CI1057: Algoritmos e Estruturas de Dados III
Ordenação Externa

Profa. Carmem Hara

Departamento de Informática/UFPR

11 de julho de 2024

Ordenação Externa

- Necessária quando a quantidade a ser ordenada não cabe na memória principal
- O custo de um algoritmo de ordenação externa considera apenas a quantidade de leituras e escritas em memória secundaria
- Podem existir restrições quanto ao acesso aos dados (sequencial / randômico): fitas, discos
- O desenvolvimento de algoritmos é dependente do estado atual da tecnologia
  - HDD (*Hard Disk Drive*)
  - SSD (*Solid State Drive:*)
    - Leitura aprox. 50 vezes mais rápida que o HDD
    - Gravação aprox. 75 vezes mais rápida que o HDD

Algoritmos de Ordenação Externa

- Objetivo: minimizar a quantidade de vezes que um mesmo item é transferido da memória principal para a memória externa e vice-versa.
- Principal método: ordenação por intercalação (*sort-merge*)

Estratégia

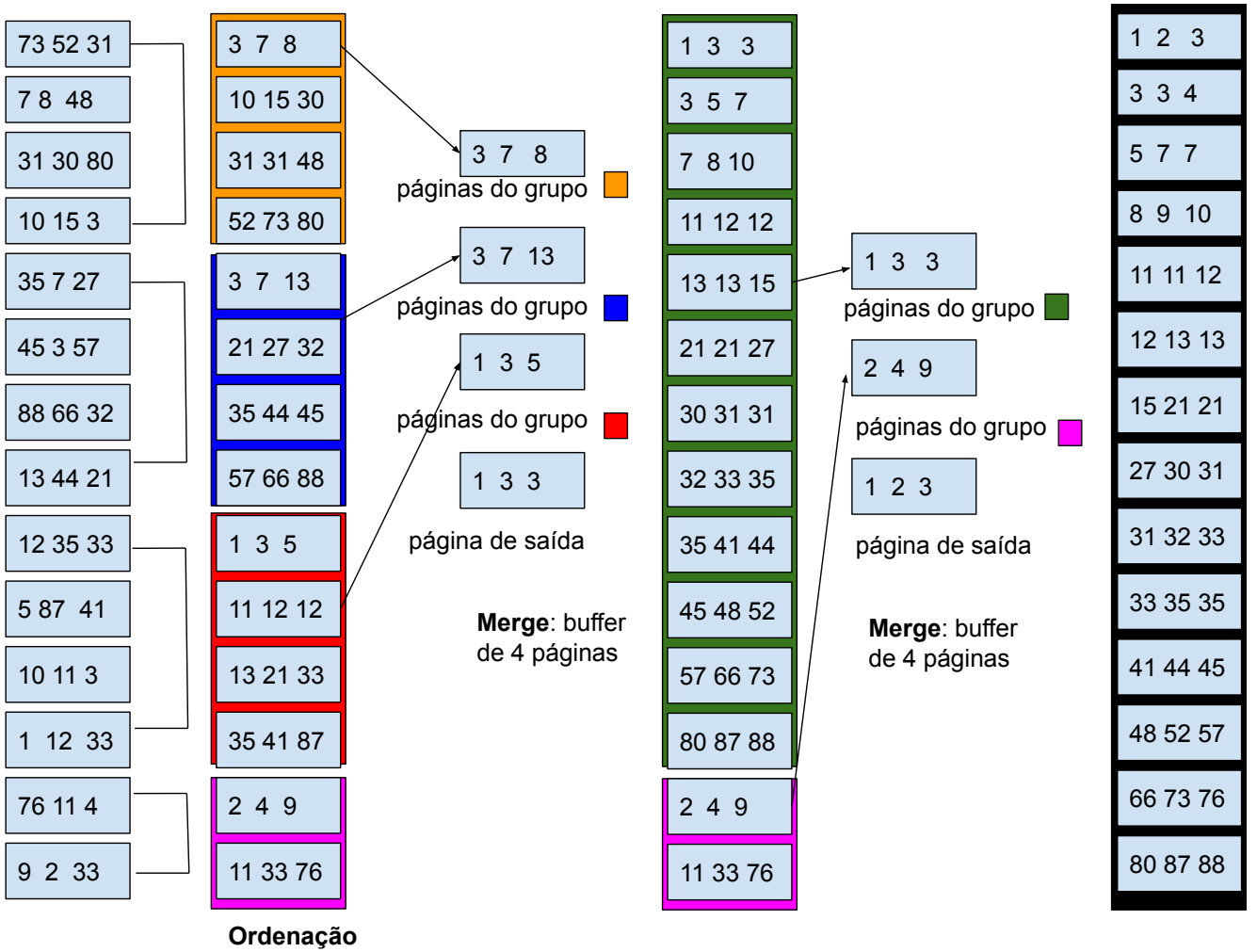
São 2 passos:

1. O arquivo é dividido em conjuntos de blocos do tamanho da memoria interna disponivel:
   Cada conjunto é ordenado na memória principal
2. Os conjuntos de blocos ordenados sao intercalados:
   Várias passadas sobre o arquivo podem ser necessárias ate' que ele esteja completamente ordenado

Exemplo

- Arquivo de entrada com 14 páginas
  ($n = 14$, a leitura é feita por página)
- Páginas contendo 3 registros
- Buffer em memória principal de 4 páginas ($p = 4$)

Exemplo


73 52 31
7 8 48
31 30 80
10 15 3
35 7 27
45 3 57
88 66 32
13 44 21
12 35 33
5 87 41
10 11 3
1 12 33
76 11 4
9 2 33
3 7 8
10 15 30
31 31 48
52 73 80
3 7 13
21 27 32
35 44 45
57 66 88
1 3 5
11 12 12
13 21 33
35 41 87
2 4 9
11 33 76
Ordenação
3 7 8
páginas do grupo
3 7 13
páginas do grupo
1 3 5
páginas do grupo
1 3 3
página de saída
Merge: buffer de 4 páginas
1 3 3
3 5 7
7 8 10
11 12 12
13 13 15
21 21 27
30 31 31
32 33 35
35 41 44
45 48 52
57 66 73
80 87 88
2 4 9
11 33 76
1 3 3
páginas do grupo
2 4 9
páginas do grupo
1 2 3
página de saída
Merge: buffer de 4 páginas
1 2 3
3 3 4
5 7 7
8 9 10
11 11 12
12 13 13
15 21 21
27 30 31
31 32 33
33 35 35
41 44 45
48 52 57
66 73 76
80 87 88

