Estruturas de Dados

Aula 4.3 – Algoritmos de Ordenação - Quick Sort e Heap Sort

Prof. Ana Carolina Sokolonski

Bacharelado em Sistemas de Informação
Instituto Federal da Bahia – Campus Feira de Santana

2026

Agenda

Quick Sort

Heap Sort

Referências

Quick Sort

Ordenação por Pivoteamento

Quick Sort — Conceito

Como funciona

Algoritmo de **divisão e conquista** baseado na operação de **particionamento**: escolhe-se um **pivô** e reorganiza o vetor de forma que todos os elementos à **esquerda** sejam menores ou iguais ao pivô e os à **direita** sejam maiores. Repete recursivamente para cada metade. [Cormen et al. 2009]

Complexidade

Melhor/médio:	$O(N \log N)$
Pior caso:	$O(N^2)$

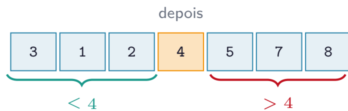
Na prática, **mais veloz** que Merge Sort e Heap Sort (constantes menores).

Vídeo: `https://youtu.be/ywWBy6J5gz8`

Particionamento com pivô = 4

antes

depois



Quick Sort — Escolha do Pivô

O problema central

A escolha do pivô determina a eficiência do Quick Sort. No **pior caso** (vetor já ordenado com pivô no extremo), a complexidade cai para $O(N^2)$.

Estratégias problemáticas

- **Primeiro elemento** — $O(N^2)$ se ordenado
- **Último elemento** — $O(N^2)$ se ordenado

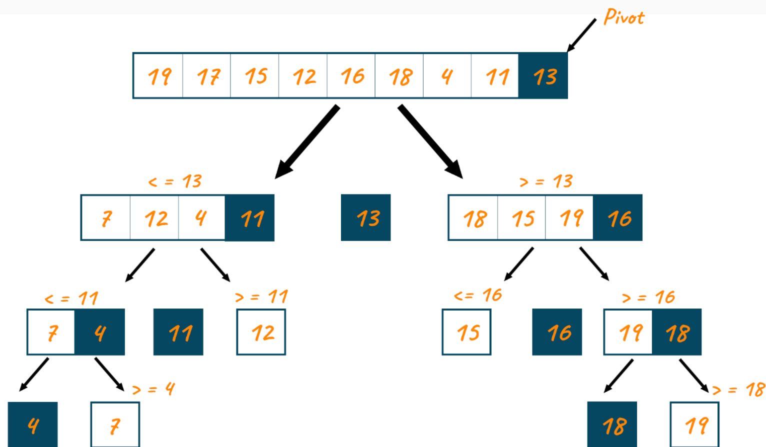
Estratégias melhores

- **Elemento do meio** — razoável
- **Elemento aleatório** — provadamente $O(N \log N)$ esperado

Melhor estratégia provada

Selecionar o pivô **aleatoriamente** garante um tempo esperado de $O(N \log N)$. A função `particionaRandom` sorteia um índice, troca com o último elemento e chama `particiona` normalmente.

Quick Sort — Visualização



Quick Sort — Pseudocódigo (parte 1)

Algorithm 1 `void troca(int vet[], int i, int j)`

1: $aux = vet[i]$
2: $vet[i] = vet[j]$
3: $vet[j] = aux$

Algorithm 2 `int particiona(int vet[], int inicio, int fim)`

1: $pivo = vet[fim]$
2: $pivoIdx = inicio$
3: **for** $i = inicio$ **to** fim **do**
4:     **if** $vet[i] \leq pivo$ **then**
5:         troca$(vet, i, pivoIdx)$
6:         $pivoIdx{+}{+}$
7:     **end if**
8: **end for**
9: troca$(vet, pivoIdx, fim)$
10: **return** $pivoIdx$

Quick Sort — Pseudocódigo (parte 2)

Algorithm 3 `int particionaRandom(int vet[], int inicio, int fim)`

1: $pivoIdx = (rand\%(fim - inicio + 1)) + inicio$
2: troca($vet, pivoIdx, fim$)
3: **return** particiona($vet, inicio, fim$)

Algorithm 4 `void quickSort(int vet[], int inicio, int fim)`

1: **if** $inicio < fim$ **then**
2: $p = particionaRandom(vet, inicio, fim)$
3: quickSort($vet, inicio, p - 1$)
4: quickSort($vet, p + 1, fim$)
5: **end if**

