}$$

Merge sort

$$T(n) = \begin{cases} 3, & n < 2 \\ 2 + 2T(\frac{n}{2}) + n, & n \geq 2 \end{cases} \tag{3}$$

Merge sort

$$T(n) = 2T(\tfrac{n}{2}) + n$$

Merge sort

$$T(n) = 2T(\tfrac{n}{2}) + n$$

$$T(n) = 2(2T(\tfrac{n}{4}) + \tfrac{n}{2}) + n = 4T(\tfrac{n}{4}) + 2n$$

Merge sort

$$T(n) = 2T(\tfrac{n}{2}) + n$$

$$T(n) = 2(2T(\tfrac{n}{4}) + \tfrac{n}{2}) + n = 4T(\tfrac{n}{4}) + 2n$$

$$T(n) = 4(2T(\tfrac{n}{8}) + \tfrac{n}{4}) + 2n = 8T(\tfrac{n}{8}) + 3n$$

Merge sort

$$T(n) = 2T(\tfrac{n}{2}) + n$$

$$T(n) = 2(2T(\tfrac{n}{4}) + \tfrac{n}{2}) + n = 4T(\tfrac{n}{4}) + 2n$$

$$T(n) = 4(2T(\tfrac{n}{8}) + \tfrac{n}{4}) + 2n = 8T(\tfrac{n}{8}) + 3n$$

$$T(n) = 2^k T(\tfrac{n}{2^k}) + k \cdot n$$

Merge sort

$$T(n) = 2T(\tfrac{n}{2}) + n$$

$$T(n) = 2(2T(\tfrac{n}{4}) + \tfrac{n}{2}) + n = 4T(\tfrac{n}{4}) + 2n$$

$$T(n) = 4(2T(\tfrac{n}{8}) + \tfrac{n}{4}) + 2n = 8T(\tfrac{n}{8}) + 3n$$

$$T(n) = 2^k T(\tfrac{n}{2^k}) + k \cdot n$$

- Fazendo $k = log_2\, n$:

Merge sort

$$T(n) = 2T(\tfrac{n}{2}) + n$$

$$T(n) = 2(2T(\tfrac{n}{4}) + \tfrac{n}{2}) + n = 4T(\tfrac{n}{4}) + 2n$$

$$T(n) = 4(2T(\tfrac{n}{8}) + \tfrac{n}{4}) + 2n = 8T(\tfrac{n}{8}) + 3n$$

$$T(n) = 2^k T(\tfrac{n}{2^k}) + k \cdot n$$

- Fazendo $k = log_2\, n$:

$$T(n) = 2^{log_2\, n} T(\tfrac{n}{2^{log_2\, n}}) + n \cdot log_2\, n$$

Merge sort

$$T(n) = 2T(\tfrac{n}{2}) + n$$

$$T(n) = 2(2T(\tfrac{n}{4}) + \tfrac{n}{2}) + n = 4T(\tfrac{n}{4}) + 2n$$

$$T(n) = 4(2T(\tfrac{n}{8}) + \tfrac{n}{4}) + 2n = 8T(\tfrac{n}{8}) + 3n$$

$$T(n) = 2^k T(\tfrac{n}{2^k}) + k \cdot n$$

- Fazendo $k = log_2\, n$:

$$T(n) = 2^{log_2\, n} T(\tfrac{n}{2^{log_2\, n}}) + n \cdot log_2\, n$$

$$T(n) = nT(\tfrac{n}{n}) + n \cdot log_2\, n = nT(1) + n \cdot log_2\, n$$

Merge sort

$$T(n) = \begin{cases} 3, & n < 2 \\ 2T(\frac{n}{2}) + n, & n \geq 2 \end{cases} \tag{4}$$

Merge sort

$$T(n) = 2^k T(\frac{n}{2^k}) + k \cdot n$$

- Fazendo $k = log_2\, n$:

$$T(n) = 2^{log_2\, n} T(\frac{n}{2^{log_2\, n}}) + n \cdot log_2\, n$$

$$T(n) = nT(\frac{n}{n}) + n \cdot log_2\, n = nT(1) + n \cdot log_2\, n$$

$$T(n) = 3n + n\, log_2\, n \longrightarrow T(n) \in \Theta(n\, log\, n)$$

Merge sort

Algoritmo 2 mergeSort (vetor v, índices $\langle inicio, fim \rangle$)

n $\leftarrow$ inicio - fim
if n < 2 **then**
  return
else
  mergeSort(v, inicio, inicio + $\lfloor n/4 \rfloor$)
  mergeSort(v, inicio + $\lfloor n/4 \rfloor$, inicio + $\lfloor n/2 \rfloor$)
  mergeSort(v, inicio + $\lfloor n/2 \rfloor$, inicio + $\lfloor 3n/4 \rfloor$)
  mergeSort(v, inicio + $\lfloor 3n/4 \rfloor$, fim)
  merge(v, inicio, fim)
end if

