HeapSort

Ciência da Computação
Laboratório de Ordenação e Pesquisa
Prof. M.Sc. Elias Gonçalves

HeapSort

→ Utiliza a estrutura da dados Heap para ordenar.

→ Heap é um vetor que simula uma árvore binária completa (exceto o último nível).

→ Cada elemento pai terá dois elementos filhos.

→ Quando o pai é maior ou igual aos filhos – Heap máximo.

→ Quando o pai é sempre menor que os filhos – Heap mínimo.

→ Coloca na raiz o maior número e troca com o último elemento do vetor a cada passo.

→ Seu tempo de execução assintótico é O(n log n);

Definindo o pai e os filhos

→ Seja o elemento do índice **i** identificado como pai;

→ O elemento do índice **((2*i)+1)** será o filho da esquerda;

→ O elemento do índice **((2*i)+2)** será o filho da direita.

153	**30**	**92**	**25**	**2**	**98**	**13**	vetor v
0	1	2	3	4	5	6	índice i

Construir o Heap

153	30	92	25	2	98	13	vetor v
0	1	2	3	4	5	6	índice i

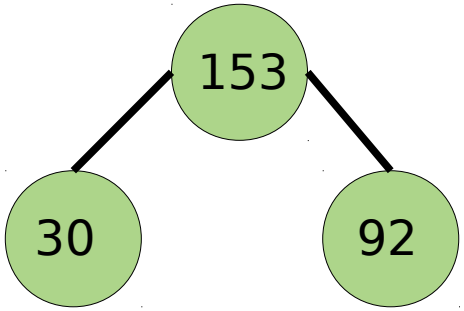
→ Seja **i** o pai (i = 0, então **153** é o primeiro pai – raiz);

→ Sabendo que o índice 0 é o pai ao aplicar **((2*i)+1)** o primeiro filho será:

2*0+1 = 0+1 = 1, então **30** é o primeiro filho (esq.);

→ Sabendo que o índice 0 é o pai ao aplicar **((2*i)+2)** o segundo filho será:

2*0+2 = 0+2 = 2, então **92** é o segundo filho (dir.);



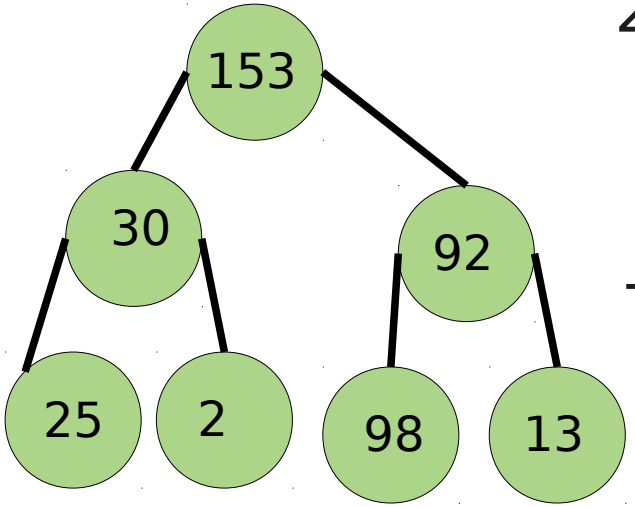
Construir o Heap

153	30	92	25	2	98	13	vetor v
0	1	2	3	4	5	6	índice i

→ Para os cada novo pai repete-se o passo anterior até o vetor todo for visualizado como uma árvore binária.

→ Pai: **i = 1**, então **30** é o pai dessa família;

→ Índice 1 é o pai. Então ao aplicar **((2*i)+1)** tem-se:



2*1+1 = 2+1 = 3, então **25** é o filho (esq.);

2*1+2 = 2+2 = 4, então **2** é o filho (dir.);

→ Pai: **i = 2**, então **92** é o pai dessa família;

→ Filho: 2*2+1 = 5, então **98** é o filho esq.

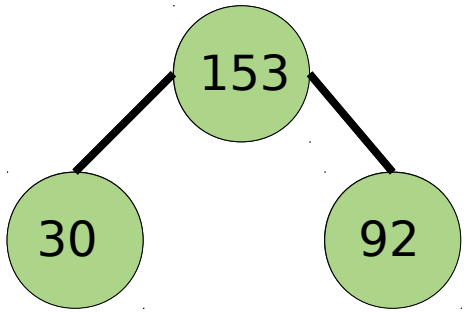
→ Filho: 2*2+2 = 6, então **13** é o filho dir.