Procedimento de Ordenação

1. Passo de ordenação
   - São lidas 4 páginas do arquivo
   - Os registros nas páginas são ordenados em memória principal
   - As páginas com os registros ordenados são gravadas em disco

   Este processo é repetido $\lceil 14/4 \rceil$ vezes
2. Passo de merge
   - 3 páginas do buffer são usadas para leitura dos conjuntos de páginas ordenadas
   - 1 página do buffer é utilizada para saída do processo de merge

   Este processo é repetido $log_{p-1}(\frac{n}{p})$ vezes

Custo da Ordenação

1. Passo de ordenação
   - Cada página é lida e gravada 1 vez
   - Custo $= 2 * n$, onde n é a quantidade de páginas do arquivo de entrada
2. Passo de merge
   - Existem $log_{p-1}(\frac{n}{p})$ passos de merge
   - Em cada passo cada página é lida e gravada 1 vez
   - Custo $= 2 * n * log_{p-1}(\frac{n}{p})$

O custo da ordenação externa é $2 * n * (log_{p-1}(\frac{n}{p}) + 1)$.

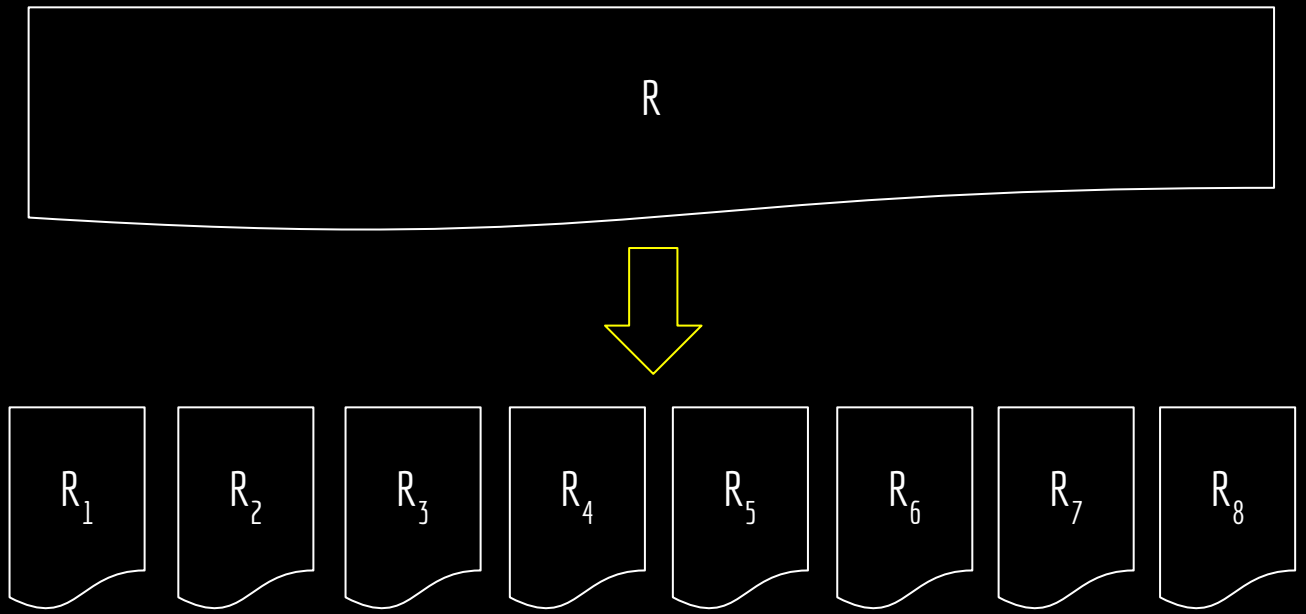
Implementação do Passo de Merge Utilizando Heap

Slides do Prof. Paulo Almeida

- A utilização do heap é importante quando a quantidade de conjuntos ordenados gerados no passo de ordenação for grande Por exemplo, se for ≥ 8 porque o custo para manter a ordem heap é de $lg(8) = 3$.

P-way merging

Particionar o arquivo *R* em *p* partes aproximadamente iguais R_1, R_2, ..., R_p, onde cada R_x cabe na memória principal.

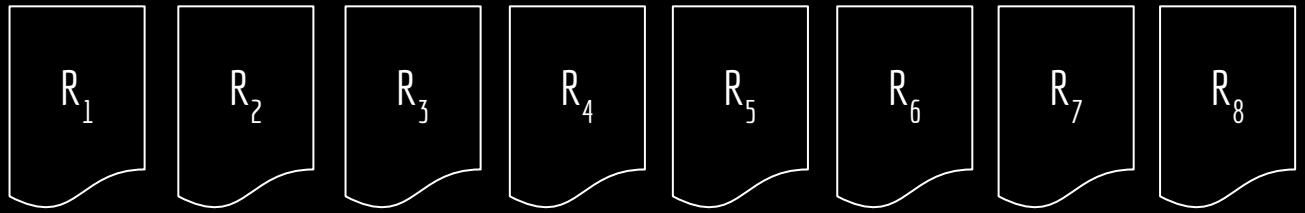


Exemplo com P=8

P-way merging

Particionar o arquivo *R* em *p* partes aproximadamente iguais R_1, R_2, ..., R_p, onde cada R_x cabe na memória principal.

Carregar cada R_x para a memória principal, ordenar, e armazenar novamente na memória secundária.



P-way merging

Particionar o arquivo *R* em *p* partes aproximadamente iguais R_1, R_2, ..., R_p, onde cada R_x cabe na memória principal.

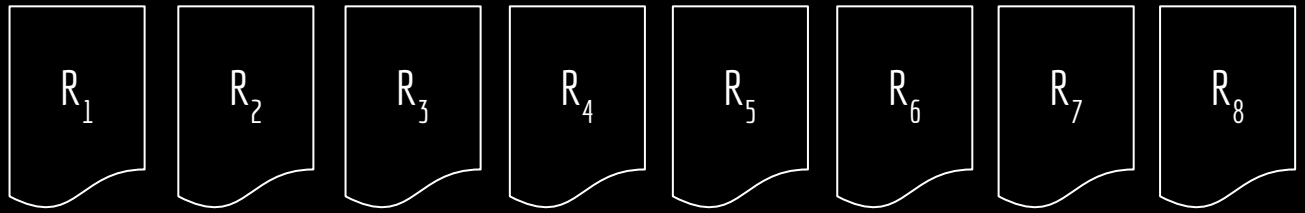
Carregar cada R_x para a memória principal, ordenar, e armazenar novamente na memória secundária.

