Análise de Algoritmos

Parte destes slides são adaptações de slides

do Prof. Paulo Feofiloff e do Prof. José Coelho de Pina.

Bucketsort

CLRS sec 8.4

Bucket Sort

Recebe um inteiro n e um vetor $A[1\,..\,n]$ onde cada elemento é um número no intervalo $[0, 1)$.

Bucket Sort

Recebe um inteiro n e um vetor $A[1\,..\,n]$ onde cada elemento é um número no intervalo $[0, 1)$.

A	.47	.93	.82	.12	.42	.03	.62	.38	.77	.91

Bucket Sort

Recebe um inteiro n e um vetor $A[1\,..\,n]$ onde cada elemento é um número no intervalo $[0, 1)$.

A	.47	.93	.82	.12	.42	.03	.62	.38	.77	.91

Devolve um vetor $C[1\,..\,n]$ com os elementos de $A[1\,..\,n]$ em ordem crescente.

Bucket Sort

Recebe um inteiro n e um vetor $A[1\,..\,n]$ onde cada elemento é um número no intervalo $[0, 1)$.

A	.47	.93	.82	.12	.42	.03	.62	.38	.77	.91

Devolve um vetor $C[1\,..\,n]$ com os elementos de $A[1\,..\,n]$ em ordem crescente.

C	.03	.12	.38	.42	.47	.62	.77	.82	.91	.93

Exemplo

.47	.93	.82	.12	.42	.03	.62	.38	.77	.91

Exemplo

.47	.93	.82	.12	.42	.03	.62	.38	.77	.91

$B[0]$:	.03	
$B[1]$:	.12	
$B[2]$:		
$B[3]$:	.38	
$B[4]$:	.47	.42
$B[5]$:		
$B[6]$:	.62	
$B[7]$:	.77	
$B[8]$:	.82	
$B[9]$:	.93	.91

Exemplo

.47	.93	.82	.12	.42	.03	.62	.38	.77	.91

$B[0]$:	.03	
$B[1]$:	.12	
$B[2]$:		
$B[3]$:	.38	
$B[4]$:	.42	.47
$B[5]$:		
$B[6]$:	.62	
$B[7]$:	.77	
$B[8]$:	.82	
$B[9]$:	.91	.93

Exemplo

.47	.93	.82	.12	.42	.03	.62	.38	.77	.91

$B[0]$:	.03	
$B[1]$:	.12	
$B[2]$:		
$B[3]$:	.38	
$B[4]$:	.42	.47
$B[5]$:		
$B[6]$:	.62	
$B[7]$:	.77	
$B[8]$:	.82	
$B[9]$:	.91	.93

.03	.12	.38	.42	.47	.62	.77	.82	.91	.93

Bucket Sort

Recebe um inteiro n e um vetor $A[1\,..\,n]$ onde cada elemento é um número no intervalo $[0, 1)$.

Devolve um vetor $C[1\,..\,n]$ com os elementos de $A[1\,..\,n]$ em ordem crescente.

```
BUCKETSORT(A, n)
 1  para i ← 0 até n − 1 faça
 2        B[i] ← NIL
 3  para i ← 1 até n faça
 4        INSIRA(B[⌊n A[i]⌋], A[i])
 5  para i ← 0 até n − 1 faça
 6        ORDENELISTA(B[i])
 7  C ← CONCATENE(B, n)
 8  devolva C
```

Bucket Sort

```
BUCKETSORT(A, n)
 1  para i ← 0 até n − 1 faça
 2        B[i] ← NIL
 3  para i ← 1 até n faça
 4        INSIRA(B[⌊n A[i]⌋], A[i])
 5  para i ← 0 até n − 1 faça
 6        ORDENELISTA(B[i])
 7  C ← CONCATENE(B, n)
 8  devolva C
```

INSIRA(p, x): insere x na lista apontada por p

ORDENELISTA(p): ordena a lista apontada por p

CONCATENE(B, n): devolve a lista obtida da concatenação das listas apontadas por $B[0], \ldots, B[n-1]$.