Construir o Heap (Código)

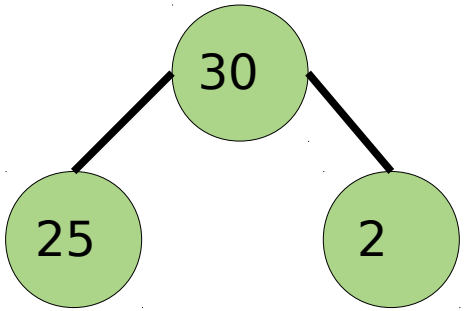
153	30	92	25	2	98	13	vetor v
0	1	2	3	4	5	6	índice i

```
// Cria as sub-árvores tendo i como raiz
void criarHeap( int v[], int n, int i ) {

    // Para o heap máximo o pai sempre tem que ser o maior
    int pai = i;
    int esq = 2*i + 1;
    int dir = 2*i + 2;
```

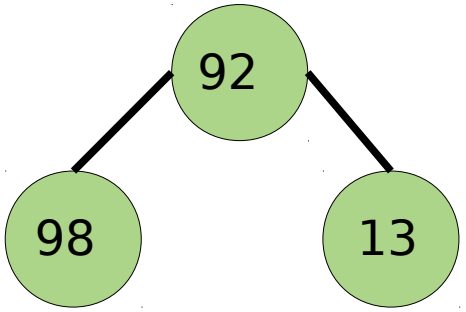
i = 0



i = 1



i = 2



Transformar o Heap em Heap máximo

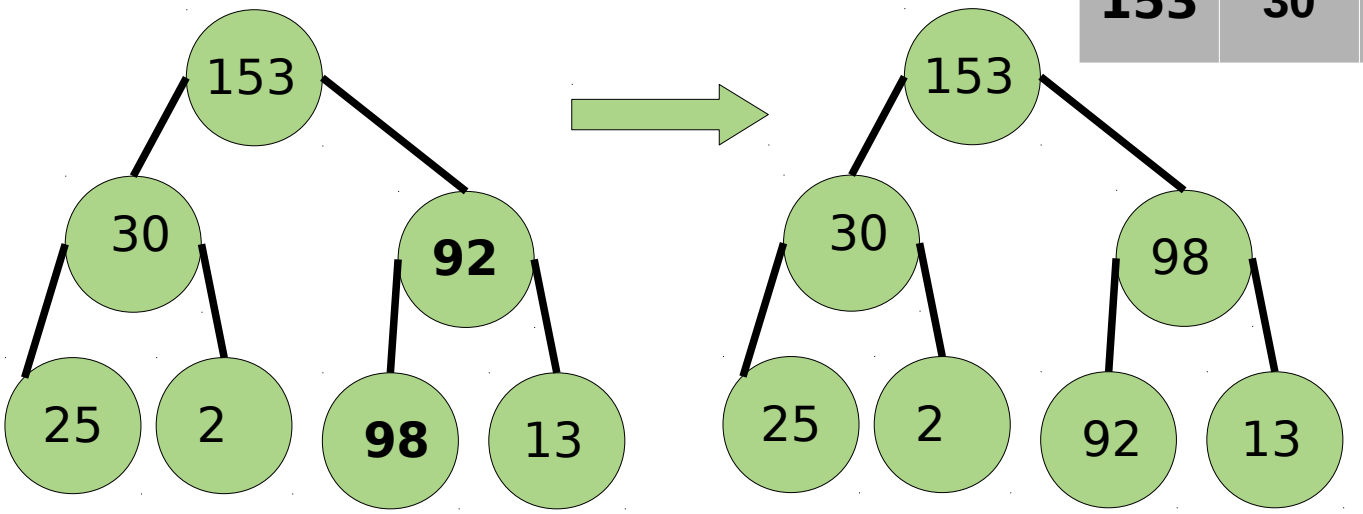
153	30	92	25	2	98	13	vetor v
0	1	2	3	4	5	6	índice i

→ Em um Heap máximo os pais sempre serão **maior ou igual** aos filhos.

→ É preciso fazer a troca caso haja algum filho maior que o pai.

→ A troca é feita entre os elementos do vetor, através de seus índices.