Abrir ponteiros para os *p* arquivos R_1, ..., R_p.

Abrir um ponteiro *S* para um arquivo que receberá os dados ordenados.

Olhar para a cabeça dos arquivos R_1, ..., R_p, e escrever em S o dado do arquivo com o menor valor.

Incrementar o ponteiro do arquivo, e repetir o processo até esgotar todos os arquivos.



Exemplo

70	58	62	5	53	76	10	22	31	39	34	73	71	21	55	0	60	11	41	27	96	90	86	97	63	12	8	84	3	2	85	77

Arquivo grande.

Exemplo

70	58	62	5	53	76	10	22	31	39	34	73	71	21	55	0	60	11	41	27	96	90	86	97	63	12	8	84	3	2	85	77

Dividir em arquivos que podemos tratar na memória principal.

70
58
62
5

53
76
10
22

31
39
34
73

71
21
55
0

60
11
41
27

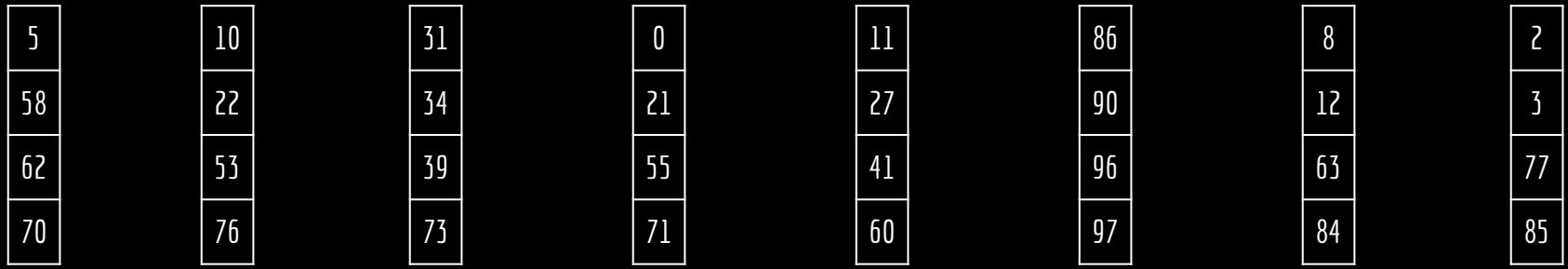
96
90
86
97

63
12
8
84

3
2
85
77

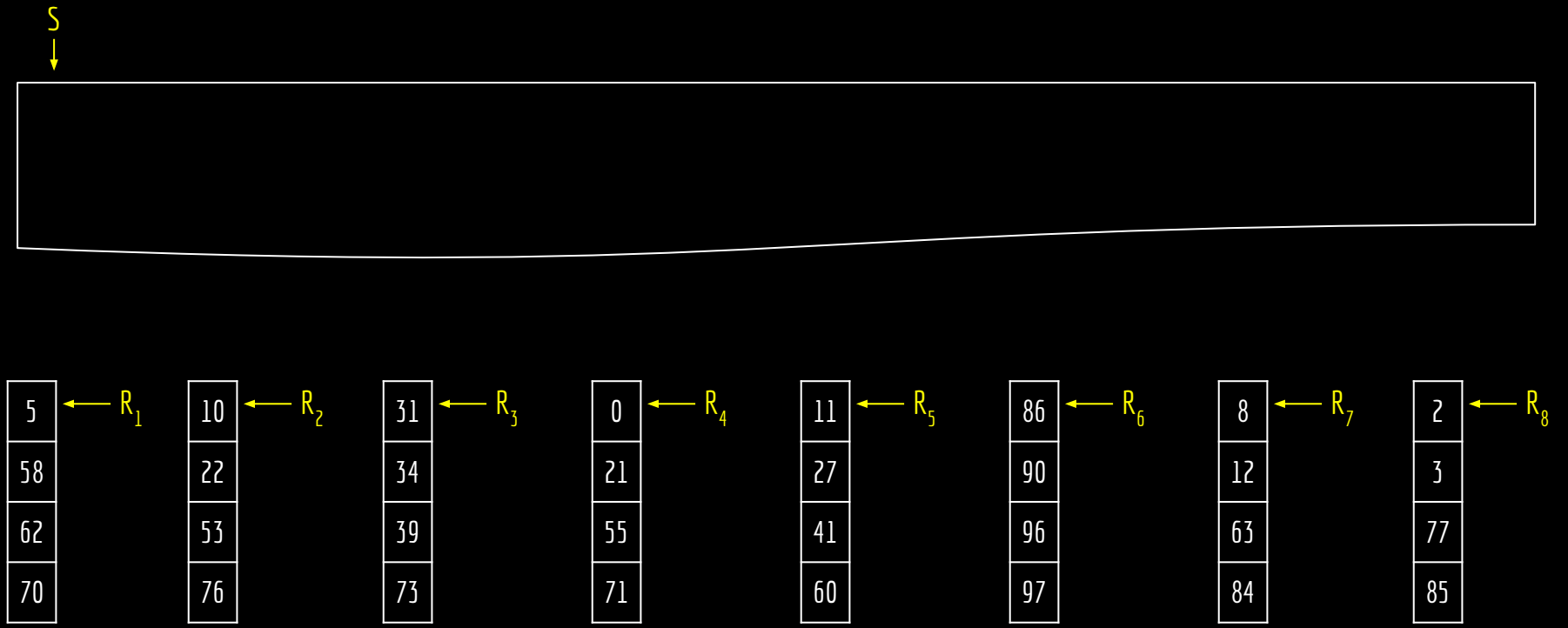
Exemplo

Ordenar cada arquivo utilizando algum algoritmo de ordenação (ex.: Merge Sort).