Consumo de tempo

Suponha que os números em $A[1\,..\,n]$ são uniformemente distribuídos no intervalo $[0, 1)$.

Suponha que o ORDENELISTA seja o INSERTIONSORT.

Consumo de tempo

Suponha que os números em $A[1\,..\,n]$ são uniformemente distribuídos no intervalo $[0, 1)$.

Suponha que o ORDENELISTA seja o INSERTIONSORT.

Seja X_i o número de elementos na lista $B[i]$.

Consumo de tempo

Suponha que os números em $A[1\,..\,n]$ são uniformemente distribuídos no intervalo $[0, 1)$.

Suponha que o ORDENELISTA seja o INSERTIONSORT.

Seja X_i o número de elementos na lista $B[i]$.

Seja

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Consumo de tempo

Suponha que os números em $A[1\,..\,n]$ são uniformemente distribuídos no intervalo $[0, 1)$.

Suponha que o ORDENELISTA seja o INSERTIONSORT.

Seja X_i o número de elementos na lista $B[i]$.

Seja

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Observe que $X_i = \sum_j X_{ij}$.

Consumo de tempo

Suponha que os números em $A[1\,..\,n]$ são uniformemente distribuídos no intervalo $[0, 1)$.

Suponha que o ORDENELISTA seja o INSERTIONSORT.

Seja X_i o número de elementos na lista $B[i]$.

Seja

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Observe que $X_i = \sum_j X_{ij}$.

Y_i: número de comparações para ordenar a lista $B[i]$.

Consumo de tempo

X_i: número de elementos na lista $B[i]$

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Y_i: número de comparações para ordenar a lista $B[i]$.

Consumo de tempo

X_i: número de elementos na lista $B[i]$

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Y_i: número de comparações para ordenar a lista $B[i]$.

Observe que $Y_i \leq {X_i}^2$.

Logo $\mathrm{E}[Y_i] \leq \mathrm{E}[{X_i}^2] = \mathrm{E}[(\sum_j X_{ij})^2]$.

Consumo de tempo

X_i: número de elementos na lista $B[i]$

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Y_i: número de comparações para ordenar a lista $B[i]$.

Observe que $Y_i \leq {X_i}^2$.

Logo $\mathrm{E}[Y_i] \leq \mathrm{E}[{X_i}^2] = \mathrm{E}[(\sum_j X_{ij})^2]$.

$$\begin{aligned} \mathrm{E}[(\sum_j X_{ij})^2] &= \mathrm{E}[\sum_j \sum_k X_{ij} X_{ik}] \\ &= \mathrm{E}[\sum_j {X_{ij}}^2 + \sum_j \sum_{k \neq j} X_{ij} X_{ik}] \end{aligned}$$

Consumo de tempo

X_i: número de elementos na lista $B[i]$

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Y_i: número de comparações para ordenar a lista $B[i]$.

Observe que $Y_i \leq {X_i}^2$.

Logo $\mathrm{E}[Y_i] \leq \mathrm{E}[{X_i}^2] = \mathrm{E}[(\sum_j X_{ij})^2]$.

$$\begin{aligned} \mathrm{E}[(\sum_j X_{ij})^2] &= \mathrm{E}[\sum_j \sum_k X_{ij} X_{ik}] \\ &= \mathrm{E}[\sum_j {X_{ij}}^2] + \mathrm{E}[\sum_j \sum_{k \neq j} X_{ij} X_{ik}] \end{aligned}$$

Consumo de tempo

X_i: número de elementos na lista $B[i]$

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Y_i: número de comparações para ordenar a lista $B[i]$.

Observe que $Y_i \leq {X_i}^2$.

Logo $\mathrm{E}[Y_i] \leq \mathrm{E}[{X_i}^2] = \mathrm{E}[(\sum_j X_{ij})^2]$.

$$\begin{aligned} \mathrm{E}[(\sum_j X_{ij})^2] &= \mathrm{E}[\sum_j \sum_k X_{ij} X_{ik}] \\ &= \sum_j \mathrm{E}[{X_{ij}}^2] + \sum_j \sum_{k \neq j} \mathrm{E}[X_{ij} X_{ik}] \end{aligned}$$

Consumo de tempo

X_i: número de elementos na lista $B[i]$

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Y_i: número de comparações para ordenar a lista $B[i]$.