Merge sort

$$T(n) = 4T(\tfrac{n}{4}) + n$$

Merge sort

$$T(n) = 4\,T(\tfrac{n}{4}) + n$$

$$T(n) = 4(4\,T(\tfrac{n}{16}) + \tfrac{n}{4}) + n = 16\,T(\tfrac{n}{16}) + 2n$$

Merge sort

$$T(n) = 4T(\tfrac{n}{4}) + n$$

$$T(n) = 4(4T(\tfrac{n}{16}) + \tfrac{n}{4}) + n = 16T(\tfrac{n}{16}) + 2n$$

$$T(n) = 16(4T(\tfrac{n}{64}) + \tfrac{n}{16}) + 2n = 64T(\tfrac{n}{64}) + 3n$$

Merge sort

$$T(n) = 4\,T(\tfrac{n}{4}) + n$$

$$T(n) = 4(4\,T(\tfrac{n}{16}) + \tfrac{n}{4}) + n = 16\,T(\tfrac{n}{16}) + 2n$$

$$T(n) = 16(4\,T(\tfrac{n}{64}) + \tfrac{n}{16}) + 2n = 64\,T(\tfrac{n}{64}) + 3n$$

$$T(n) = 4^k\,T(\tfrac{n}{4^k}) + k \cdot n$$

Merge sort

$$T(n) = 4^k T(\tfrac{n}{4^k}) + k \cdot n$$

Merge sort

$$T(n) = 4^k T(\frac{n}{4^k}) + k \cdot n$$

- Fazendo $k = log_4\, n$:

Merge sort

$$T(n) = 4^k T(\frac{n}{4^k}) + k \cdot n$$

- Fazendo $k = log_4\ n$:

$$T(n) = 4^{log_4\ n} T(\frac{n}{4^{log_4\ n}}) + n \cdot log_4\ n$$

Merge sort

$$T(n) = 4^k T(\frac{n}{4^k}) + k \cdot n$$

- Fazendo $k = log_4\ n$:

$$T(n) = 4^{log_4\ n} T(\frac{n}{4^{log_4\ n}}) + n \cdot log_4\ n$$

$$T(n) = nT(\frac{n}{n}) + n \cdot log_4\ n = nT(1) + n \cdot log_4\ n$$

Merge sort

$$T(n) = 4^k T(\frac{n}{4^k}) + k \cdot n$$

- Fazendo $k = log_4\ n$:

$$T(n) = 4^{log_4\ n} T(\frac{n}{4^{log_4\ n}}) + n \cdot log_4\ n$$

$$T(n) = nT(\frac{n}{n}) + n \cdot log_4\ n = nT(1) + n \cdot log_4\ n$$

$$T(n) = 3n + n\ log_4\ n$$

Merge sort

$$T(n) = 2T(\tfrac{n}{2}) + n \longrightarrow T(n) = 3n + n\,log_2\,n$$

Merge sort

$$T(n) = 2T(\tfrac{n}{2}) + n \longrightarrow T(n) = 3n + n\,log_2\,n$$

$$T(n) = 4T(\tfrac{n}{4}) + n \longrightarrow T(n) = 3n + n\,log_4\,n$$

Merge sort

$$T(n) = 2T(\tfrac{n}{2}) + n \longrightarrow T(n) = 3n + n\,log_2\,n$$

$$T(n) = 4T(\tfrac{n}{4}) + n \longrightarrow T(n) = 3n + n\,log_4\,n$$

$$T(n) = 8T(\tfrac{n}{8}) + n \longrightarrow T(n) = 3n + n\,log_8\,n$$

Merge sort

$$T(n) = 2T(\tfrac{n}{2}) + n \longrightarrow T(n) = 3n + n\,log_2\,n$$

$$T(n) = 4T(\tfrac{n}{4}) + n \longrightarrow T(n) = 3n + n\,log_4\,n$$

$$T(n) = 8T(\tfrac{n}{8}) + n \longrightarrow T(n) = 3n + n\,log_8\,n$$

$$T(n) = nT(\tfrac{n}{n}) + n \longrightarrow T(n) = 3n + n\,log_n\,n$$

Merge sort

$$T(n) = \begin{cases} 3, & n < 2 \\ 2T(\frac{n}{2}) + t_{merge}, & n \geq 2 \end{cases} \tag{5}$$

Merge sort

$$T(n) = \begin{cases} 3, & n < 2 \\ 2T(\frac{n}{2}) + n, & n \geq 2 \end{cases} \tag{6}$$

Merge sort

$$T(n) = \begin{cases} 3, & n < 2 \\ d\,T(\frac{n}{d}) + (d-1)\cdot n, & n \geq 2 \end{cases} \tag{7}$$

Merge sort

$$T(n) = dT(\tfrac{n}{d}) + (d-1) \cdot n$$

Merge sort

$$T(n) = dT(\tfrac{n}{d}) + (d-1) \cdot n$$

$$T(n) = d(dT(\tfrac{n}{d^2}) + (d-1)\tfrac{n}{d}) + (d-1) \cdot n$$

Merge sort

$$T(n) = dT(\tfrac{n}{d}) + (d-1) \cdot n$$

$$T(n) = d(dT(\tfrac{n}{d^2}) + (d-1)\tfrac{n}{d}) + (d-1) \cdot n$$

$$T(n) = d^2 T(\tfrac{n}{d^2}) + 2 \cdot (d-1) \cdot n$$

Merge sort

$$T(n) = dT(\tfrac{n}{d}) + (d-1) \cdot n$$

$$T(n) = d(dT(\tfrac{n}{d^2}) + (d-1)\tfrac{n}{d}) + (d-1) \cdot n$$

$$T(n) = d^2 T(\tfrac{n}{d^2}) + 2 \cdot (d-1) \cdot n$$

$$T(n) = d^2(dT(\tfrac{n}{d^3}) + (d-1) \cdot \tfrac{n}{d^2}) + 2 \cdot (d-1) \cdot n$$