Quick Sort — Código em C

```c
5  void quick(int vet[], int esq, int dir){
6      int pivo = esq, i,ch,j;
7      for(i=esq+1;i<=dir;i++){
8          j = i;
9          if(vet[j] < vet[pivo]){
0              ch = vet[j];
1              while(j > pivo){
2                  vet[j] = vet[j-1];
3                  j--;
4              }
5              vet[j] = ch;
6              pivo++;
7          }
8      }
9      if(pivo-1 >= esq){
0          quick(vet,esq,pivo-1);
1      }
2      if(pivo+1 <= dir){
3          quick(vet,pivo+1,dir);
4      }
5  }
```

Heap Sort

Ordenação por Árvore Binária

Heap Sort — Conceito

▪ Visão geral

Algoritmo descoberto por **J.W.J. Williams** em 1964. Usa a estrutura de **heap** (árvore binária) para ordenar. Tem custo $O(N \log N)$ **mesmo no pior caso**.

▪ Pré-requisito

O Heap Sort será estudado em detalhes quando aprendermos **árvores binárias**. Por ora, vejamos a ideia central.

▪ Índices como árvore

Para qualquer vetor $v[1 \mathinner{..} m]$, os índices formam uma árvore binária:

- Raiz: índice 1
- Pai de f: $\lfloor f/2 \rfloor$
- Filho esquerdo de p: $2p$
- Filho direito de p: $2p+1$

▪ Complexidade

Todos os casos: $O(N \log N)$ Espaço: $O(1)$

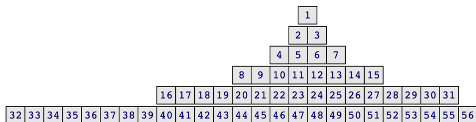
Vídeo: `https://youtu.be/Xw2D9aJRBY4`

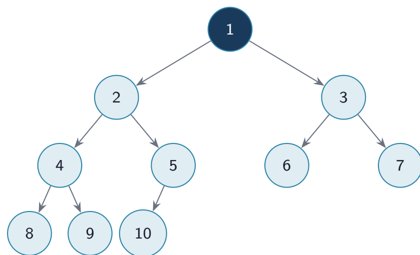
Simulação: `https://cs.usfca.edu/~galles/visualization/HeapSort.html`

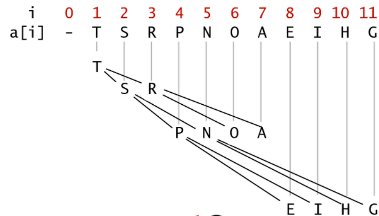
Representação em camadas

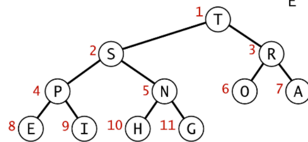
O vetor pode ser desenhado em camadas — cada filho fica na camada seguinte ao pai. Cada camada tem o **dobro** de elementos da anterior.

O número de camadas de $v[1 \mathinner{..} m]$ é exatamente $1 + \lfloor \log m \rfloor$.









Heap representations

Heap Sort — Peneira e Código

```c
void peneira(int *vet, int raiz, int fundo) {
    int pronto, filhoMax, tmp;

    pronto = 0;
    while ((raiz*2 <= fundo) && (!pronto)) {
        if (raiz*2 == fundo) {
            filhoMax = raiz * 2;
        }
        else if (vet[raiz * 2] > vet[raiz * 2 + 1]) {
            filhoMax = raiz * 2;
        }
        else {
            filhoMax = raiz * 2 + 1;
        }

    if (vet[raiz] < vet[filhoMax]) {
        tmp = vet[raiz];
        vet[raiz] = vet[filhoMax];
        vet[filhoMax] = tmp;
        raiz = filhoMax;
    }
    else {
      pronto = 1;
    }
  }
}
```

```c
void heapsort(int *vet, int n) {
    int i, tmp;

    for (i = (n / 2); i >= 0; i--) {
        peneira(vet, i, n - 1);
    }

    for (i = n-1; i >= 1; i--) {
        tmp = vet[0];
        vet[0] = vet[i];
        vet[i] = tmp;
        peneira(vet, 0, i-1);
    }
}
```

Heap Sort — Programa Principal

```
int main() {
    int vetor[max], i;

    for(i = 0; i < max; i++){
        printf("Digite o %d° elemento: ",i+1);
        scanf("%d",&vetor[i]);
    }

    heapsort(vetor,max);
    for (i = 0; i < max; i++) {
        printf("%d ", vetor[i]);
    }
    return(0);
}
```

Referências I

CORMEN, T. H. et al. *Introduction to Algorithms*. 2nd. ed. [S.l.]: The MIT Press, 2009. ISBN 0262032937.