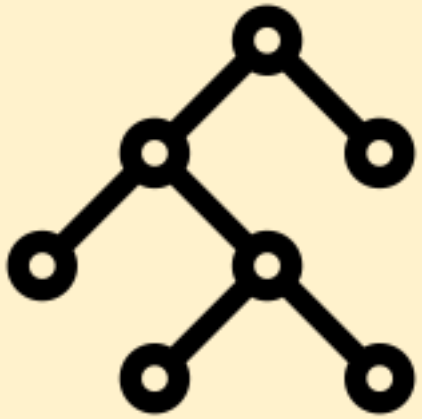
## Algoritmo de ordenação

# Heapsort



**Universidade Federal do Pará**
**Faculdade de Computação**
**Estruturas de Dados**

**Lucas Maués de Menezes**

# História

Considerado generalista e eficiente, o Heapsort faz parte da família de algoritmos de **ordenação por seleção**.

Utiliza a **estrutura de dados heap**.

Desenvolvido por J. W. J. Williams em **1964** e posteriormente refinado por R. W. Floyd, ambos cientistas da computação.

**Williams**

**Floyd**

# Funcionamento do Heapsort

1. **Construir um heap máximo** a partir do array de elementos a serem ordenados

2. **Extrair o elemento máximo** (raiz do heap) e colocar na última posição do array

3. Reduzir o tamanho do heap em 1 e ajustar a propriedade do heap

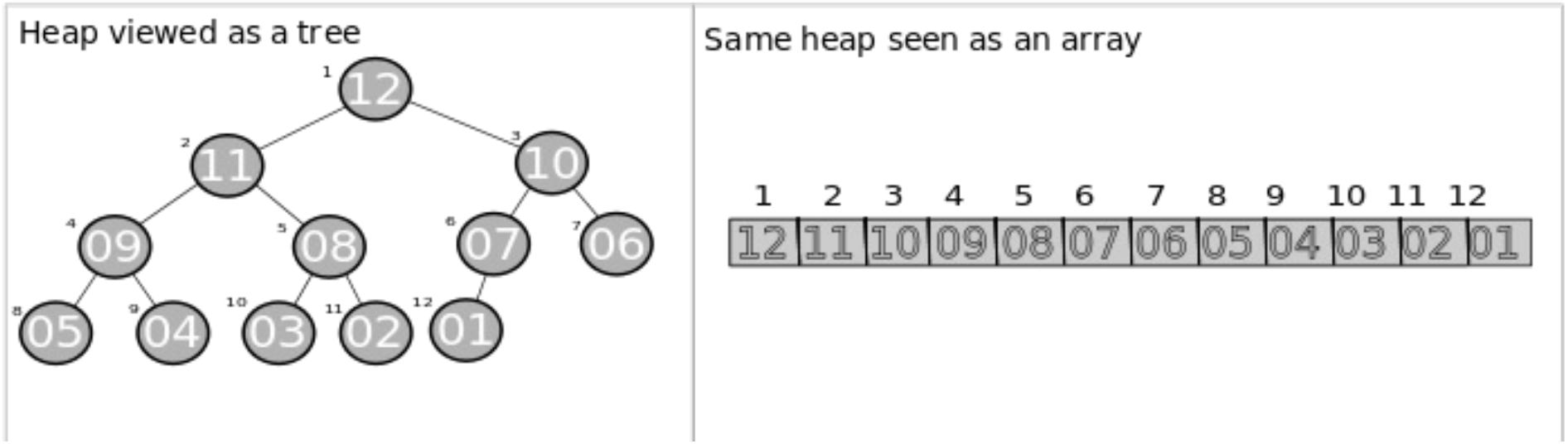
* Repetir 2 e 3 até que o array esteja ordenado.

**Heap** é uma “árvore binária especial” = Estrutura Binária + Propriedade de Heap.

Implementação **Max-Heap**:
O valor do nó pai é sempre maior ou igual dos filhos. O nó raiz contém o maior valor da árvore. Ordenará os elementos em ordem decrescente.

# Representação do Heapsort

**Heap**




**Árvore**

**Array**

# Complexidade do Heapsort

$$O(n \log n)$$

$O$ = notação assintótica Big O

$n$ = tamanho do conjunto de dados

$log\ n$ = logaritmo na base 2 de $n$

**Heapsort tem a mesma taxa de crescimento do tempo de execução** em relação ao tamanho da entrada (*n*) para melhor, pior e médio caso.

Ou seja, se a entrada de elementos (*n*) aumenta, o tempo de execução do algoritmo cresce proporcionalmente a *n* multiplicado pelo logaritmo de *n (2 x log n).*

**Não importa** se os dados estão desordenados, parcialmente ordenados ou já ordenados, o desempenho do Heapsort permanece inalterado, resultando sempre em uma complexidade de tempo *O*(*n* log *n*).