Merge sort

$$T(n) = dT(\tfrac{n}{d}) + (d-1) \cdot n$$

$$T(n) = d(dT(\tfrac{n}{d^2}) + (d-1)\tfrac{n}{d}) + (d-1) \cdot n$$

$$T(n) = d^2 T(\tfrac{n}{d^2}) + 2 \cdot (d-1) \cdot n$$

$$T(n) = d^2(dT(\tfrac{n}{d^3}) + (d-1) \cdot \tfrac{n}{d^2}) + 2 \cdot (d-1) \cdot n$$

$$T(n) = d^3 T(\tfrac{n}{d^3}) + 3 \cdot (d-1) \cdot n$$

Merge sort

$$T(n) = dT(\tfrac{n}{d}) + (d-1)\cdot n$$

$$T(n) = d(dT(\tfrac{n}{d^2}) + (d-1)\tfrac{n}{d}) + (d-1)\cdot n$$

$$T(n) = d^2\,T(\tfrac{n}{d^2}) + 2\cdot(d-1)\cdot n$$

$$T(n) = d^2(dT(\tfrac{n}{d^3}) + (d-1)\cdot\tfrac{n}{d^2}) + 2\cdot(d-1)\cdot n$$

$$T(n) = d^3\,T(\tfrac{n}{d^3}) + 3\cdot(d-1)\cdot n$$

$$T(n) = d^k\,T(\tfrac{n}{d^k}) + k\cdot(d-1)\cdot n$$

Merge sort

$$T(n) = d^k T(\tfrac{n}{d^k}) + k \cdot (d-1) \cdot n$$

Merge sort

$$T(n) = d^k T(\tfrac{n}{d^k}) + k \cdot (d-1) \cdot n$$

- Fazendo $k = log_d\, n$:

Merge sort

$$T(n) = d^k T(\frac{n}{d^k}) + k \cdot (d-1) \cdot n$$

- Fazendo $k = log_d\, n$:

$$T(n) = d^{log_d\, n} T(\frac{n}{d^{log_d\, n}}) + log_d\, n \cdot (d-1) \cdot n$$

Merge sort

$$T(n) = d^k T(\frac{n}{d^k}) + k \cdot (d-1) \cdot n$$

- Fazendo $k = log_d\, n$:

$$T(n) = d^{log_d\, n} T(\frac{n}{d^{log_d\, n}}) + log_d\, n \cdot (d-1) \cdot n$$

$$T(n) = nT(\frac{n}{n}) + log_d\, n \cdot (d-1) \cdot n$$

Merge sort

$$T(n) = d^k T(\frac{n}{d^k}) + k \cdot (d-1) \cdot n$$

- Fazendo $k = log_d\, n$:

$$T(n) = d^{log_d\, n} T(\frac{n}{d^{log_d\, n}}) + log_d\, n \cdot (d-1) \cdot n$$

$$T(n) = nT(\frac{n}{n}) + log_d\, n \cdot (d-1) \cdot n$$

$$T(n) = nT(1) + log_d\, n \cdot (d-1) \cdot n$$

Merge sort

$$T(n) = d^k T(\frac{n}{d^k}) + k \cdot (d-1) \cdot n$$

- Fazendo $k = log_d\, n$:

$$T(n) = d^{log_d\, n} T(\frac{n}{d^{log_d\, n}}) + log_d\, n \cdot (d-1) \cdot n$$

$$T(n) = nT(\frac{n}{n}) + log_d\, n \cdot (d-1) \cdot n$$

$$T(n) = nT(1) + log_d\, n \cdot (d-1) \cdot n$$

$$T(n) = 3n + log_d\, n \cdot (d-1) \cdot n$$

Merge sort

$$T(n) = 3n + (d-1) \cdot n \log_d n$$

Merge sort

$$T(n) = 3n + (d-1) \cdot n \log_d n$$

$$d = 2 \longrightarrow T(n) = 3n + (2-1) n \log_2 n = 3n + n \log_2 n$$

Merge sort

$$T(n) = 3n + (d-1) \cdot n \log_d n$$

$$d = 2 \longrightarrow T(n) = 3n + (2-1)n \log_2 n = 3n + n \log_2 n$$

$$d = 4 \longrightarrow T(n) = 3n + (4-1)n \log_4 n = 3n + 3n \log_4 n$$

Merge sort

$$T(n) = 3n + (d-1) \cdot n \log_d n$$

$$d = 2 \longrightarrow T(n) = 3n + (2-1)n \log_2 n = 3n + n \log_2 n$$

$$d = 4 \longrightarrow T(n) = 3n + (4-1)n \log_4 n = 3n + 3n \log_4 n$$

$$d = n \longrightarrow T(n) = 3n + (n-1)n \log_n n = 3n + (n-1)n$$

Merge sort

$$T(n) = 3n + log_d\, n \cdot (d-1) \cdot n$$

$$d = 2 \longrightarrow T(n) = 3n + n\, log_2\, n \in \Theta(n\, log\, n)$$

Merge sort

$$T(n) = 3n + log_d\, n \cdot (d-1) \cdot n$$

$$d = 2 \longrightarrow T(n) = 3n + n\, log_2\, n \in \Theta(n\, log\, n)$$

$$d = 4 \longrightarrow T(n) = 3n + 3n\, log_4\, n \in \Theta(n\, log\, n)$$

Merge sort

$$T(n) = 3n + log_d\, n \cdot (d-1) \cdot n$$

$$d = 2 \longrightarrow T(n) = 3n + n\, log_2\, n \in \Theta(n\, log\, n)$$

$$d = 4 \longrightarrow T(n) = 3n + 3n\, log_4\, n \in \Theta(n\, log\, n)$$

$$d = n \longrightarrow T(n) = 3n + (n-1)n \in \Theta(n^2)$$

```
1  void quick_sort (int v[], int inicio, int fim) {
2       if (fim - inicio < 2) return;
3       else {
4            int p = partition(v, inicio, fim);
5            quick_sort(v, inicio, p);
6            quick_sort(v, p+1, fim);
7       }
8  }
```