Observe que $Y_i \leq {X_i}^2$. Ademais,

$$\mathrm{E}[Y_i] \leq \sum_j \mathrm{E}[{X_{ij}}^2] + \sum_j \sum_{k \neq j} \mathrm{E}[X_{ij} X_{ik}].$$

Consumo de tempo

X_i: número de elementos na lista $B[i]$

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Y_i: número de comparações para ordenar a lista $B[i]$.

Observe que $Y_i \leq {X_i}^2$. Ademais,

$$\mathrm{E}[Y_i] \leq \sum_j \mathrm{E}[{X_{ij}}^2] + \sum_j \sum_{k \neq j} \mathrm{E}[X_{ij} X_{ik}].$$

Observe que ${X_{ij}}^2$ é uma variável aleatória binária. Vamos calcular sua esperança:

$$\mathrm{E}[{X_{ij}}^2] = \Pr[{X_{ij}}^2 = 1] = \Pr[X_{ij} = 1] = \frac{1}{n}.$$

Consumo de tempo

Para calcular $\mathrm{E}[X_{ij}X_{ik}]$ para $j \neq k$, primeiro note que X_{ij} e X_{ik} são variáveis aleatórias independentes.

Portanto, $\mathrm{E}[X_{ij}X_{ik}] = \mathrm{E}[X_{ij}]\mathrm{E}[X_{ik}]$.

Ademais, $\mathrm{E}[X_{ij}] = \Pr[X_{ij} = 1] = \frac{1}{n}$.

Consumo de tempo

Para calcular $\mathrm{E}[X_{ij}X_{ik}]$ para $j \neq k$, primeiro note que X_{ij} e X_{ik} são variáveis aleatórias independentes.

Portanto, $\mathrm{E}[X_{ij}X_{ik}] = \mathrm{E}[X_{ij}]\mathrm{E}[X_{ik}]$.

Ademais, $\mathrm{E}[X_{ij}] = \Pr[X_{ij} = 1] = \frac{1}{n}$.

Logo,

$$\begin{aligned}
\mathrm{E}[Y_i] &\leq \sum_j \frac{1}{n} + \sum_j \sum_{k \neq j} \frac{1}{n^2} \\
&= \frac{n}{n} + n(n-1)\frac{1}{n^2} \\
&= 1 + (n-1)\frac{1}{n} \\
&= 2 - \frac{1}{n}.
\end{aligned}$$

Consumo de tempo

Agora, seja $Y = \sum_i Y_i$.

Note que Y é o número de comparações realizadas pelo BUCKETSORT no total.

Assim $\mathrm{E}[Y]$ é o número esperado de comparações realizadas pelo algoritmo, e tal número determina o consumo assintótico de tempo do BUCKETSORT.

Mas então $\mathrm{E}[Y] = \sum_i \mathrm{E}[Y_i] \leq 2n - 1 = \mathrm{O}(n)$.

Consumo de tempo

Agora, seja $Y = \sum_i Y_i$.

Note que Y é o número de comparações realizadas pelo BUCKETSORT no total.

Assim $\mathrm{E}[Y]$ é o número esperado de comparações realizadas pelo algoritmo, e tal número determina o consumo assintótico de tempo do BUCKETSORT.

Mas então $\mathrm{E}[Y] = \sum_i \mathrm{E}[Y_i] \leq 2n - 1 = \mathrm{O}(n)$.

O consumo de tempo esperado do BUCKETSORT quando os números em $A[1\,..\,n]$ são uniformemente distribuídos no intervalo $[0, 1)$ é $\mathrm{O}(n)$.

Dicas de implementação

Mergesort

Dicas de implementação

Mergesort

- Onde declarar o vetor auxiliar?