153	30	98	25	2	92	13



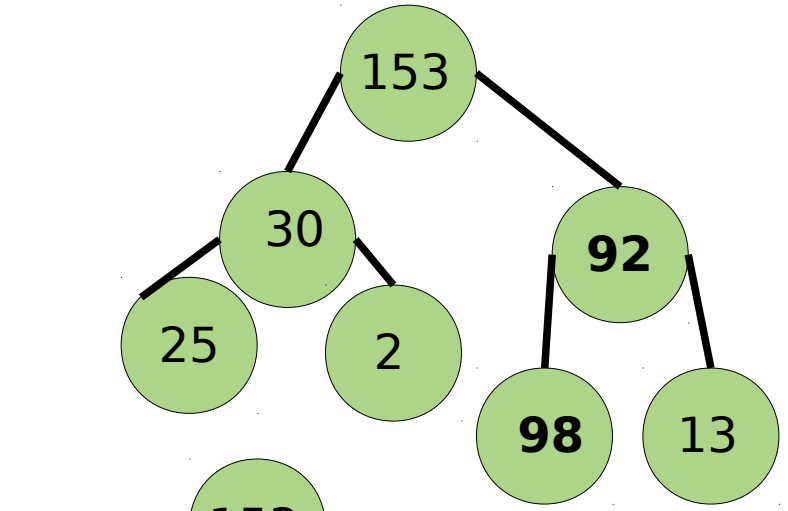
Transformar em Heap máximo (Código)

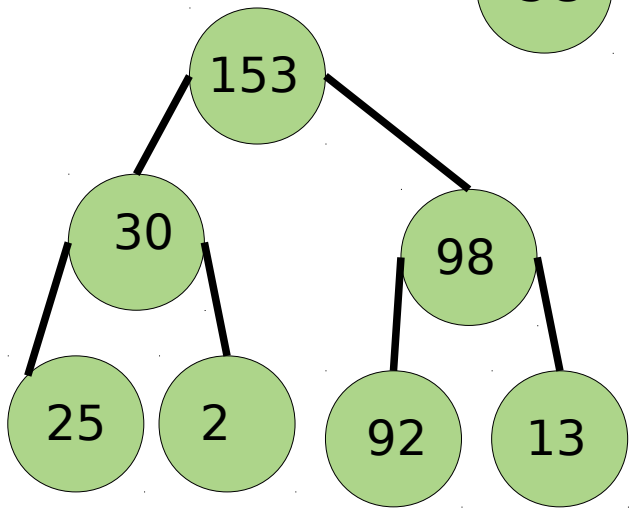
153	30	92	25	2	98	13
0	1	2	3	4	5	6

vetor v

índice i

```
// Se o filho da esquerda for maior que a raiz (pai)
if ( esq < n && v[esq] > v[pai] )
    pai = esq;

// Se o filho da direita for maior que a raiz (pai)
if ( dir < n && v[dir] > v[pai] )
    pai = dir;

// Se o pai (maior) não for a raiz
if ( pai != i ) {

    // Troca
    trocar( &v[i], &v[pai] );

    // Chamada recursiva
    criarHeap( v, n, pai );
}
```





153	30	98	25	2	92	13

Obteve-se o Heap máximo. Trocar a raiz com o último elemento!

Trocar a raiz com o último elemento

153	30	98	25	2	92	13	vetor v
0	1	2	3	4	5	6	índice i

→ Trocar o elemento do primeiro índice com o elemento do último índice.

13	30	98	25	2	92	153

→ Remover o último elemento do Heap.

13	30	98	25	2	92	153

→ Aplicar o Heap máximo novamente e recomeçar o processo com n-1 elementos no vetor.

13	30	98	25	2	92	~~153~~

```
// Se o pai (maior) não for a raiz
if ( pai != i ) {

    // Troca
    trocar( &v[i], &v[pai] );

    // Chamada recursiva
    criarHeap( v, n, pai );
}
```

