



Estrutura de Dados I

Heapsort

Bruno Prado

Departamento de Computação / UFS

Introdução

- ▶ O que é Heapsort?
 - ▶ Criado por J. W. J. Williams em 1964
 - ▶ Estratégia de Transformação e Conquista
 - ▶ Converte os dados para uma estrutura de Heap



Introdução

- ▶ Vantagens
 - ▶ In-place
 - ▶ Sem necessidade de espaço adicional
 - ▶ Eficiência algorítmica
 - ▶ Complexidade $O(n \log_2 n)$
 - ▶ Aplicações de tempo real
 - ▶ Ordem exata de execução

Introdução

- ▶ Desvantagens

- ▶ Desempenho

- ▶ Mais lento que outros algoritmos mais flexíveis
 - ▶ Não explora o princípio da localidade

- ▶ Paralelismo

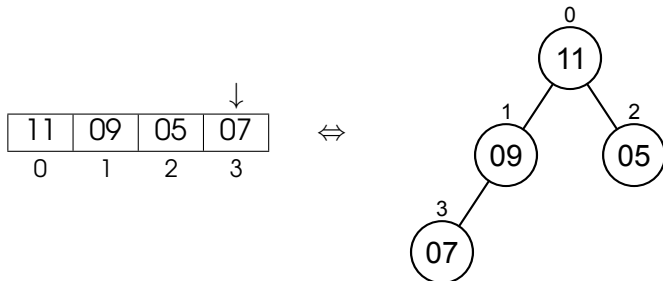
- ▶ Não é tão facilmente paralelizável

Heapsort

- ▶ Conceitos chave
 - ▶ Heap mínimo ou máximo
 - ▶ Construção da estrutura de Heap
 - ▶ Manutenção da propriedade do Heap
 - ▶ Não é estável

Árvores Heap

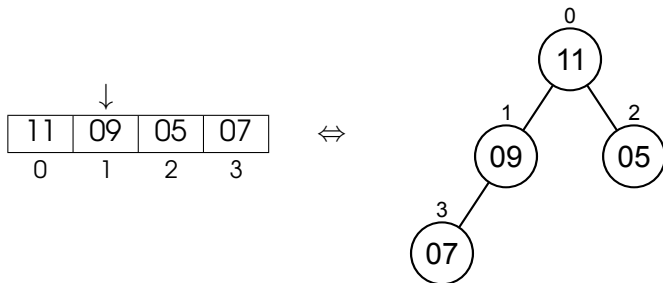
- Armazenamento e indexação
 - Nó pai



$$Pai(i) = \frac{i-1}{2}$$

Árvores Heap

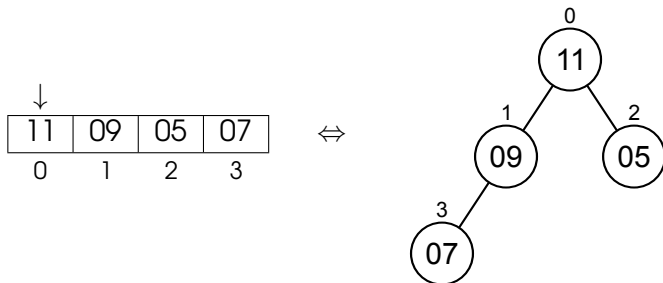
- ▶ Armazenamento e indexação
 - ▶ Nó filho esquerdo



$$\text{Esquerdo}(i) = 2i + 1$$

Árvores Heap

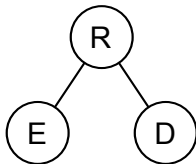
- ▶ Armazenamento e indexação dos nós
 - ▶ Nó filho direito



$$\text{Direito}(i) = 2i + 2$$

Árvores Heap

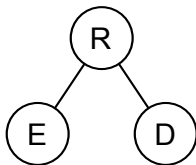
- ▶ Tipos de árvores heap
 - ▶ Heap mínimo



Propriedade $R \leq E$ e $R \leq D$

Árvores Heap

- ▶ Tipos de árvores heap
 - ▶ Heap máximo



Propriedade $R \geq E$ e $R \geq D$

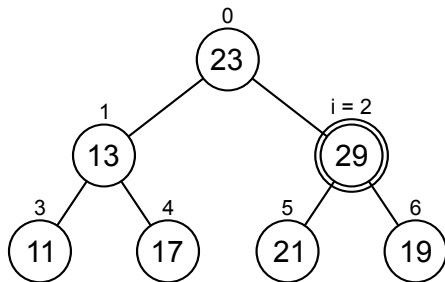
Árvores Heap

- ▶ Manutenção da propriedade de heap
 - ▶ Procedimento heapify

```
void heapify(int V[], int i, int n) {  
    unsigned int P = i;  
    unsigned int E = esquerdo(i);  
    unsigned int D = direito(i);  
    if(E < n && V(E) > V(P)) P = E;  
    if(D < n && V(D) > V(P)) P = D;  
    if(P != i) {  
        trocar(&V(P), &V(i));  
        heapify(V, P, n);  
    }  
}
```