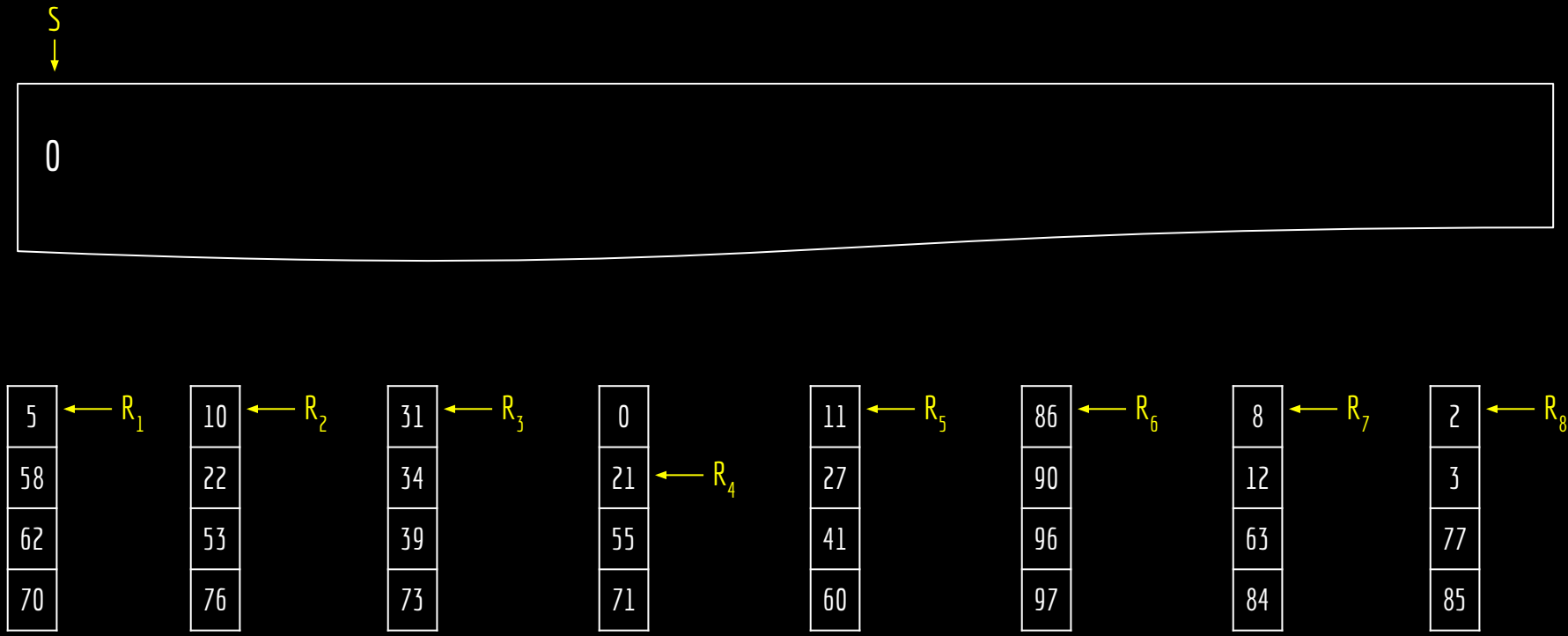
Exemplo

Abrir um ponteiro para cada arquivo, e um ponteiro para o arquivo ordenado.



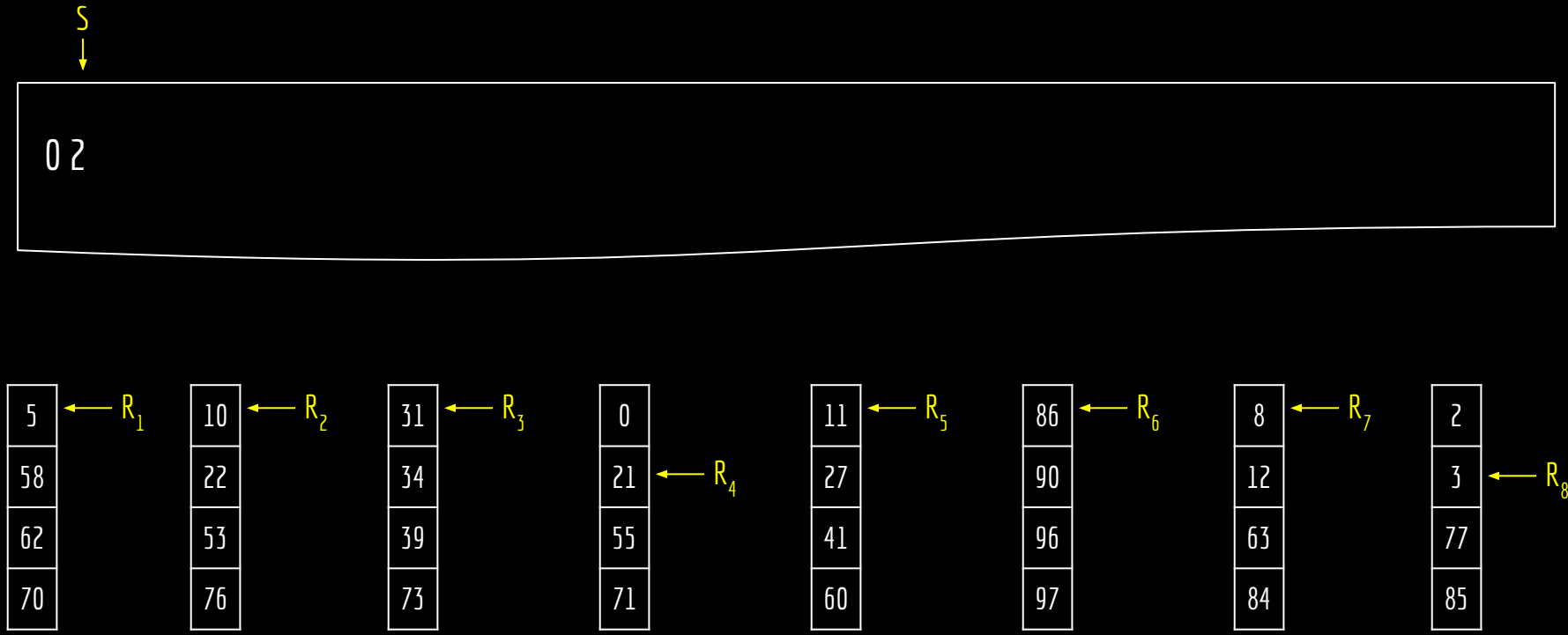
Exemplo

Verificar qual dos ponteiros escrever a seguir no arquivo.
Utilizar um Selection Sort, tomando *P-1* comparações a cada iteração.