Criar o Heap máximo novamente

13	30	98	25	2	92	
0	1	2	3	4	5	6

vetor v

índice i

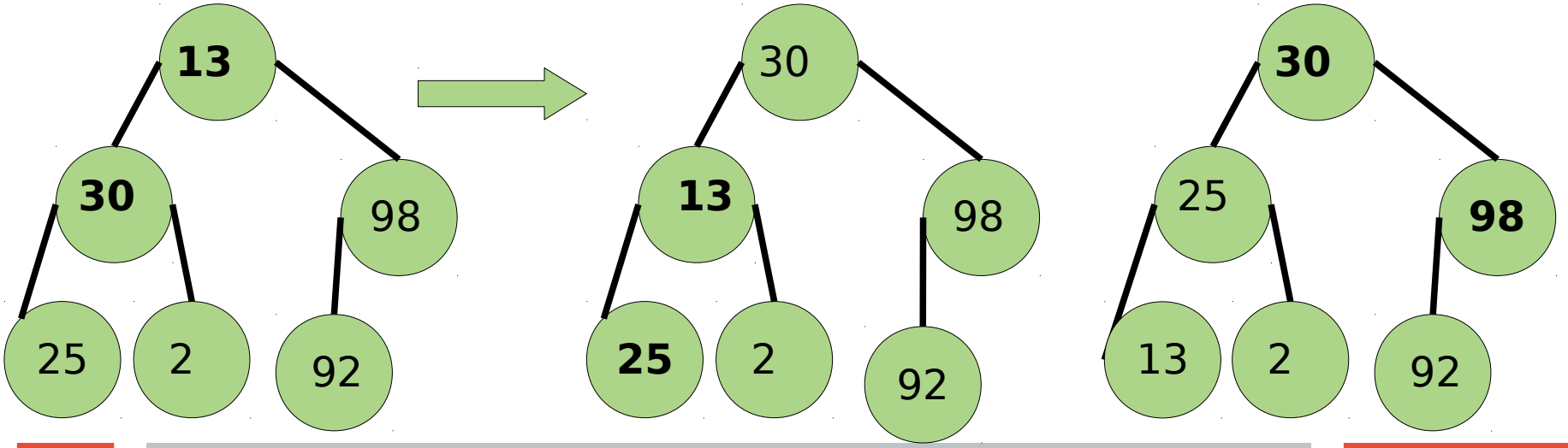
→ Em um Heap máximo os pais sempre serão **maior ou igual** aos filhos.

→ É preciso fazer a troca caso haja algum filho maior que o pai.

→ A troca é feita entre os elementos do vetor, através de seus índices.

30	13	98	25	2	92

30	25	98	13	2	92



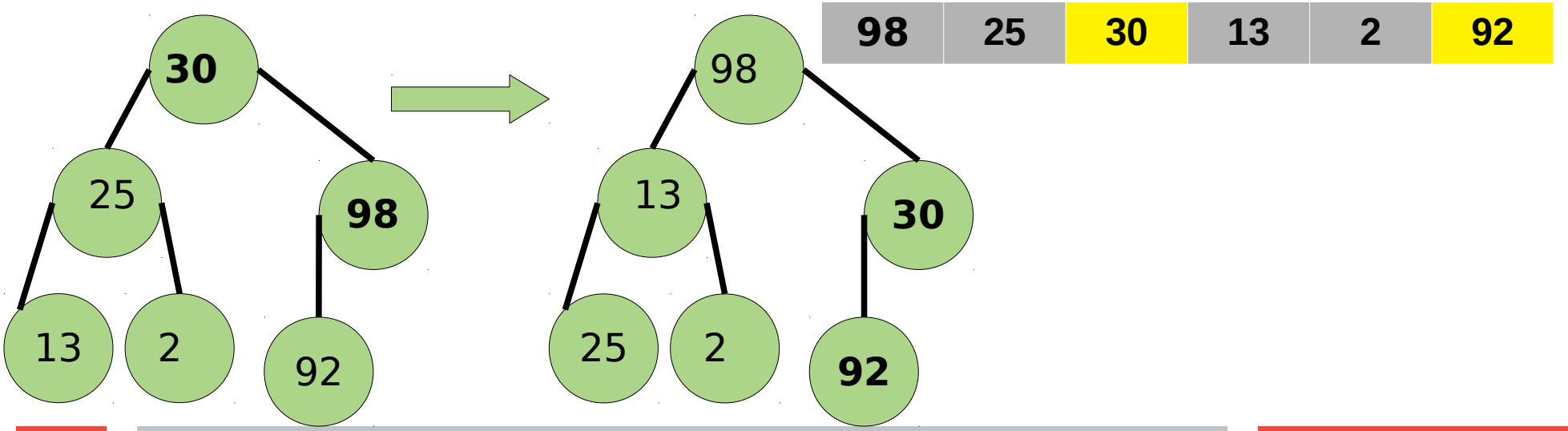
Criar o Heap máximo novamente

30	25	98	13	2	92		vetor v
0	1	2	3	4	5	6	índice i

→ Em um Heap máximo os pais sempre serão **maior ou igual** aos filhos.