Quick sort

Algoritmo 3 quickSort (vetor v, índices $\langle inicio, fim \rangle$)

Quick sort

Algoritmo 3 quickSort (vetor v, índices $\langle inicio, fim \rangle$)

```
if fim - inicio < 2 then
    return
```

Quick sort

Algoritmo 3 quickSort (vetor v, índices $\langle inicio, fim \rangle$)

if fim - inicio $<$ 2 **then**

return

else

end if

Quick sort

Algoritmo 3 quickSort (vetor v, índices $\langle inicio, fim \rangle$)

if fim - inicio $<$ 2 **then**

return

else

p $\leftarrow$ partition(v, inicio, fim)

end if

Quick sort

Algoritmo 3 quickSort (vetor v, índices $\langle inicio, fim \rangle$)

```
if fim - inicio < 2 then
    return
else
    p ← partition(v, inicio, fim)
    quickSort(v, inicio, p)

end if
```

Quick sort

Algoritmo 3 quickSort (vetor v, índices $\langle inicio, fim \rangle$)

```
if fim - inicio < 2 then
    return
else
    p ← partition(v, inicio, fim)
    quickSort(v, inicio, p)
    quickSort(v, p+1, fim)
end if
```

Quick sort

$$T(n) = \begin{cases} 2, & n < 2 \\ t_{partition} + T(menor) + T(maior), & n \geq 2 \end{cases} \tag{8}$$

```
1  void partition (int v[], int inicio, int fim) {
2      int aux = inicio, i = inicio + 1, j = fim - 1;
3      while (i <= j) {
4          while (i <= j && v[i] <= v[aux]) i++;
5          while (i <= j && v[j] >= v[aux]) j--;
6          if (v[i] > v[j]) swap(v[i++], v[j--]);
7      }
8      swap(v[aux],v[j]);
9      return j;
10 }
```

Algoritmo 4 partition(vetor v, índices $\langle inicio, fim \rangle$)

Algoritmo 4 partition(vetor v, índices $\langle inicio, fim \rangle$)

aux $\leftarrow$ inicio;

Algoritmo 4 partition(vetor v, índices $\langle inicio, fim \rangle$)

aux $\leftarrow$ inicio; i $\leftarrow$ inicio $+$ 1; j $\leftarrow$ fim - 1

Algoritmo 4 partition(vetor v, índices $\langle inicio, fim \rangle$)

aux $\leftarrow$ inicio; i $\leftarrow$ inicio + 1; j $\leftarrow$ fim - 1

while i $\leq$ j **do**

end while

Algoritmo 4 partition(vetor v, índices $\langle inicio, fim \rangle$)

```
aux ← inicio; i ← inicio + 1; j ← fim - 1
while i ≤ j do
    while i ≤ j and v[i] ≤ v[aux] do
        i ← i + 1
    end while




end while
```

Algoritmo 4 partition(vetor v, índices $\langle inicio, fim \rangle$)

```
aux ← inicio; i ← inicio + 1; j ← fim - 1
while i ≤ j do
    while i ≤ j and v[i] ≤ v[aux] do
        i ← i + 1
    end while
    while i ≤ j and v[j] > v[aux] do
        j ← j - 1
    end while




end while
```

Algoritmo 4 partition(vetor v, índices $\langle inicio, fim \rangle$)

```
aux ← inicio; i ← inicio + 1; j ← fim - 1
while i ≤ j do
    while i ≤ j and v[i] ≤ v[aux] do
        i ← i + 1
    end while
    while i ≤ j and v[j] > v[aux] do
        j ← j - 1
    end while
    if v[i] > v[j] then
        swap(v[i], v[j])
        i ← i + 1, j ← j - 1
    end if
end while
```

Algoritmo 4 partition(vetor v, índices $\langle inicio, fim \rangle$)

```
aux ← inicio; i ← inicio + 1; j ← fim - 1
while i ≤ j do
    while i ≤ j and v[i] ≤ v[aux] do
        i ← i + 1
    end while
    while i ≤ j and v[j] > v[aux] do
        j ← j - 1
    end while
    if v[i] > v[j] then
        swap(v[i], v[j])
        i ← i + 1, j ← j - 1
    end if
end while
swap(v[aux], v[j])
```

Algoritmo 4 partition(vetor v, índices $\langle inicio, fim \rangle$)

```
aux ← inicio; i ← inicio + 1; j ← fim - 1
while i ≤ j do
    while i ≤ j and v[i] ≤ v[aux] do
        i ← i + 1
    end while
    while i ≤ j and v[j] > v[aux] do
        j ← j - 1
    end while
    if v[i] > v[j] then
        swap(v[i], v[j])
        i ← i + 1, j ← j - 1
    end if
end while
swap(v[aux], v[j])
return j
```

Quick sort

$$T(n) = \begin{cases} 2, & n < 2 \\ \Theta(n) + T(menor) + T(maior), & n \geq 2 \end{cases} \tag{9}$$