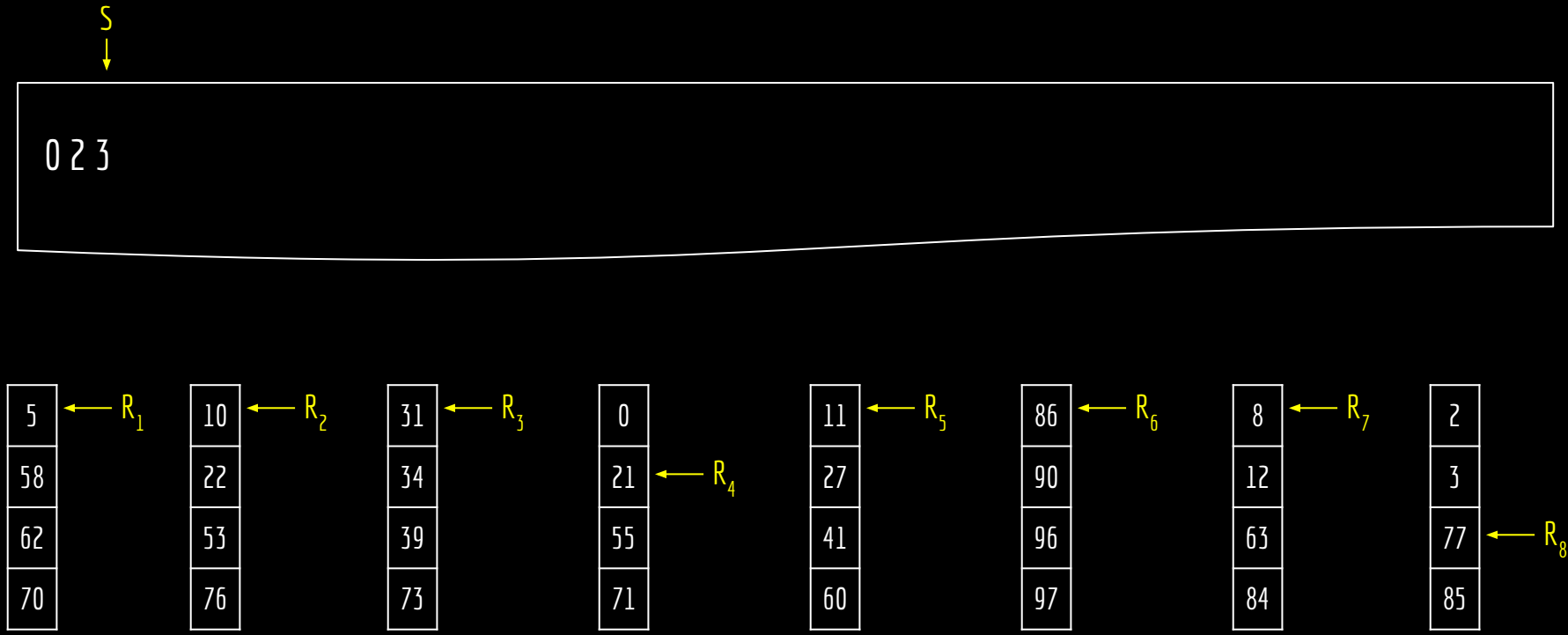
Exemplo

Verificar qual dos ponteiros escrever a seguir no arquivo.
Utilizar um Selection Sort, tomando *P-1* comparações a cada iteração.



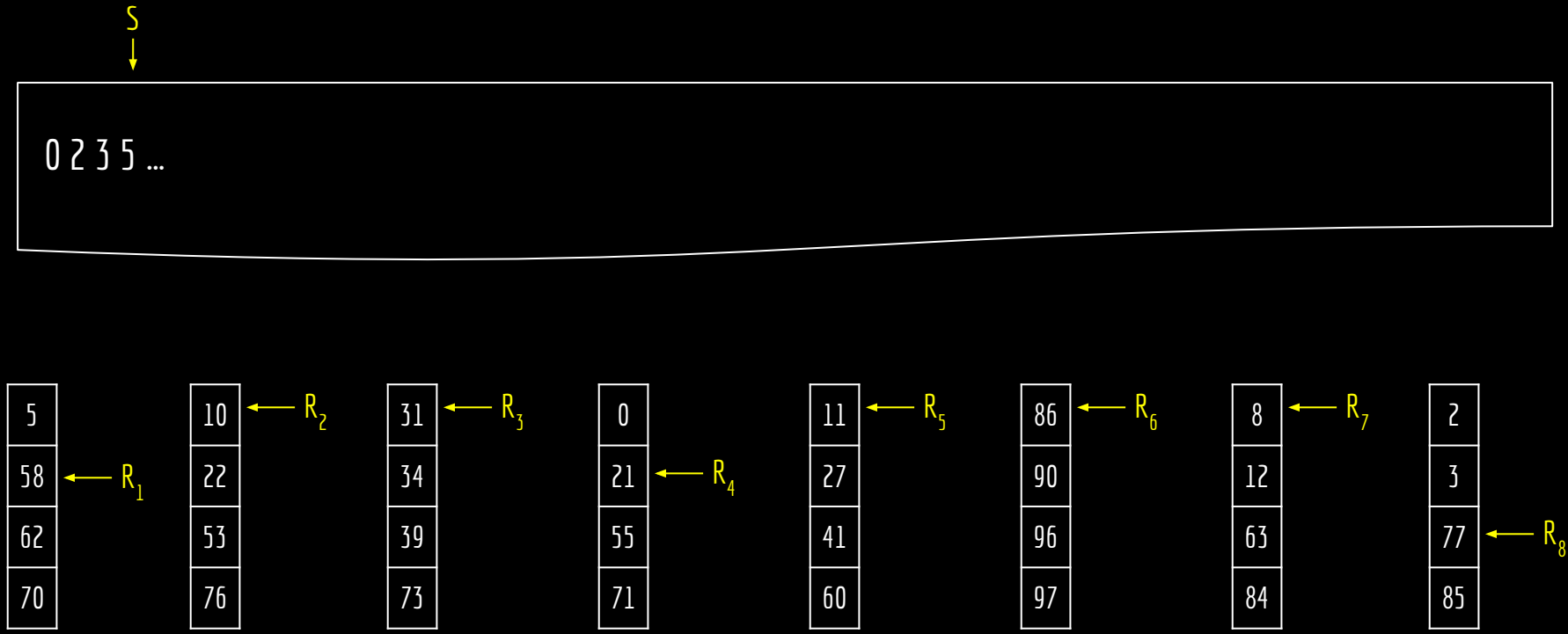
Exemplo

Verificar qual dos ponteiros escrever a seguir no arquivo.
Utilizar um Selection Sort, tomando *P-1* comparações a cada iteração.



