

Técnicas de Busca e Ordenação (TBO)

Heap Sort

Departamento de Informática (DI)
Centro Tecnológico (CT)
Universidade Federal do Espírito Santo (UFES)

(Material baseado nos slides dos Professores Eduardo Zambon e Giovanni Comarela)

- **Aula de hoje:** Algoritmo de ordenação *heap sort*.

Referências

Chapter 9 – Priority Queues and Heapsort

R. Sedgewick

Revisão – O que é um heap?

Desafio – De heap para vetor ordenado

Heap Sort – Intuição

Heap sort

Heap sort: novo algoritmo de ordenação baseado em fila com prioridade.

```
void sort(Item *a, int lo, int hi) {  
    int N = hi - lo + 1;  
    PQ_init(N);  
    for (int i = 0; i < N; i++) {  
        PQ_insert(a[i]);  
    }  
    for (int i = N-1; i >= 0; i--) {  
        a[i] = PQ_delmax();  
    }  
}
```

Propriedades:

- **Pior caso:** ordem de $N \log N$ comparações.
- **In-place?** Não, mas é possível melhorar. (Veja adiante.)
- **Estável?** Não. *Heap* pode embaralhar chaves iguais.

Heap sort in-place “top-down”

Ideia geral para *sort in-place*:

- Entrada é um *array* com N chaves e ordenação arbitrária.
- **Suposição**: entradas indexadas de 1 a N .
- (Possível começar de 0, feito somente por conveniência.)
- **Construção do heap**: usa um método *top-down* para construir um *max-heap* com as N chaves.
- **Sort down**: *loop* simples que joga o máximo para o fim do *array*.

Heap sort in-place “top-down”

```
void sort(Item *a, int lo, int hi) {  
    int N = hi - lo + 1;  
    for (int i = lo; i <= hi; i++)  
        fix_up(a, i);    // "Top-down" heap construction.  
    while (N > 1) { // Sort down.  
        exch(a[1], a[N]);  
        fix_down(a, --N, 1);  
    }  
}
```

4 8 6 3 1 7 9 2 0 5
4|8 6 3 1 7 9 2 0 5
8 4|6 3 1 7 9 2 0 5
8 4 6|3 1 7 9 2 0 5
8 4 6 3|1 7 9 2 0 5
8 4 6 3 1|7 9 2 0 5
8 4 7 3 1 6|9 2 0 5
9 4 8 3 1 6 7|2 0 5
9 4 8 3 1 6 7 2|0 5
9 4 8 3 1 6 7 2 0|5
9 5 8 3 4 6 7 2 0 1|
0 1 2 3 4 5 6 7 8 9

Invariante da construção:

- A cada passo do *loop* o *heap* cresce uma posição.
- Posição *i* é o novo elemento.
- Chaves à esquerda de *i* formam o *heap*.
- Chaves à direita de *i* ainda não foram vistas.

Heap sort in-place “top-down”

Número de comparações no pior caso:

- Construção do *heap*: N chamadas de `fix_up`. Custo total:

$$N(1 + \lg N)$$

- total:

$$2N \lg N$$

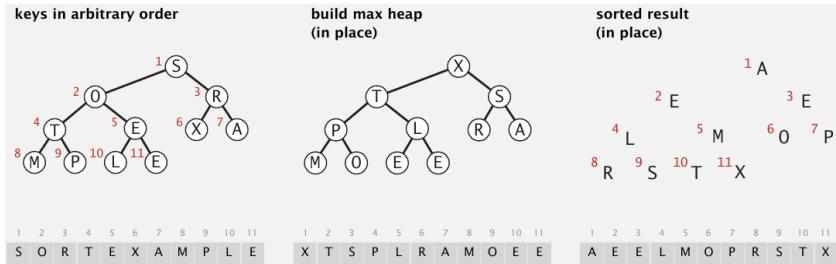


Próxima versão é a equivalente “*bottom-up*”.

Heap sort in-place (“bottom-up”)

Ideia geral para *sort in-place*:

- Considera *array* de entrada como uma **árvore binária completa**.
- **Construção do heap**: Constrói um **max-heap** com as N chaves.
- **Sort down**: remove repetidamente a chave maior.