Melhor caso

$$T(n) = 2T(\tfrac{n}{2}) + n$$

Melhor caso

$$T(n) = 2T(\tfrac{n}{2}) + n$$

- Igual ao mergesort:

Melhor caso

$$T(n) = 2T(\tfrac{n}{2}) + n$$

- Igual ao mergesort:

$$T(n) = nT(1) + n \cdot log_2\, n$$

Quick sort

$$T(n) = \begin{cases} 2, & n < 2 \\ n + T(menor) + T(maior), & n \geq 2 \end{cases} \tag{10}$$

Melhor caso

$$T(n) = 2T(\tfrac{n}{2}) + n$$

- Igual ao mergesort:

$$T(n) = nT(1) + n \cdot log_2\, n$$

Melhor caso

$$T(n) = 2T(\tfrac{n}{2}) + n$$

- Igual ao mergesort:

$$T(n) = nT(1) + n \cdot log_2\, n$$

$$T(n) = 2n + n \cdot log_2\, n$$

- Pior caso

$$T(n) = n + T(n-1) + T(0) = n + T(n-1) + 2$$

- Pior caso

$$T(n) = n + T(n-1) + T(0) = n + T(n-1) + 2$$

- Iteração (simplificando)

- Pior caso

$$T(n) = n + T(n-1) + T(0) = n + T(n-1) + 2$$

- Iteração (simplificando)

$$T(n) = T(n-1) + n$$

- Pior caso

$$T(n) = n + T(n-1) + T(0) = n + T(n-1) + 2$$

- Iteração (simplificando)

$$T(n) = T(n-1) + n$$

$$T(n = (T(n-2) + n - 1) + n = T(n-2) + 2n - 1$$

- Pior caso

$$T(n) = n + T(n-1) + T(0) = n + T(n-1) + 2$$

- Iteração (simplificando)

$$T(n) = T(n-1) + n$$

$$T(n = (T(n-2) + n - 1) + n = T(n-2) + 2n - 1$$

$$T(n) = (T(n-3) + n - 2) + 2n - 1 = T(n-3) + 3n - 3$$

- Pior caso

$$T(n) = n + T(n-1) + T(0) = n + T(n-1) + 2$$

- Iteração (simplificando)

$$T(n) = T(n-1) + n$$

$$T(n = (T(n-2) + n - 1) + n = T(n-2) + 2n - 1$$

$$T(n) = (T(n-3) + n - 2) + 2n - 1 = T(n-3) + 3n - 3$$

$$T(n) = (T(n-4) + n - 3) + 3n - 3 = T(n-4) + 4n - 6$$

- Pior caso

$$T(n) = n + T(n-1) + T(0) = n + T(n-1) + 2$$

- Iteração (simplificando)

$$T(n) = T(n-1) + n$$

$$T(n = (T(n-2) + n - 1) + n = T(n-2) + 2n - 1$$

$$T(n) = (T(n-3) + n - 2) + 2n - 1 = T(n-3) + 3n - 3$$

$$T(n) = (T(n-4) + n - 3) + 3n - 3 = T(n-4) + 4n - 6$$

$$T(n) = (T(n-5) + n - 4) + 3n - 6 = T(n-5) + 5n - 10$$

- Pior caso

$$T(n) = n + T(n-1) + T(0) = n + T(n-1) + 2$$

- Iteração (simplificando)

$$T(n) = T(n-1) + n$$

$$T(n = (T(n-2) + n - 1) + n = T(n-2) + 2n - 1$$

$$T(n) = (T(n-3) + n - 2) + 2n - 1 = T(n-3) + 3n - 3$$

$$T(n) = (T(n-4) + n - 3) + 3n - 3 = T(n-4) + 4n - 6$$

$$T(n) = (T(n-5) + n - 4) + 3n - 6 = T(n-5) + 5n - 10$$

$$T(n) = T(n-k) + kn - \sum_{i=1}^{k-1} i$$

- Pior caso

$$T(n) = T(n-k) + kn - \sum_{i=1}^{k-1} i$$

- Pior caso

$$T(n) = T(n-k) + kn - \sum_{i=1}^{k-1} i$$

$$T(n) = T(n-k) + kn - \frac{k(k-1)}{2}$$

- Pior caso

$$T(n) = T(n-k) + kn - \sum_{i=1}^{k-1} i$$

$$T(n) = T(n-k) + kn - \frac{k(k-1)}{2}$$

- Fazendo $k = n$:

- Pior caso

$$T(n) = T(n-k) + kn - \sum_{i=1}^{k-1} i$$

$$T(n) = T(n-k) + kn - \frac{k(k-1)}{2}$$

- Fazendo $k = n$:

$$T(n) = T(n-n) + n \cdot n - \frac{n(n-1)}{2}$$

- Pior caso

$$T(n) = T(n-k) + kn - \sum_{i=1}^{k-1} i$$

$$T(n) = T(n-k) + kn - \frac{k(k-1)}{2}$$

- Fazendo $k = n$:

$$T(n) = T(n-n) + n \cdot n - \frac{n(n-1)}{2}$$

$$T(n) = T(0) + n^2 - \frac{(n^2-n)}{2}$$

- Pior caso

$$T(n) = T(n-k) + kn - \sum_{i=1}^{k-1} i$$

$$T(n) = T(n-k) + kn - \frac{k(k-1)}{2}$$

- Fazendo $k = n$:

$$T(n) = T(n-n) + n \cdot n - \frac{n(n-1)}{2}$$

$$T(n) = T(0) + n^2 - \frac{(n^2-n)}{2}$$

$$T(n) = T(0) + \frac{n^2}{2} + \frac{n}{2}$$

$$T(n) = 2 + \frac{n^2}{2} + \frac{n}{2} \longrightarrow T(n) \in \Theta(n^2)$$

Escolha do pivô

- Extremidade do vetor: $\Theta(1)$

Escolha do pivô

- Extremidade do vetor: $\Theta(1) \longrightarrow T_{qsort}(n) \in \Theta(n^2)$

Escolha do pivô

- Extremidade do vetor: $\Theta(1) \longrightarrow T_{qsort}(n) \in \Theta(n^2)$
- Elemento central do vetor: $\Theta(1)$