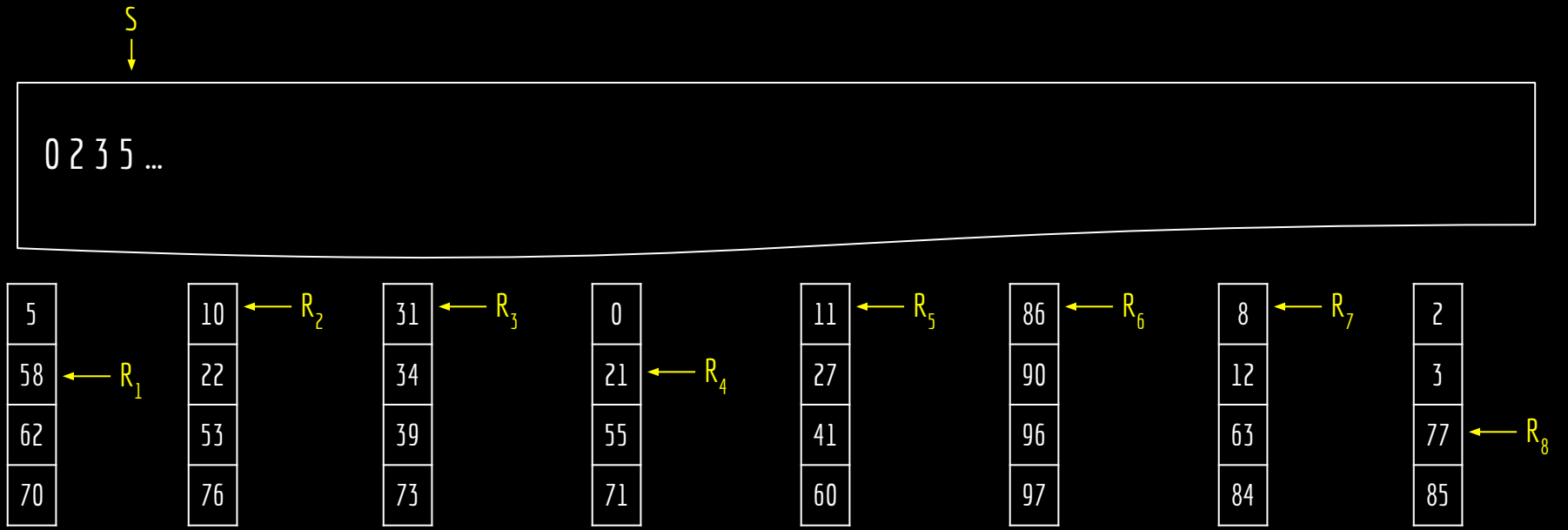
Exemplo

Verificar qual dos ponteiros escrever a seguir no arquivo.
Utilizar um Selection Sort, tomando *P-1* comparações a cada iteração.



Problemas

Problemas com essa estratégia?



Problemas

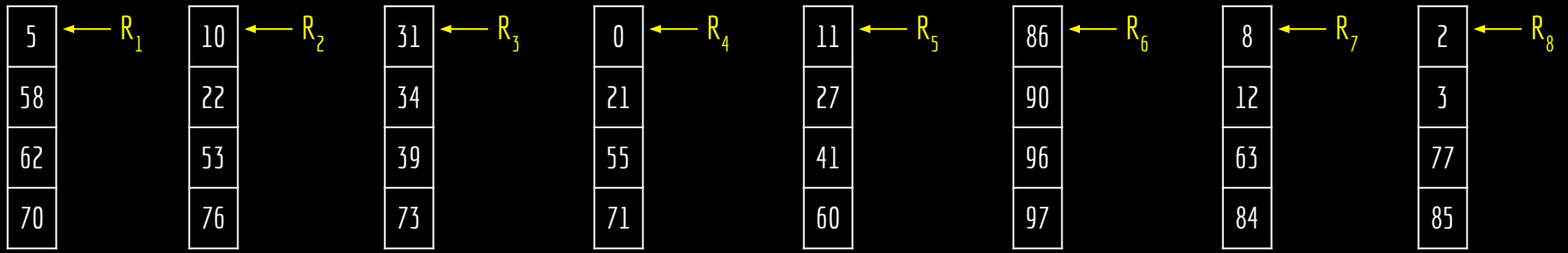
Problemas com essa estratégia?

- Funciona bem para valores de *P* pequenos.
- Valores grandes de *P* levam a um grande número de comparações a cada iteração pelo *Selection Sort*.

Resolvendo

Precisamos de uma estrutura que garantidamente mantenha a cabeça do arquivo vencedor prontamente acessível, e seja simples de atualizar.

O que podemos usar?

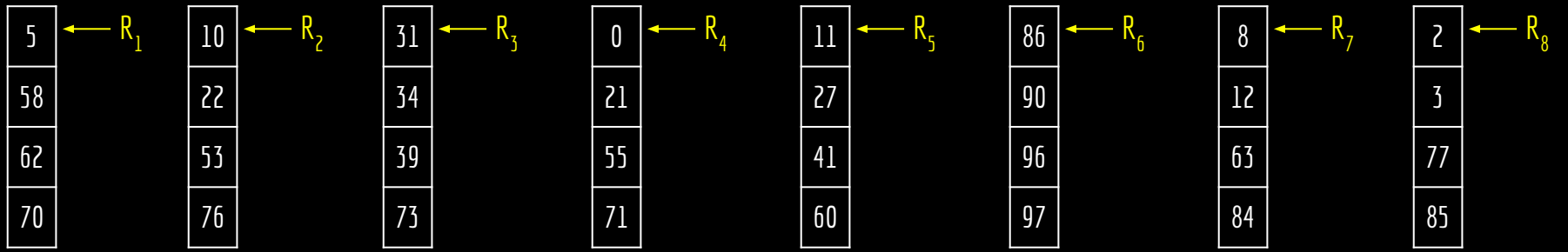


Heaps - Relembrando

Propriedades da Min-Heap.