→ É preciso fazer a troca caso haja algum filho maior que o pai.

→ A troca é feita entre os elementos do vetor, através de seus índices.



98	25	30	13	2	92

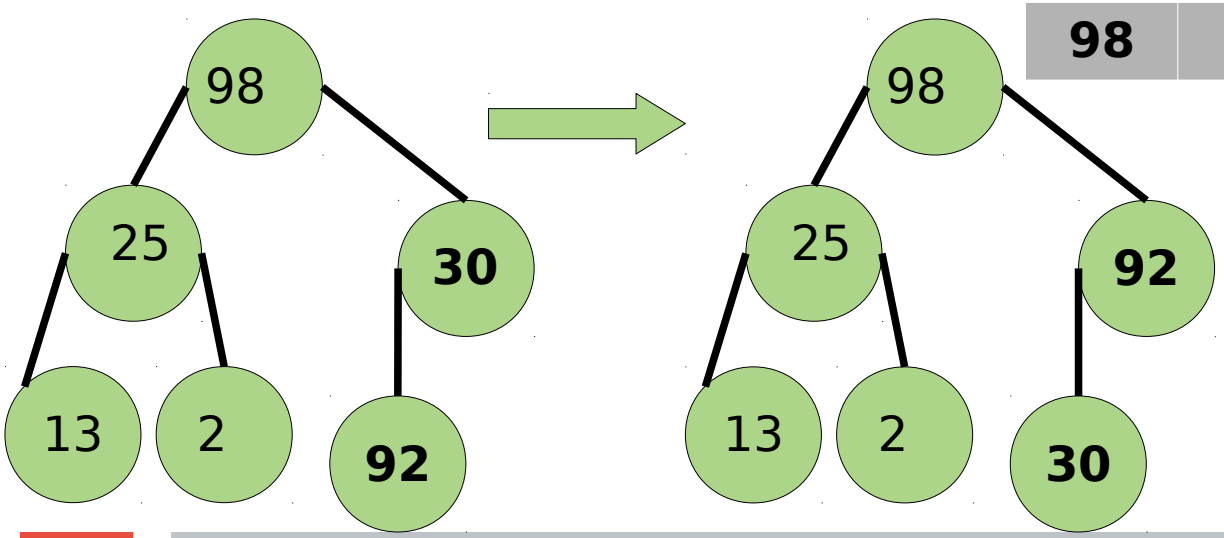
Criar o Heap máximo novamente

98	25	30	13	2	92		vetor v
0	1	2	3	4	5	6	índice i

→ Em um Heap máximo os pais sempre serão **maior ou igual** aos filhos.

→ É preciso fazer a troca caso haja algum filho maior que o pai.

→ A troca é feita entre os elementos do vetor, através de seus índices.



98	25	92	13	2	30

Obteve-se o Heap máximo
Trocar a raiz com o último!

Trocar a raiz com o último elemento

98	25	92	13	2	30	153	vetor v
0	1	2	3	4	5	6	índice i

→ Trocar o elemento do primeiro índice com o elemento do último índice.

30	13	92	25	2	98	153

→ Remover o último elemento do Heap.

30	25	92	13	2	98	153

→ Aplicar o Heap máximo novamente e recomeçar o processo com n-1 elementos no vetor.