Algoritmo 4 partition(vetor v, índices $\langle inicio, fim \rangle$)

```
aux ← inicio; i ← inicio + 1; j ← fim - 1
while i ≤ j do
    while i ≤ j and v[i] ≤ v[aux] do
        i ← i + 1
    end while
    while i ≤ j and v[j] > v[aux] do
        j ← j - 1
    end while
    if v[i] > v[j] then
        swap(v[i], v[j])
        i ← i + 1, j ← j - 1
    end if
end while
swap(v[aux], v[j])
return  j
```

Algoritmo 4 partition(vetor v, índices $\langle inicio, fim \rangle$)

```
aux ← (fim - inicio) / 2;
aux ← inicio; i ← inicio + 1; j ← fim - 1
while i ≤ j do
    while i ≤ j and v[i] ≤ v[aux] do
        i ← i + 1
    end while
    while i ≤ j and v[j] > v[aux] do
        j ← j - 1
    end while
    if v[i] > v[j] then
        swap(v[i], v[j])
        i ← i + 1, j ← j - 1
    end if
end while
swap(v[aux], v[j])
return  j
```

Algoritmo 4 partition(vetor v, índices $\langle inicio, fim \rangle$)

```
aux ← (fim - inicio) / 2; swap(v[inicio], v[aux])
aux ← inicio; i ← inicio + 1; j ← fim - 1
while i ≤ j do
    while i ≤ j and v[i] ≤ v[aux] do
        i ← i + 1
    end while
    while i ≤ j and v[j] > v[aux] do
        j ← j - 1
    end while
    if v[i] > v[j] then
        swap(v[i], v[j])
        i ← i + 1, j ← j - 1
    end if
end while
swap(v[aux], v[j])
return  j
```

Escolha do pivô

- Extremidade do vetor: $\Theta(1) \longrightarrow T_{qsort}(n) \in \Theta(n^2)$
- Elemento central do vetor: $\Theta(1)$

Escolha do pivô

- Extremidade do vetor: $\Theta(1) \longrightarrow T_{qsort}(n) \in \Theta(n^2)$
- Elemento central do vetor: $\Theta(1) \longrightarrow T_{qsort}(n) \in \Theta(n^2)$

Escolha do pivô

- Extremidade do vetor: $\Theta(1) \longrightarrow T_{qsort}(n) \in \Theta(n^2)$
- Elemento central do vetor: $\Theta(1) \longrightarrow T_{qsort}(n) \in \Theta(n^2)$
- Aleatório: $\Theta(1)$

Algoritmo 4 partition(vetor v, índices $\langle inicio, fim \rangle$)

```
aux ← inicio; i ← inicio + 1; j ← fim - 1
while i ≤ j do
    while i ≤ j and v[i] ≤ v[aux] do
        i ← i + 1
    end while
    while i ≤ j and v[j] > v[aux] do
        j ← j - 1
    end while
    if v[i] > v[j] then
        swap(v[i], v[j])
        i ← i + 1, j ← j - 1
    end if
end while
swap(v[aux], v[j])
return  j
```

Algoritmo 4 partition(vetor v, índices $\langle inicio, fim \rangle$)

```
aux = rand(inicio, fim);
aux ← inicio; i ← inicio + 1; j ← fim - 1
while i ≤ j do
    while i ≤ j and v[i] ≤ v[aux] do
        i ← i + 1
    end while
    while i ≤ j and v[j] > v[aux] do
        j ← j - 1
    end while
    if v[i] > v[j] then
        swap(v[i], v[j])
        i ← i + 1, j ← j - 1
    end if
end while
swap(v[aux], v[j])
return  j
```

Algoritmo 4 partition(vetor v, índices $\langle inicio, fim \rangle$)

```
aux = rand(inicio, fim); swap(v[inicio], v[aux])
aux ← inicio; i ← inicio + 1; j ← fim - 1
while i ≤ j do
    while i ≤ j and v[i] ≤ v[aux] do
        i ← i + 1
    end while
    while i ≤ j and v[j] > v[aux] do
        j ← j - 1
    end while
    if v[i] > v[j] then
        swap(v[i], v[j])
        i ← i + 1, j ← j - 1
    end if
end while
swap(v[aux], v[j])
return  j
```

Escolha do pivô

- Extremidade do vetor: $\Theta(1) \longrightarrow T_{qsort}(n) \in \Theta(n^2)$
- Elemento central do vetor: $\Theta(1) \longrightarrow T_{qsort}(n) \in \Theta(n^2)$
- Aleatório: $\Theta(1)$