1. A árvore é completa ou quase completa.
2. Uma heap de mínimo (min-heap) deve satisfazer a propriedade da min-heap.
   $h[pai(i)] \leq h[i], \ \forall i > 1$

Usando Heaps

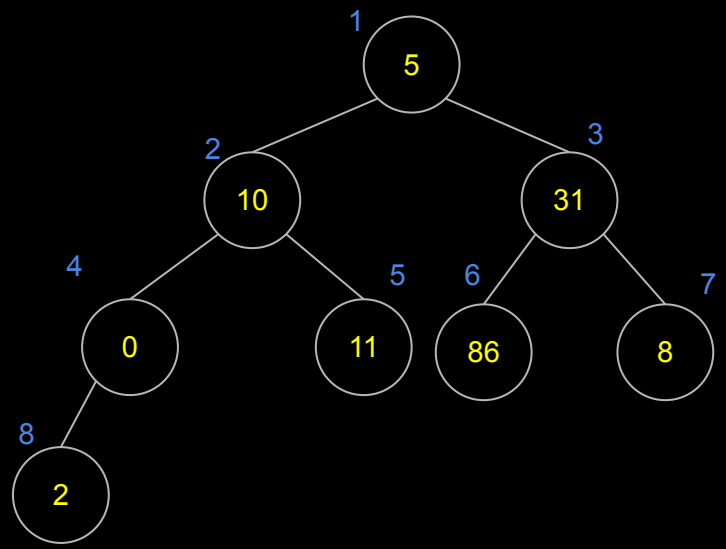


```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	



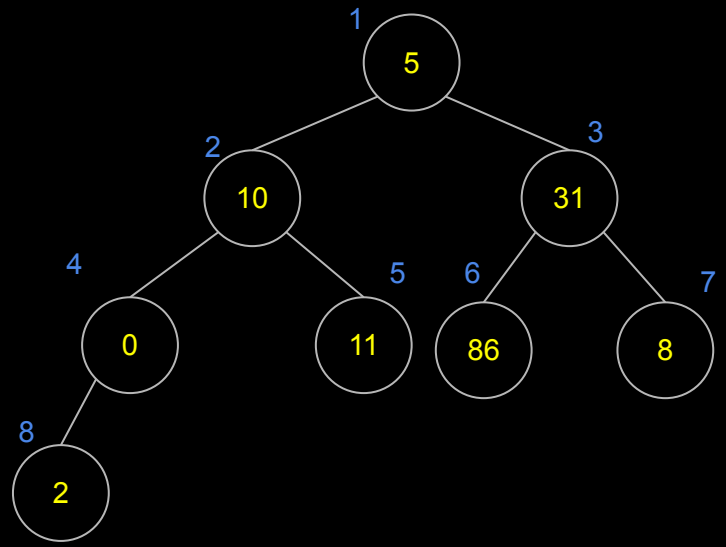
R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

construir-min-heap(v,8)

n	i
8	4

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```



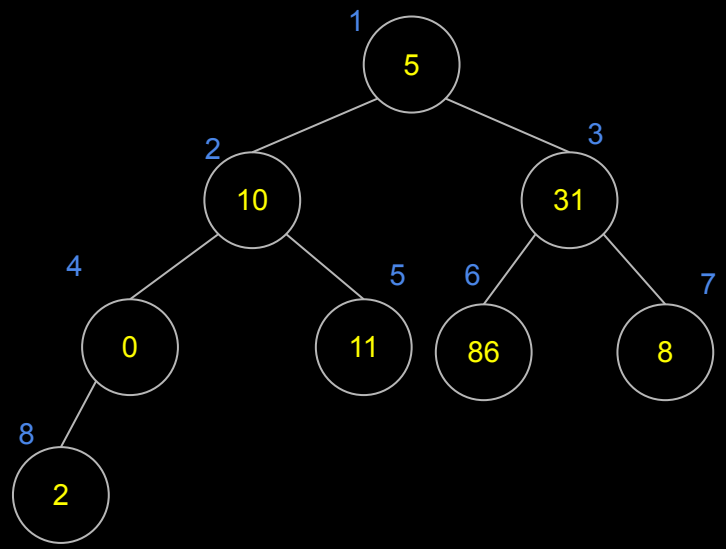
R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

construir-min-heap(v,8)

n	i
8	4

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

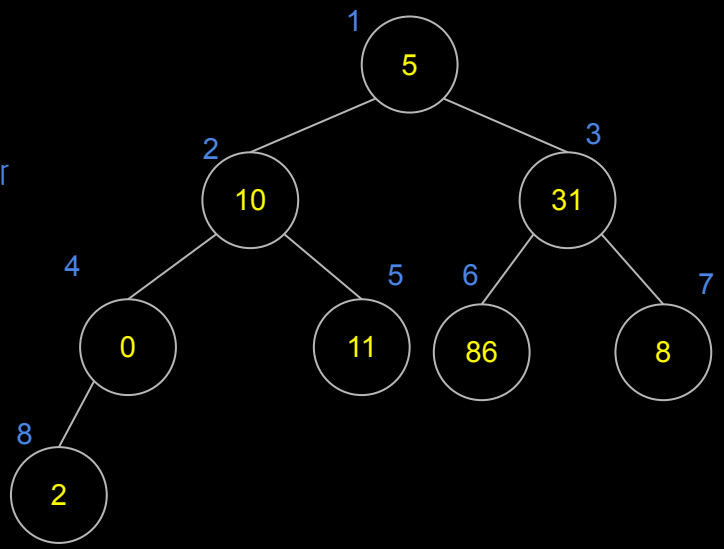
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	4

min-heapify(v,4,8)

n	i	l	r	menor
8	4	8		



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

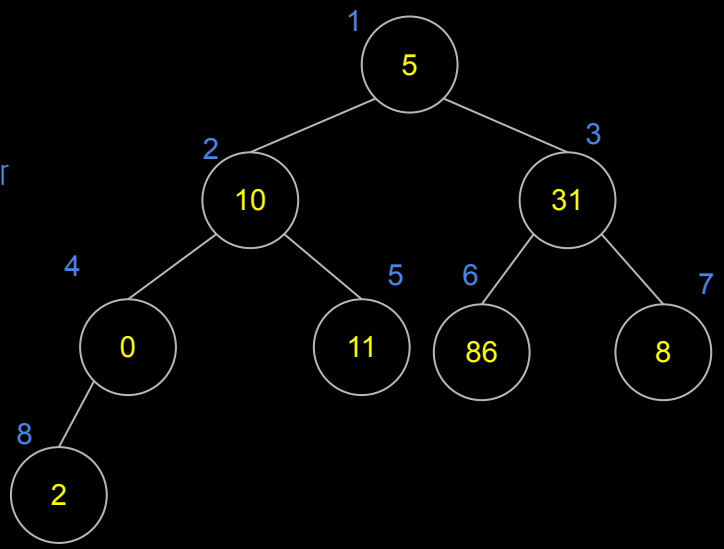
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	4

min-heapify(v,4,8)

n	i	l	r	menor
8	4	8	9	