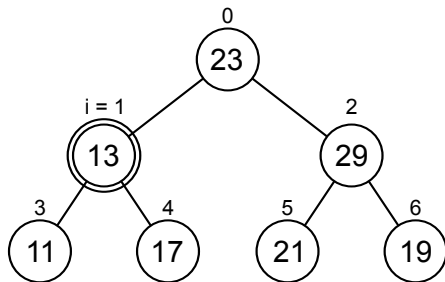
Heapsort

- ▶ Princípio de funcionamento
 - ▶ Etapa de transformação



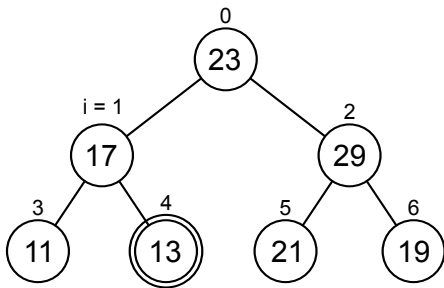
Heapsort

- ▶ Princípio de funcionamento
 - ▶ Etapa de transformação



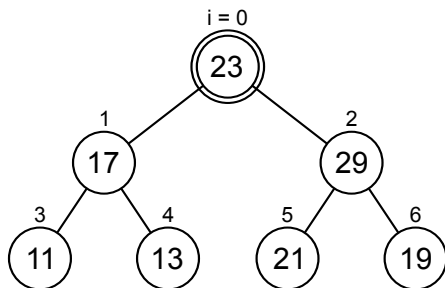
Heapsort

- ▶ Princípio de funcionamento
 - ▶ Etapa de transformação



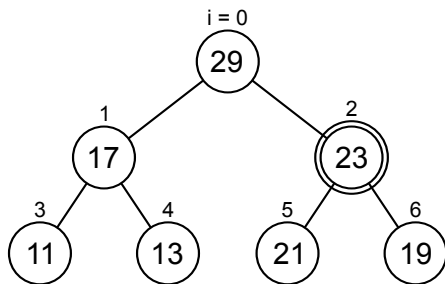
Heapsort

- ▶ Princípio de funcionamento
 - ▶ Etapa de transformação



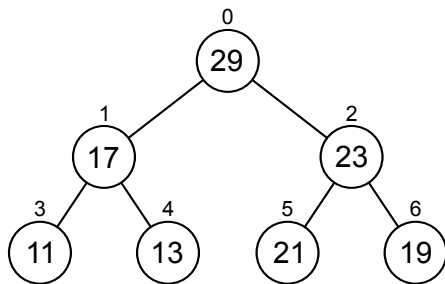
Heapsort

- ▶ Princípio de funcionamento
 - ▶ Etapa de transformação



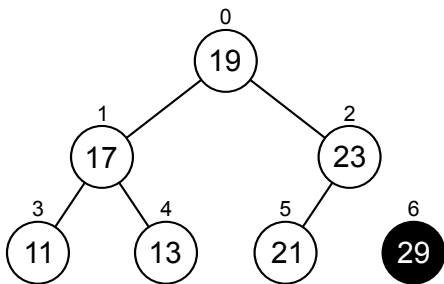
Heapsort

- ▶ Dados estruturados
 - ▶ Heap máximo



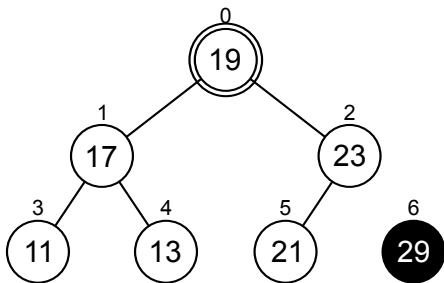
Heapsort

- ▶ Princípio de funcionamento
 - ▶ Etapa de conquista



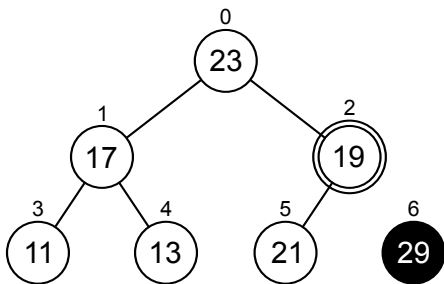
Heapsort

- ▶ Princípio de funcionamento
 - ▶ Etapa de conquista



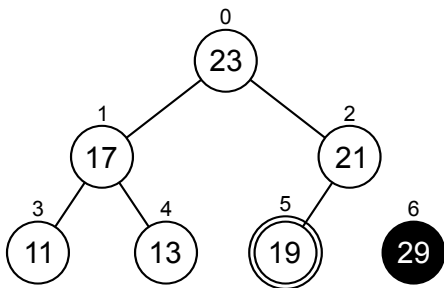
Heapsort

- ▶ Princípio de funcionamento
 - ▶ Etapa de conquista



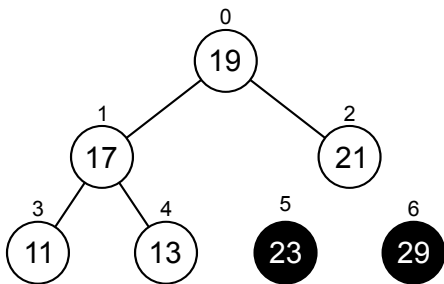
Heapsort

- ▶ Princípio de funcionamento
 - ▶ Etapa de conquista



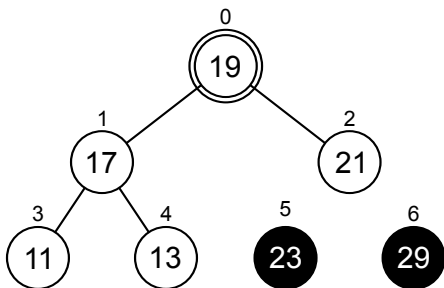
Heapsort

- ▶ Princípio de funcionamento
 - ▶ Etapa de conquista



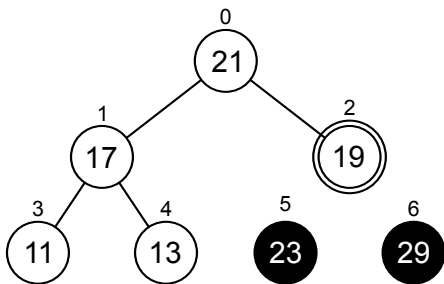
Heapsort

- ▶ Princípio de funcionamento
 - ▶ Etapa de conquista



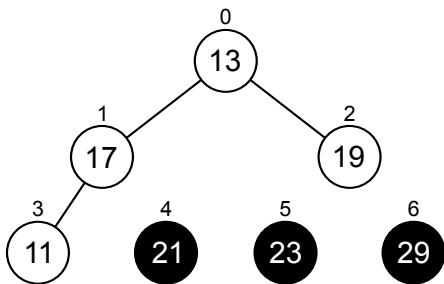
Heapsort

- ▶ Princípio de funcionamento
 - ▶ Etapa de conquista



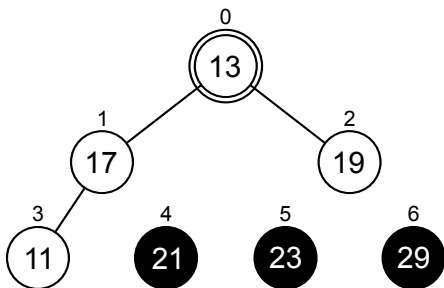
Heapsort

- ▶ Princípio de funcionamento