30	25	92	13	2	~~98~~	~~153~~

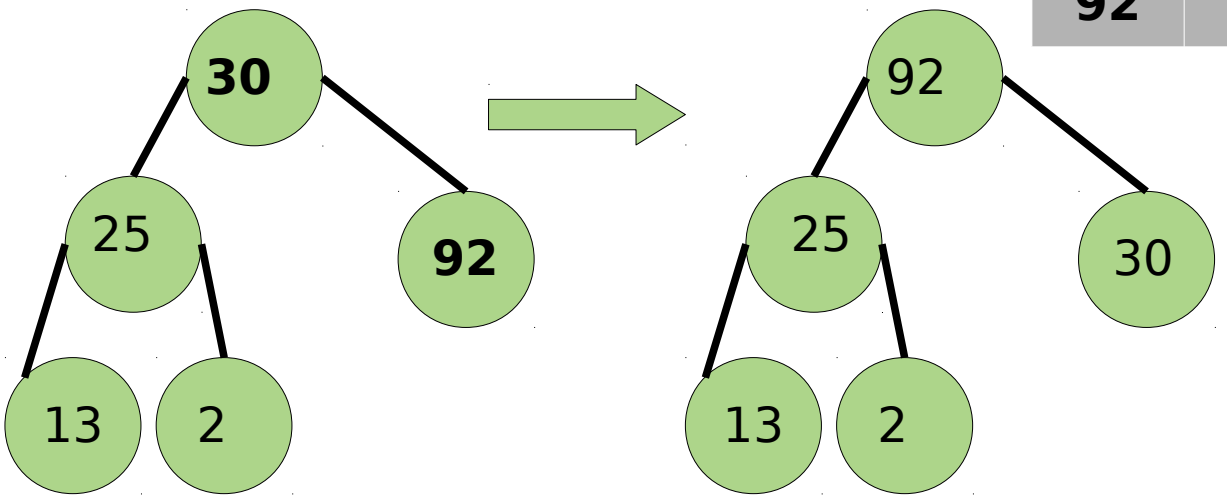
Criar o Heap máximo novamente

30	25	92	13	2			vetor v
0	1	2	3	4	5	6	índice i

→ Em um Heap máximo os pais sempre serão **maior ou igual** aos filhos.

→ É preciso fazer a troca caso haja algum filho maior que o pai.

→ A troca é feita entre os elementos do vetor, através de seus índices.



92	25	30	13	2

Obteve-se o Heap máximo

Trocar a raiz com o último!

Trocar a raiz com o último elemento

92	25	30	13	2	98	153	vetor v
0	1	2	3	4	5	6	índice i

→ Trocar o elemento do primeiro índice com o elemento do último índice.

2	25	30	13	92	98	153

→ Remover o último elemento do Heap.

2	25	30	13	92	98	153

→ Aplicar o Heap máximo novamente e recomeçar o processo com n-1 elementos no vetor.

2	25	30	13	~~92~~	~~98~~	~~153~~

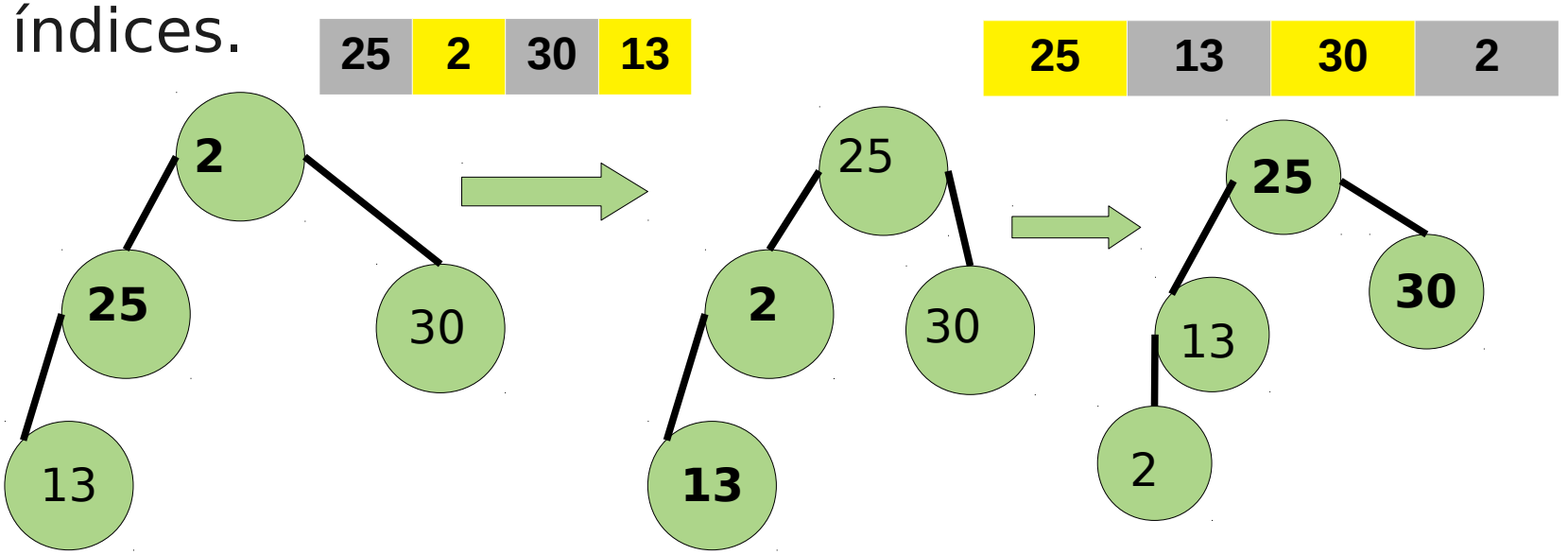
Criar o Heap máximo novamente

2	25	30	13				vetor v
0	1	2	3	4	5	6	índice i

→ Em um Heap máximo os pais sempre serão **maior ou igual** aos filhos.