# Complexidade do Heapsort

**Algoritmos O(n log n) são eficientes para grandes conjuntos de dados** em comparação aos com complexidade quadrática $O(n^2)$.

Heapsort é utilizado em sistemas operacionais no **gerenciamento de memória** e na promoção de **filas de prioridade**, pois possui desempenho em tempo de execução satisfatório em sequências aleatoriamente desordenadas, utiliza pouca memória e o desempenho no pior cenário é praticamente igual ao médio.

* Muitos algoritmos de ordenação tem desempenho insatisfatório no pior caso, tanto no tempo de execução, quanto no uso de memória, por exemplo, BubbleSort e SelectionSort, que possuem ordem de complexidade $O(n^2)$.

# Exemplo de código em C

**Repositório código fonte:** https://github.com/lucasm/estruturas-de-dados-c/blob/main/atividade-3/heapsort.c

**heapify()** – cria heap máximo

**heapSort()** – ordena

**printArray()**

**main()**

# Exemplo de código em C

**main()**

```c
int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    // cria um array de inteiros

    int n = sizeof(arr) / sizeof(arr[0]);
    // cacula a quantidade de elementos do array

    printf("Array original:\n");
    printArray(arr, n);

    heapSort(arr, n);

    printf("Array ordenado:\n");
    printArray(arr, n);

    return 0;
}
```

# Exemplo de código em C

**`printArray()`**

```c
// função imprimir array
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; ++i)
        printf("%d ", arr[i]);
    printf("\n");
}
```

# Exemplo de código em C

**heapSort()**

```c
// função ordenar array usando heapsort
void heapSort(int arr[], int n)
{
  // constrói o heap
  for (int i = n / 2 - 1; i >= 0; i--)
  {
    heapify(arr, n, i); // chama a função heapify para cada nó não folha
  }

  // extrai um elemento por vez do heap
  for (int i = n - 1; i > 0; i--)
  {
    int temp = arr[0];  // move a raiz atual para o final
    arr[0] = arr[i];    // troca o primeiro elemento com o último
    arr[i] = temp;      // coloca o maior elemento na parte de trás
    heapify(arr, i, 0); // chama a função heapify na heap reduzida
  }
}
```

# Exemplo de código em C

**heapify()**

```c
// função criar um heap máximo
void heapify(int arr[], int n, int i)
{
  // inicializa o maior como raiz do heap
  int largest = i;       // raiz
  int left = 2 * i + 1;  // filho da esquerda
  int right = 2 * i + 2; // filho da direita

  // filho esquerda > raiz atual
  if (left < n && arr[left] > arr[largest])
  {
    largest = left; // troca raiz
  }

  // filho direita > raiz atual
  if (right < n && arr[right] > arr[largest])
  {
    largest = right; // troca raiz
  }

  // maior não for a raiz atual
  if (largest != i)
  {
    int temp = arr[i]; // troca a raiz com o maior
    arr[i] = arr[largest];
    arr[largest] = temp;
    heapify(arr, n, largest); // chama recursivamente para a subárvore afeta
  }
}
```

# Exemplo de código em C

```
Array original:
12 11 13 5 6 7

Array ordenado:
5 6 7 11 12 13
```

```c
#include <stdio.h>

// função criar um heap máximo
void heapify(int arr[], int n, int i)…

// função ordenar array usando heapsort
void heapSort(int arr[], int n)…

// função imprimir array
void printArray(int arr[], int n)…

int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    // cria um array de inteiros

    int n = sizeof(arr) / sizeof(arr[0]);
    // cacula a quantidade de elementos do array

    printf("Array original:\n");
    printArray(arr, n);

    heapSort(arr, n);

    printf("Array ordenado:\n");
    printArray(arr, n);

    return 0;
}
```

# Conclusão

- **Heapsort é um algoritmo de ordenação eficiente**
- Usa a estrutura de dados heap, muito útil quando é necessário remover repetidamente o elemento com a prioridade mais alta (ou mais baixa)
- Apresenta um desempenho consistente de O(n log n) no pior e médio caso
- Não é estável

# Vídeo

Link vídeo: Uma forma divertida de ver o algoritmo em ação, formação da Heap e o funcionamento do Heapsort

# Referências


DSC UFCG. **História da Computação - O Algoritmo Heapsort**. Disponível em: http://www.dsc.ufcg.edu.br/~pet/jornal/maio2013/materias/historia_da_computacao.html.

CTC INE UFSC. **Ordenação de Dados III**. Disponível em: https://www.inf.ufsc.br/~r.mello/ine5384/17-OrdenacaoDados3.pdf.

DCC UFMG. **Ordenação: Heapsort**. Disponível em: https://homepages.dcc.ufmg.br/~cunha/teaching/20121/aeds2/heapsort.pdf.

Moodle USP. **Vídeo interativo - Heapsort (parte 2): algoritmo de ordenação.** Disponível em: https://edisciplinas.usp.br/mod/hvp/view.php?id=3264206.

Acessos em: 12 de dezembro de 2023.