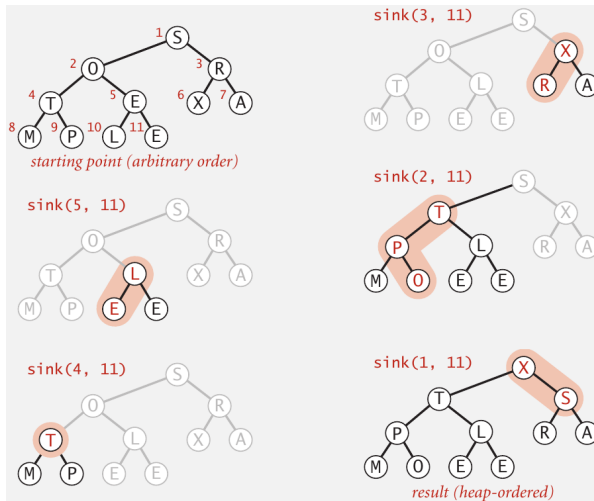
Heap sort demo

- Entrada é um *array* com ordenação arbitrária.
- **Suposição**: entradas indexadas de 1 a N .
- (Possível começar de 0, feito somente para ficar compatível com as figuras.)
- **Construção do heap**: usa um método *bottom-up*.
- Invariante da construção é **indutivo**: assume que *heaps* menores já foram construídos para construir um *heap* maior.
- **Sort down**: *loop* simples que joga o máximo para o fim do *array*.

Ver arquivo 24DemoHeapsort.mov.

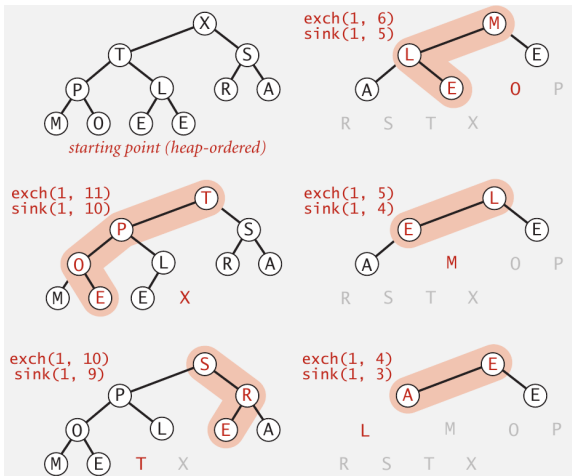
Heap sort - 1a. passada: construção do heap

```
for (int k = N/2; k >= 1; k--)  
    fix_down(a, N, k); // "Bottom-up" heap construction.
```



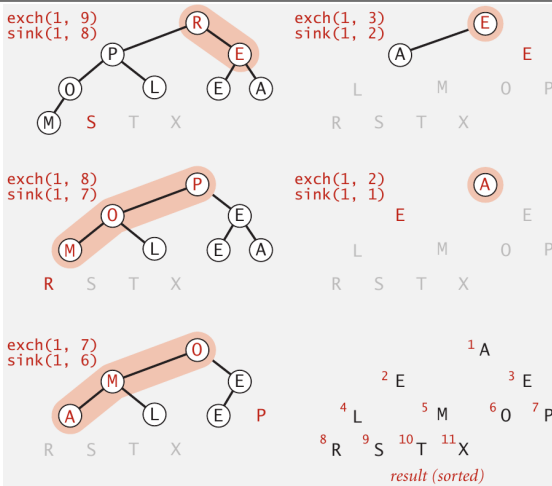
Heap sort - 2a. passada: sort down

```
while (N > 1) { // Sort down.  
    exch(a[1], a[N]);  
    fix_down(a, --N, 1); }
```



Heap sort - 2a. passada: sort down

```
while (N > 1) { // Sort down.  
    exch(a[1], a[N]);  
    fix_down(a, --N, 1); }
```