→ É preciso fazer a troca caso haja algum filho maior que o pai.

→ A troca é feita entre os elementos do vetor, através de seus índices.



Criar o Heap máximo novamente

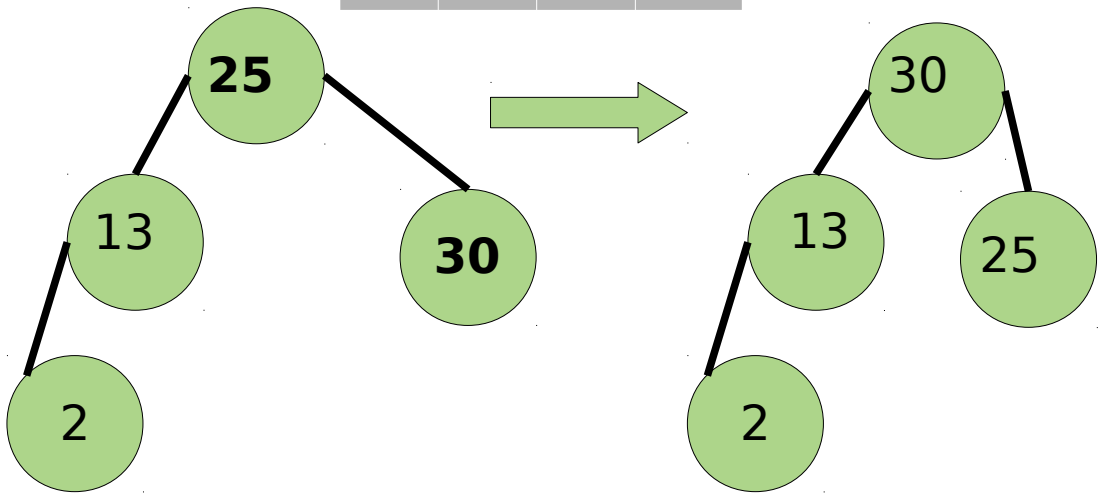
25	13	30	2				vetor v
0	1	2	3	4	5	6	índice i

→ Em um Heap máximo os pais sempre serão **maior ou igual** aos filhos.

→ É preciso fazer a troca caso haja algum filho maior que o pai.

→ A troca é feita entre os elementos do vetor, através de seus índices.

30	13	25	2



Obteve-se o Heap máximo

Trocar a raiz com o último!

Trocar a raiz com o último elemento

30	13	25	2	92	98	153	vetor v
0	1	2	3	4	5	6	índice i

→ Trocar o elemento do primeiro índice com o elemento do último índice.

2	13	25	30	92	98	153

→ Remover o último elemento do Heap.

2	13	25	30	92	98	153

→ Aplicar o Heap máximo novamente e recomeçar o processo com n-1 elementos no vetor.