Escolha do pivô

- Extremidade do vetor: $\Theta(1) \longrightarrow T_{qsort}(n) \in \Theta(n^2)$
- Elemento central do vetor: $\Theta(1) \longrightarrow T_{qsort}(n) \in \Theta(n^2)$
- Aleatório: $\Theta(1) \longrightarrow T_{qsort}(n) \in \Theta(n^2)$

Escolha do pivô

- Extremidade do vetor: $\Theta(1) \longrightarrow T_{qsort}(n) \in \Theta(n^2)$
- Elemento central do vetor: $\Theta(1) \longrightarrow T_{qsort}(n) \in \Theta(n^2)$
- Aleatório: $\Theta(1) \longrightarrow T_{qsort}(n) \in \Theta(n^2)$
- Mediana: $\Theta(n)$

Escolha do pivô

- Extremidade do vetor: $\Theta(1) \longrightarrow T_{qsort}(n) \in \Theta(n^2)$
- Elemento central do vetor: $\Theta(1) \longrightarrow T_{qsort}(n) \in \Theta(n^2)$
- Aleatório: $\Theta(1) \longrightarrow T_{qsort}(n) \in \Theta(n^2)$
- Mediana: $\Theta(n) \longrightarrow T_{qsort}(n) \in \Theta(n \log n)$

Generalizando o método da iteração

- Merge/quick sort: $T(n) = 2T(\frac{n}{2}) + n \longrightarrow T(n) \in \Theta(n \log n)$

Generalizando o método da iteração

- Merge/quick sort: $T(n) = 2T(\frac{n}{2}) + n \longrightarrow T(n) \in \Theta(n \log n)$
- Busca binária Seq: $T(n) = T(\frac{n}{2}) + 3 \longrightarrow T(n) \in \Theta(\log n)$

Generalizando o método da iteração

- Merge/quick sort: $T(n) = 2T(\frac{n}{2}) + n \longrightarrow T(n) \in \Theta(n \log n)$
- Busca binária Seq: $T(n) = T(\frac{n}{2}) + 3 \longrightarrow T(n) \in \Theta(\log n)$
- Busca binária Lista: $T(n) = 2T(\frac{n}{2}) + 4 \longrightarrow T(n) \in \Theta(n)$

Generalizando o método da iteração

- Busca binária Seq: $T(n) = T(\frac{n}{2}) + 3 \longrightarrow T(n) \in \Theta(log\ n)$
- Busca binária Lista: $T(n) = 2T(\frac{n}{2}) + 4 \longrightarrow T(n) \in \Theta(n)$
- Merge/quick sort: $T(n) = 2T(\frac{n}{2}) + n \longrightarrow T(n) \in \Theta(n\ log\ n)$

Generalizando o método da iteração

$$T(n) = aT(\frac{n}{b}) + cn^d \tag{11}$$

$$T(n) = \begin{cases} \Theta(n^d \, log \, n), & \text{se } a = b^d \quad (\text{ou } log_b \, a = d) \\ \\ \\ \end{cases} \tag{12}$$

Generalizando o método da iteração

$$T(n) = aT(\frac{n}{b}) + cn^d \tag{11}$$

$$T(n) = \begin{cases} \Theta(n^d \, log \, n), & \text{se } a = b^d \quad (\text{ou } log_b \, a = d) \\ \Theta(n^d), & \text{se } a < b^d \quad (\text{ou } log_b \, a < d) \end{cases} \tag{12}$$

Generalizando o método da iteração

$$T(n) = aT(\frac{n}{b}) + cn^d \tag{11}$$

$$T(n) = \begin{cases} \Theta(n^d \, log \, n), & \text{se } a = b^d \quad (\text{ou } log_b \, a = d) \\ \Theta(n^d), & \text{se } a < b^d \quad (\text{ou } log_b \, a < d) \\ \Theta(n^{log_b \, a}), & \text{se } a > b^d \quad (\text{ou } log_b \, a > d) \end{cases} \tag{12}$$

Generalizando o método da iteração

Busca binária Seq: $T(n) = T(\frac{n}{2}) + 3$

Generalizando o método da iteração

Busca binária Seq: $T(n) = T(\frac{n}{2}) + 3$

a=1, b=2, d=0

Generalizando o método da iteração

Busca binária Seq: $T(n) = T(\frac{n}{2}) + 3$

a=1, b=2, d=0 $a = b^d$

Generalizando o método da iteração

Busca binária Seq: $T(n) = T(\frac{n}{2}) + 3$

a=1, b=2, d=0 $a = b^d$ $T(n) \in \Theta(n^d logn)$

Generalizando o método da iteração

Busca binária Seq: $T(n) = T(\frac{n}{2}) + 3 \longrightarrow T(n) \in \Theta(log\ n)$

a=1, b=2, d=0 $a = b^d$ $T(n) \in \Theta(n^d logn)$

Generalizando o método da iteração

Busca binária Seq: $T(n) = T(\frac{n}{2}) + 3 \longrightarrow T(n) \in \Theta(log\ n)$

a=1, b=2, d=0 $a = b^d$ $T(n) \in \Theta(n^d logn)$

Busca binária Lista: $T(n) = 2T(\frac{n}{2}) + 2$

Generalizando o método da iteração

Busca binária Seq: $T(n) = T(\frac{n}{2}) + 3 \longrightarrow T(n) \in \Theta(log\ n)$

a=1, b=2, d=0 $a = b^d$ $T(n) \in \Theta(n^d logn)$

Busca binária Lista: $T(n) = 2T(\frac{n}{2}) + 2$

a=2, b=2, d=0

Generalizando o método da iteração

Busca binária Seq: $T(n) = T(\frac{n}{2}) + 3 \longrightarrow T(n) \in \Theta(log\ n)$