Heap sort: implementação em C

```
void sort(Item *a, int lo, int hi) {  
    int N = hi - lo + 1;  
    for (int k = N/2; k >= 1; k--)  
        fix_down(a, N, k); // "Bottom-up" heap construction.  
    while (N > 1) { // Sort down.  
        exch(a[1], a[N]);  
        fix_down(a, --N, 1); }  
}
```

In-place: ✓

Estável: ✗

Heap sort: trace

		a[i]											
N	k	0	1	2	3	4	5	6	7	8	9	10	11
initial values			S	O	R	T	E	X	A	M	P	L	E
11	5		S	O	R	T	L	X	A	M	P	E	E
11	4		S	O	R	T	L	X	A	M	P	E	E
11	3		S	O	X	T	L	R	A	M	P	E	E
11	2		S	T	X	P	L	R	A	M	O	E	E
11	1		X	T	S	P	L	R	A	M	O	E	E
heap-ordered			X	T	S	P	L	R	A	M	O	E	E
10	1		T	P	S	O	L	R	A	M	E	E	X
9	1		S	P	R	O	L	E	A	M	E	T	X
8	1		R	P	E	O	L	E	A	M	S	T	X
7	1		P	O	E	M	L	E	A	R	S	T	X
6	1		O	M	E	A	L	E	P	R	S	T	X
5	1		M	L	E	A	E	O	P	R	S	T	X
4	1		L	E	E	A	M	O	P	R	S	T	X
3	1		E	A	E	L	M	O	P	R	S	T	X
2	1		E	A	E	L	M	O	P	R	S	T	X
1	1		A	E	E	L	M	O	P	R	S	T	X
sorted result			A	E	E	L	M	O	P	R	S	T	X
Heapsort trace (array contents just after each sink)													

Heap sort: análise

Ordem de crescimento no pior caso (versão “*bottom-up*”):

- Construção do *heap*: # comparações é $N/2(2 \lg N)$.
- *Sort down*: # comparações é $2N \lg N$.
- \Rightarrow Pior caso é $\sim 3N \lg N$.

Significância: nenhum outro algoritmo *in-place* tem esse desempenho.

- *Merge sort*: espaço extra **linear**.
(*Merge sort in-place* é possível mas não é prático.)
- *Quick sort*: **quadrático** no pior caso. (*Quick sort* com pior caso $N \lg N$ é possível mas não é prático.)
- *Heap sort*: algoritmo ótimo para tempo e espaço!

Na prática:

- Não é estável.
- Não usa bem o cache.
- Menos eficiente que *quick sort* na maioria da vezes.

Objetivos:

- Algoritmo tão rápido quanto *quick sort* na prática.
- Com ordem de crescimento $N \log N$ no pior caso.
- *In-place*.

Intro sort:

- Executa *quick sort*.
- *Cut-off* para *heap sort* se tamanho pilha exceder $2 \lg N$.
- *Cut-off* para *insertion sort* quando $N = 16$.

Amplamente utilizado na prática:

- C++ STL.
- .NET Framework.

Algoritmos de ordenação: sumário

	inplace?	stable?	best	average	worst	remarks
selection	✓		$\frac{1}{2} n^2$	$\frac{1}{2} n^2$	$\frac{1}{2} n^2$	n exchanges
insertion	✓	✓	n	$\frac{1}{4} n^2$	$\frac{1}{2} n^2$	use for small n or partially ordered
shell	✓		$n \log_3 n$?	$c n^{3/2}$	tight code; subquadratic
merge		✓	$\frac{1}{2} n \lg n$	$n \lg n$	$n \lg n$	$n \log n$ guarantee; stable
timsort		✓	n	$n \lg n$	$n \lg n$	improves mergesort when preexisting order
quick	✓		$n \lg n$	$2 n \ln n$	$\frac{1}{2} n^2$	$n \log n$ probabilistic guarantee; fastest in practice
3-way quick	✓		n	$2 n \ln n$	$\frac{1}{2} n^2$	improves quicksort when duplicate keys
heap	✓		$3 n$	$2 n \lg n$	$2 n \lg n$	$n \log n$ guarantee; in-place
?	✓	✓	n	$n \lg n$	$n \lg n$	holy sorting grail