Dicas de implementação

Mergesort

- Onde declarar o vetor auxiliar?
- Alternância entre dois vetores

Dicas de implementação

Mergesort

- Onde declarar o vetor auxiliar?
- Alternância entre dois vetores
- Teste se já ordenado

Dicas de implementação

Mergesort

- Onde declarar o vetor auxiliar?
- Alternância entre dois vetores
- Teste se já ordenado
- Inserção direta para vetores pequenos

Dicas de implementação

Mergesort

- Onde declarar o vetor auxiliar?
- Alternância entre dois vetores
- Teste se já ordenado
- Inserção direta para vetores pequenos

Quicksort

Dicas de implementação

Mergesort

- Onde declarar o vetor auxiliar?
- Alternância entre dois vetores
- Teste se já ordenado
- Inserção direta para vetores pequenos

Quicksort

- Elementos repetidos: partição ternária

Dicas de implementação

Mergesort

- Onde declarar o vetor auxiliar?
- Alternância entre dois vetores
- Teste se já ordenado
- Inserção direta para vetores pequenos

Quicksort

- Elementos repetidos: partição ternária
- Mediana de três

Dicas de implementação

Mergesort

- Onde declarar o vetor auxiliar?
- Alternância entre dois vetores
- Teste se já ordenado
- Inserção direta para vetores pequenos

Quicksort

- Elementos repetidos: partição ternária
- Mediana de três
- Recursão de cauda

Dicas de implementação

Mergesort

- Onde declarar o vetor auxiliar?
- Alternância entre dois vetores
- Teste se já ordenado
- Inserção direta para vetores pequenos

Quicksort

- Elementos repetidos: partição ternária
- Mediana de três
- Recursão de cauda
- Inserção direta para vetores pequenos

Dicas de implementação: Mergesort

Rearranja $A[p\,..\,r]$ em ordem crescente.

```
MERGESORT (A, p, r)
1   se p < r
2       então q ← ⌊(p + r)/2⌋
3             MERGESORT (A, p, q)
4             MERGESORT (A, q + 1, r)
5             INTERCALA (A, p, q, r)
```

- Onde declarar o vetor auxiliar usado no INTERCALA?

Dicas de implementação: Mergesort

```
MERGESORT (A, n)                    ⟵ aqui
1   MERGESORTREC (A, 1, n)

MERGESORTREC (A, p, r)
1   se p < r
2       então q ← ⌊(p + r)/2⌋
3             MERGESORTREC (A, p, q)
4             MERGESORTREC (A, q + 1, r)
5             INTERCALA (A, p, q, r)
```

- Onde declarar o vetor auxiliar usado no INTERCALA?

Dicas de implementação: Mergesort

```
MERGESORT (A, n)
1   MERGESORTREC (A, 1, n)

MERGESORTREC (A, p, r)
1   se p < r
2       então q ← ⌊(p + r)/2⌋
3             MERGESORTREC (A, p, q)
4             MERGESORTREC (A, q + 1, r)
5             INTERCALA (A, p, q, r)
```

- Onde declarar o vetor auxiliar usado no INTERCALA?
- Alternância entre dois vetores

Dicas de implementação: Mergesort

```
MERGESORT (A, n)
1   B ← A
2   MERGESORTREC (B, A, 1, n)

MERGESORTREC (A, B, p, r)
1   se p < r
2       então q ← ⌊(p + r)/2⌋
3             MERGESORTREC (B, A, p, q)
4             MERGESORTREC (B, A, q + 1, r)
5             INTERCALA (A, B, p, q, r)
```

- Onde declarar o vetor auxiliar usado no INTERCALA?
- Alternância entre dois vetores

Dicas de implementação: Mergesort

```
INTERCALA (A, B, p, q, r)
1   i ← p    j ← q + 1    k ← p
2   enquanto i ≤ q e j ≤ r faça
3       se A[i] ≤ A[j]
4           então B[k] ← A[i]    k ← k+1    i ← i+1
5           senão B[k] ← A[j]    k ← k+1    j ← j+1
6   enquanto i ≤ q faça
7       B[k] ← A[i]    k ← k+1    i ← i+1
8   enquanto j ≤ r faça
9       B[k] ← A[j]    k ← k+1    j ← j+1
```