2	13	25	~~30~~	~~92~~	~~98~~	~~153~~

Criar o Heap máximo novamente

2	13	25				
0	1	2	3	4	5	6

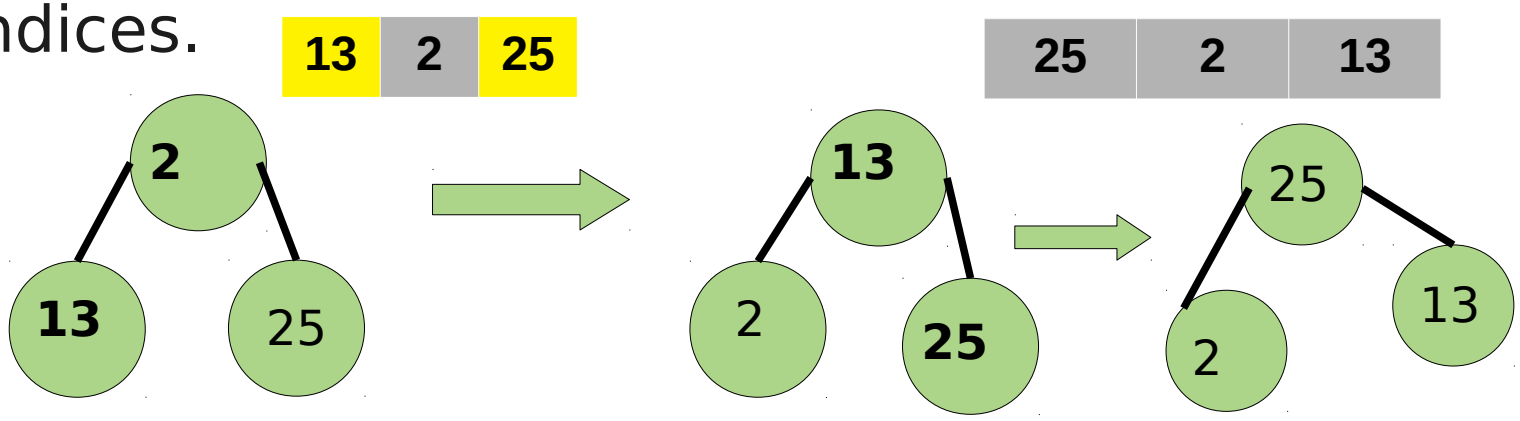
vetor v

índice i

→ Em um Heap máximo os pais sempre serão **maior ou igual** aos filhos.

→ É preciso fazer a troca caso haja algum filho maior que o pai.

→ A troca é feita entre os elementos do vetor, através de seus índices.



Obteve-se o Heap máximo
Trocar a raiz com o último!

Trocar a raiz com o último elemento

25	2	13	30	92	98	153	vetor v
0	1	2	3	4	5	6	índice i

→ Trocar o elemento do primeiro índice com o elemento do último índice.

13	2	25	30	92	98	153

→ Remover o último elemento do Heap.

13	2	25	30	92	98	153

→ Aplicar o Heap máximo novamente e recomeçar o processo com n-1 elementos no vetor.

13	2	~~25~~	~~30~~	~~92~~	~~98~~	~~153~~

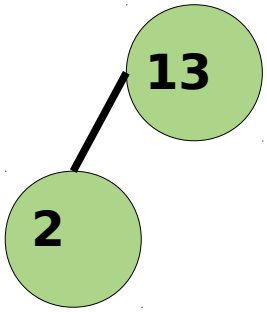
Criar o Heap máximo novamente

13	2						vetor v
0	1	2	3	4	5	6	índice i

→ Em um Heap máximo os pais sempre serão **maior ou igual** aos filhos.

→ É preciso fazer a troca caso haja algum filho maior que o pai.

→ A troca é feita entre os elementos do vetor, através de seus índices.



13	2

Já é Heap máximo
Troca a raiz com o último!

Trocar a raiz com o último elemento

13	2	25	30	92	98	153	vetor v
0	1	2	3	4	5	6	índice i

→ Trocar o elemento do primeiro índice com o elemento do último índice.

2	13	25	30	92	98	153

→ Remover o último elemento do Heap.

2	13	25	30	92	98	153

→ Com 1 elemento o algoritmo termina, pois está ordenado.

2	13	25	30	92	98	153

Funções heapSort e trocar (Códigos)

```c
void heapSort( int v[], int n ) {

    // Construir o heap
    for ( int i = n/2-1; i >= 0; i--)
        criarHeap( v, n, i );

    // Extraindo elementos do heap
    for ( int i = n-1; i>0; i-- ) {

        // Trocar a raiz com o último elemento
        trocar( &v[0], &v[i] );

        // Criar Heap máximo
        criarHeap( v, i, 0 );
    }
}
```

```c
// Troca dois elementos
void trocar( int *x, int *y ){
    int aux;
    aux = *x;
    *x = *y;
    *y = aux;
}
```

Função criarHeap (Código)

```
// Cria as sub-árvores tendo i como raiz
void criarHeap( int v[], int n, int i ) {

    // Para o heap máximo o pai sempre tem que ser o maior
    int pai = i;
    int esq = 2*i + 1;
    int dir = 2*i + 2;


    // Se o filho da esquerda for maior que a raiz (pai)
    if ( esq < n && v[esq] > v[pai] )
        pai = esq;

    // Se o filho da direita for maior que a raiz (pai)
    if ( dir < n && v[dir] > v[pai] )
        pai = dir;

    // Se o pai (maior) não for a raiz
    if ( pai != i ) {

        // Troca
        trocar( &v[i], &v[pai] );

        // Chamada recursiva
        criarHeap( v, n, pai );
    }
}
```