a=1, b=2, d=0 $a = b^d$ $T(n) \in \Theta(n^d logn)$

Busca binária Lista: $T(n) = 2T(\frac{n}{2}) + 2$

a=2, b=2, d=0 $a > b^d$

Generalizando o método da iteração

Busca binária Seq: $T(n) = T(\frac{n}{2}) + 3 \longrightarrow T(n) \in \Theta(log\ n)$

a=1, b=2, d=0 $a = b^d$ $T(n) \in \Theta(n^d logn)$

Busca binária Lista: $T(n) = 2T(\frac{n}{2}) + 2$

a=2, b=2, d=0 $a > b^d$ $T(n) \in \Theta(n^{log_b a})$

Generalizando o método da iteração

Busca binária Seq: $T(n) = T(\frac{n}{2}) + 3 \longrightarrow T(n) \in \Theta(log\ n)$

a=1, b=2, d=0 $a = b^d$ $T(n) \in \Theta(n^d logn)$

Busca binária Lista: $T(n) = 2T(\frac{n}{2}) + 2 \longrightarrow T(n) \in \Theta(n)$

a=2, b=2, d=0 $a > b^d$ $T(n) \in \Theta(n^{log_b a})$

Generalizando o método da iteração

Merge/quick sort: $T(n) = 2T(\frac{n}{2}) + n$

Generalizando o método da iteração

Merge/quick sort: $T(n) = 2T(\frac{n}{2}) + n$

a=2, b=2, d=1

Generalizando o método da iteração

Merge/quick sort: $T(n) = 2T(\frac{n}{2}) + n$

a=2, b=2, d=1 $a = b^d$

Generalizando o método da iteração

Merge/quick sort: $T(n) = 2T(\frac{n}{2}) + n$

a=2, b=2, d=1 $a = b^d$ $T(n) \in \Theta(n^d logn)$

Generalizando o método da iteração

Merge/quick sort: $T(n) = 2T(\frac{n}{2}) + n \longrightarrow T(n) \in \Theta(n \log n)$

a=2, b=2, d=1 $a = b^d$ $T(n) \in \Theta(n^d \log n)$

Generalizando o método da iteração

Algoritmo 5 msum (matrizes $\langle A, B \rangle$, ordem n)

if n $= 1$ **then**
  return A $+$ B
else
  C_{11} = msum(A_{11}, B_{11}, $n/2$)
  C_{12} = msum(A_{12}, B_{12}, $n/2$)
  C_{21} = msum(A_{21}, B_{21}, $n/2$)
  C_{22} = msum(A_{22}, B_{22}, $n/2$)
  return C
end if

Generalizando o método da iteração

- Caso base: 2

Generalizando o método da iteração

- Caso base: 2
- Caso recursivo

Generalizando o método da iteração

- Caso base: 2
- Caso recursivo
  - Recursão: $4\,T(\frac{n}{2})$

Generalizando o método da iteração

- Caso base: 2
- Caso recursivo
  - Recursão: $4\,T(\frac{n}{2})$
  - Demais operações: 2

Generalizando o método da iteração

$$T(n) = \begin{cases} 2, & n = 1 \\ 4\,T(n/2) + 2, & n \geq 2 \end{cases} \tag{13}$$

Generalizando o método da iteração

Merge/quick sort: $T(n) = 2T(\frac{n}{2}) + n \longrightarrow T(n) \in \Theta(n \log n)$

a=2, b=2, d=1 $a = b^d$ $T(n) \in \Theta(n^d \log n)$

MSum: $T(n) = 4T(\frac{n}{2}) + 2$

Generalizando o método da iteração

Merge/quick sort: $T(n) = 2T(\frac{n}{2}) + n \longrightarrow T(n) \in \Theta(n \log n)$

a=2, b=2, d=1 $a = b^d$ $T(n) \in \Theta(n^d \log n)$

MSum: $T(n) = 4T(\frac{n}{2}) + 2$

a=4, b=2, d=0

Generalizando o método da iteração

Merge/quick sort: $T(n) = 2T(\frac{n}{2}) + n \longrightarrow T(n) \in \Theta(n \log n)$

a=2, b=2, d=1 $a = b^d$ $T(n) \in \Theta(n^d \log n)$

MSum: $T(n) = 4T(\frac{n}{2}) + 2$

a=4, b=2, d=0 $a > b^d$

Generalizando o método da iteração

Merge/quick sort: $T(n) = 2T(\frac{n}{2}) + n \longrightarrow T(n) \in \Theta(n \log n)$

a=2, b=2, d=1 $a = b^d$ $T(n) \in \Theta(n^d \log n)$

MSum: $T(n) = 4T(\frac{n}{2}) + 2$

a=4, b=2, d=0 $a > b^d$ $T(n) \in \Theta(n^{\log_b a})$

Generalizando o método da iteração

Merge/quick sort: $T(n) = 2T(\frac{n}{2}) + n \longrightarrow T(n) \in \Theta(n \log n)$

a=2, b=2, d=1 $a = b^d$ $T(n) \in \Theta(n^d \log n)$

MSum: $T(n) = 4T(\frac{n}{2}) + 2 \longrightarrow T(n) \in \Theta(n^2)$

a=4, b=2, d=0 $a > b^d$ $T(n) \in \Theta(n^{\log_b a})$