- Onde declarar o vetor auxiliar usado no INTERCALA?
- Alternância entre dois vetores

Dicas de implementação: Mergesort

```
INTERCALA (A, p, q, r)
1   para i ← p até q faça B[i] ← A[i]
2   para j ← q + 1 até r faça B[r + q + 1 − j] ← A[j]
3   i ← p    j ← r
4   para k ← p até r faça
5       se B[i] ≤ B[j]
6           então A[k] ← B[i]
7                 i ← i + 1
8           senão A[k] ← B[j]
9                 j ← j − 1
```

- Onde declarar o vetor auxiliar usado no INTERCALA?
- Alternância entre dois vetores
- Teste se já ordenado

Dicas de implementação: Mergesort

```
INTERCALA (A, p, q, r)
0   se A[q] > A[q+1] então          ⟵ aqui
1       para i ← p até q faça B[i] ← A[i]
2       para j ← q+1 até r faça B[r+q+1−j] ← A[j]
3       i ← p    j ← r
4       para k ← p até r faça
5           se B[i] ≤ B[j]
6               então A[k] ← B[i]
7                     i ← i+1
8               senão A[k] ← B[j]
9                     j ← j−1
```

- Onde declarar o vetor auxiliar usado no INTERCALA?
- Alternância entre dois vetores
- Teste se já ordenado

Dicas de implementação: Quicksort

```
QUICKSORT (A, p, r)
1    se p < r
2         então q ← PARTICIONE (A, p, r)
3               QUICKSORT (A, p, q − 1)
4               QUICKSORT (A, q + 1, r)
```

Dicas de implementação: Quicksort

```
QUICKSORT (A, p, r)
1    se p < r
2         então (q, t) ← TRIPARTICIONE (A, p, r)
3               QUICKSORT (A, p, q − 1)
4               QUICKSORT (A, t + 1, r)
```

- Elementos repetidos: partição ternária

Dicas de implementação: Quicksort

PARTICIONE-ALEA(A, p, r)
1 $i \leftarrow$ RANDOM (p, r)
2 $A[i] \leftrightarrow A[r]$
3 **devolva** PARTICIONE (A, p, r)

- Elementos repetidos: partição ternária
- Mediana de três

Dicas de implementação: Quicksort

```
PARTICIONE-ALEA(A, p, r)
1   i ← RANDOM (p, r)
2   A[i] ↔ A[r]
3   devolva PARTICIONE (A, p, r)
```

- Elementos repetidos: partição ternária
- Mediana de três

Em vez de escolher um pivô aleatoriamente,
escolhe três elementos do vetor aleatoriamente,
e usa como pivô a mediana dos três.

Dicas de implementação: Quicksort

PARTICIONE-ALEA(A, p, r)
1 $i \leftarrow$ RANDOM (p, r)
2 $A[i] \leftrightarrow A[r]$
3 **devolva** PARTICIONE (A, p, r)

- Elementos repetidos: partição ternária
- Mediana de três

Em vez de escolher um pivô aleatoriamente,
escolhe três elementos do vetor aleatoriamente,
e usa como pivô a mediana dos três.

Aproveite para colocar o menor dos três no extremo
esquerdo, e o maior dos três no extremo direito,
e use-os de sentinela na partição!