 - ▶ Etapa de conquista



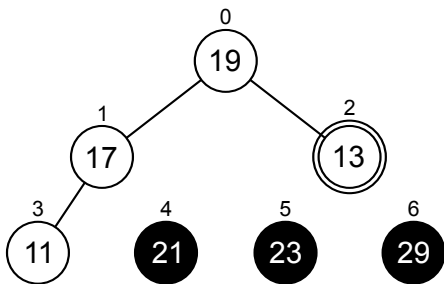
Heapsort

- ▶ Princípio de funcionamento
 - ▶ Etapa de conquista



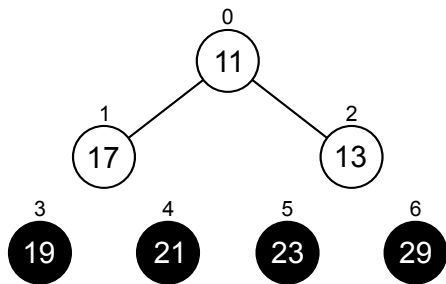
Heapsort

- ▶ Princípio de funcionamento
 - ▶ Etapa de conquista



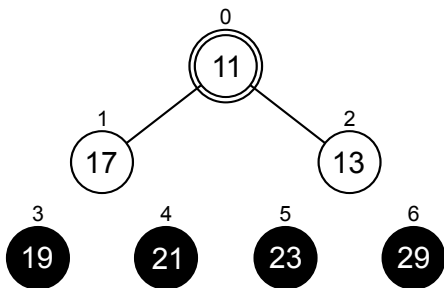
Heapsort

- ▶ Princípio de funcionamento
 - ▶ Etapa de conquista



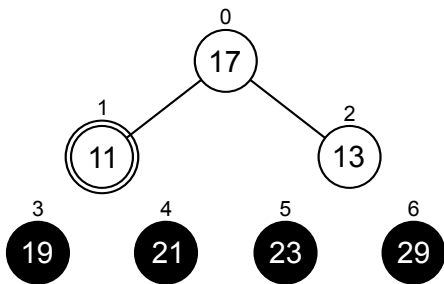
Heapsort

- ▶ Princípio de funcionamento
 - ▶ Etapa de conquista



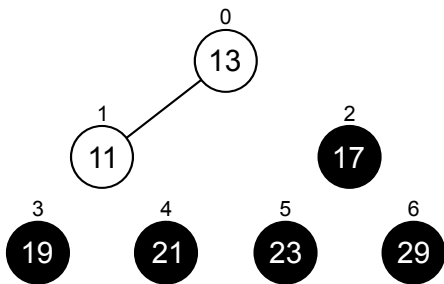
Heapsort

- ▶ Princípio de funcionamento
 - ▶ Etapa de conquista



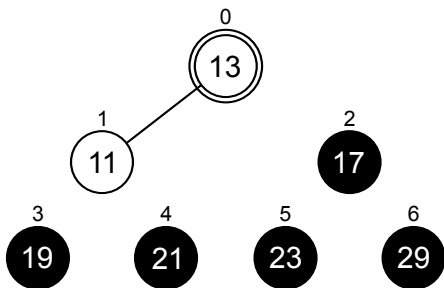
Heapsort

- ▶ Princípio de funcionamento
 - ▶ Etapa de conquista



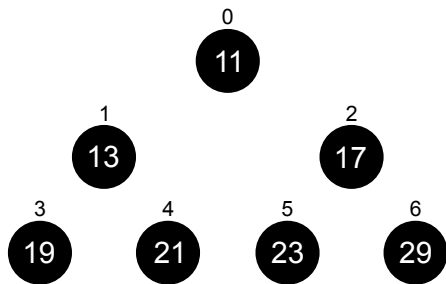
Heapsort

- ▶ Princípio de funcionamento
 - ▶ Etapa de conquista



Heapsort

- ▶ Princípio de funcionamento
 - ▶ Etapa de conquista



Heapsort

- ▶ Implementação em C
 - ▶ Procedimento heapsort

```
void heapsort(int V[], int n) {  
    construir_heap(V, n);  
    int i;  
    for(i = n - 1; i > 0; i--) {  
        trocar(&V[0], &V[i]);  
        heapify(V, 0, i);  
    }  
}
```

Heapsort

- ▶ Análise de complexidade
 - ▶ Espaço $O(1)$
 - ▶ Implementação iterativa
 - ▶ Sem alocação de espaço extra
 - ▶ Tempo $O(n \log_2 n)$
 - ▶ Construção é $O(n)$
 - ▶ Heapify é $O(\log_2 n)$

Exemplo

- ▶ Considerando o algoritmo Heapsort, realize a ordenação decrescente do vetor
 - ▶ Sequência 32, 54, 92, 74, 23, 3, 43, 63
 - ▶ Utilize o heap mínimo e máximo
 - ▶ Execute o algoritmo passo a passo

Exercício

- ▶ A empresa de telecomunicações Poxim Tech está construindo um sistema de comunicação, baseado no protocolo de datagrama do usuário (UDP) para transferência de pacotes em redes TCP/IP
 - ▶ Os dados são organizados em sequências de bytes em formato hexadecimal de tamanho variável, mas limitados até o tamanho máximo de 512 bytes
 - ▶ Devido às características de roteamento de redes TCP/IP, os pacotes podem chegar ao seu destino desordenados, sendo necessária a ordenação dos pacotes para reconstruir os dados corretamente
 - ▶ Para permitir o acesso rápido dos dados, é possível processar as informações recebidas desde que estejam parcialmente ordenadas, com os pacotes iniciais, sendo este processamento disparado por uma determinada quantidade de pacotes recebidas

Exercício

► Formato de arquivo de entrada

- [#n total de pacotes] [Quantidade de pacotes]
- [Número do pacote] [#m₁ Tamanho do pacote] [B1] ... [B_{m₁}]
- ...
- [Número do pacote] [#m_n Tamanho do pacote] [B1] ... [B_{m_n}]

```
7 2
1 2 04 05
2 4 06 07 08 09
0 3 01 02 03
5 6 11 12 13 14 15 16
3 5 0A 0B 0C 0D 0E
6 3 AA BB CC
4 2 0F 10
```

Exercício

- ▶ Formato de arquivo de saída
 - ▶ Quando uma quantidade determinada de pacotes é recebida, é feita a ordenação parcial dos pacotes para verificar se é possível exibir a parte inicial completa dos dados que já foram recebidos

[0]	01	02	03	04	05	06	07	08	09
[1]	0A	0B	0C	0D	0E				
[2]	0F	10	11	12	13	14	15	16	AA BB CC