Particione

Rearranja $A[p\,..\,r]$ de modo que $p \le q \le r$ e
$A[p\,..\,q{-}1] \le A[q] < A[q{+}1\,..\,r]$

Supõe que $A[p] \le A[r] \le A[r-1]$ e use $A[r]$ como pivô.

```
PARTICIONE (A, p, r)
1   x ← A[r]          ▷ x é o "pivô"
2   i ← p + 1    j ← r − 2
3   enquanto i ≤ j faça
4         enquanto A[i] ≤ x faça i ← i + 1
5         enquanto A[j] ≥ x faça j ← j − 1
6         se i < j então A[i] ↔ A[j]
7   A[i] ↔ A[r]
8   devolva i
```

Dicas de implementação: Quicksort

```
QUICKSORT (A, p, r)
1    se p < r
2         então q ← PARTICIONE (A, p, r)
3               QUICKSORT (A, p, q − 1)
4               QUICKSORT (A, q + 1, r)        ⟵ aqui
```

- Elementos repetidos: partição ternária
- Mediana de três
- Recursão de cauda

Dicas de implementação: Quicksort

```
QUICKSORT (A, p, r)
1   enquanto p < r faça        ⟵ virou enquanto aqui!
2       q ← PARTICIONE (A, p, r)
3       QUICKSORT (A, p, q − 1)
4       p ← q + 1              ⟵ ajusta os parâmetros!
```

- Elementos repetidos: partição ternária
- Mediana de três
- Recursão de cauda
  Substitua essa chamada recursiva por um enquanto.

Dicas de implementação: Quicksort

```
QUICKSORT (A, p, r)
1   enquanto p < r faça          ⟵ virou enquanto aqui!
2       q ← PARTICIONE (A, p, r)
3       QUICKSORT (A, p, q − 1)
4       p ← q + 1    ⟵ ajusta os parâmetros!
```

- Elementos repetidos: partição ternária
- Mediana de três
- Recursão de cauda
  Correto: substitua a chamada da maior metade!

Dicas de implementação: Quicksort

```
QUICKSORT (A, p, r)
1    enquanto p < r              ⟵ virou enquanto aqui!
2        q ← PARTICIONE (A, p, r)
3        se q − p < r − q
4            então QUICKSORT (A, p, q − 1)
5                  p ← q + 1
6            senão QUICKSORT (A, q + 1, r)
7                  r ← q − 1
```

- Elementos repetidos: partição ternária
- Mediana de três
- Recursão de cauda
  Correto: substitua a chamada da maior metade!

Dicas de implementação: Quicksort

```
QUICKSORT (A, p, r)
1    enquanto p < r              ⟵ virou enquanto aqui!
2          q ← PARTICIONE (A, p, r)
3          se q − p < r − q
4                então QUICKSORT (A, p, q − 1)
5                     p ← q + 1
6                senão QUICKSORT (A, q + 1, r)
7                     r ← q − 1
```

- Elementos repetidos: partição ternária
- Mediana de três
- Recursão de cauda
  Correto: substitua a chamada da maior metade!
  Profundidade da pilha da recursão: $\mathrm{O}(\lg n)$.

Dicas de implementação

Inserção direta para vetores pequenos.

Mergesort

- Onde declarar o vetor auxiliar usado no INTERCALA?
- Alternância entre dois vetores
- Teste se já ordenado

Quicksort

- Elementos repetidos: partição ternária
- Mediana de três
- Recursão de cauda

Heapsort

- Desce tudo sem comparar, depois sobe o que precisa.

Comparação entre os algoritmos

Mergesort:

Entre $\frac{1}{2}\, n \lg n$ e $n \lg n$ comparações.
Usa espaço extra linear.

Comparação entre os algoritmos

Mergesort:

Entre $\frac{1}{2}\, n \lg n$ e $n \lg n$ comparações.
Usa espaço extra linear.

Quicksort:

Em média, $2n \lg n$ comparações (e $\frac{1}{3}\, n \lg n$ trocas).
Espaço extra logarítmico.

Comparação entre os algoritmos

Mergesort:

Entre $\frac{1}{2}\, n \lg n$ e $n \lg n$ comparações.
Usa espaço extra linear.

Quicksort:

Em média, $2n \lg n$ comparações (e $\frac{1}{3}\, n \lg n$ trocas).
Espaço extra logarítmico.

Heapsort:

Menos de $2n \lg n + 2n$ comparações.
(Em média, cai para metade disso com a melhora).
Espaço extra constante.
Performance de cache ruim.