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

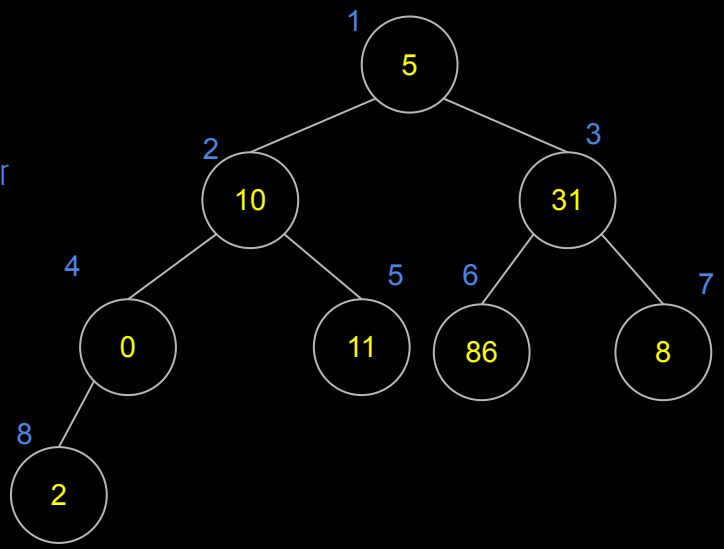
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	4

min-heapify(v,4,8)

n	i	l	r	menor
8	4	8	9	4



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

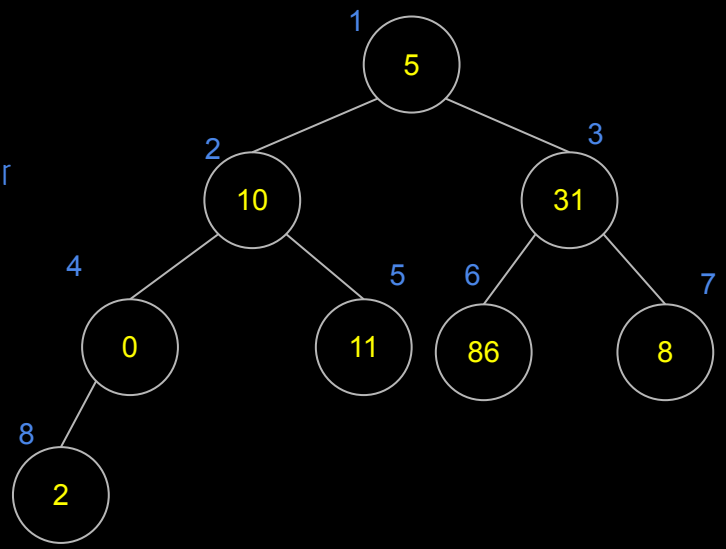
construir-min-heap(v,8)

n	i
8	4

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

min-heapify(v,4,8)

n	i	l	r	menor
8	4	8	9	4



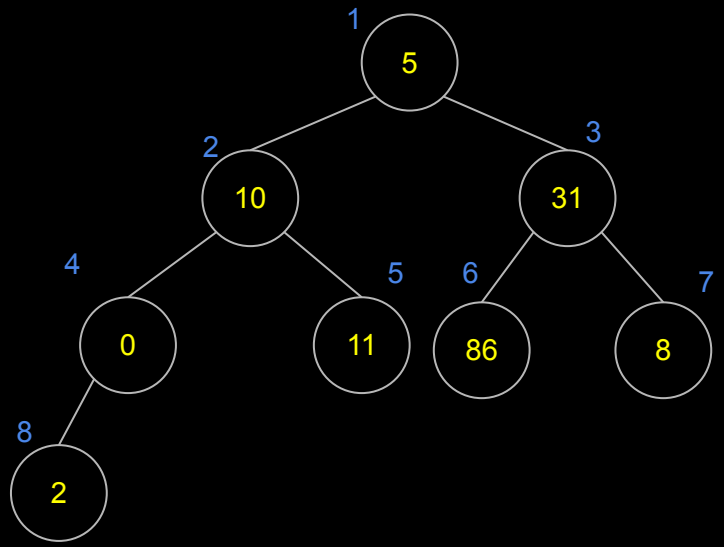
R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

construir-min-heap(v,8)

n	i
8	3

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```



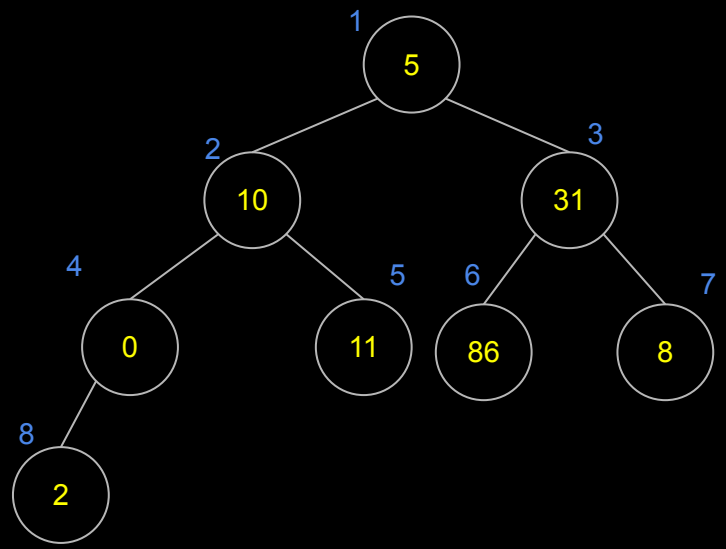
R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

construir-min-heap(v,8)

n	i
8	3

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

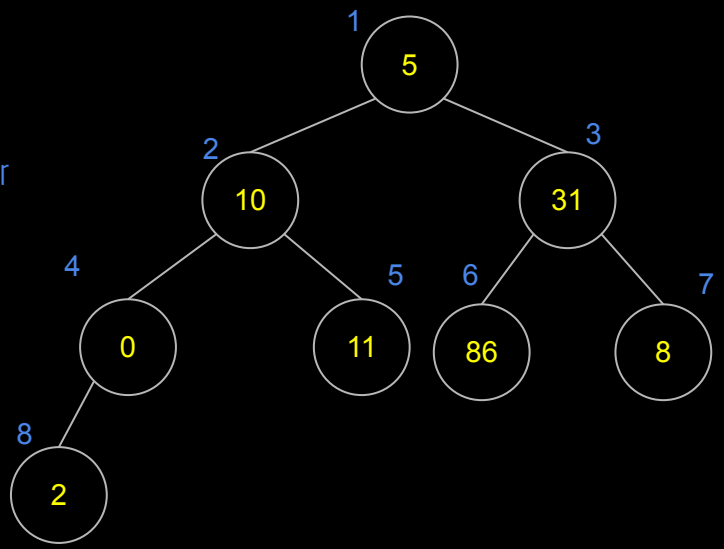
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	3

min-heapify(v,3,8)

n	i	l	r	menor
8	3	6	7	



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

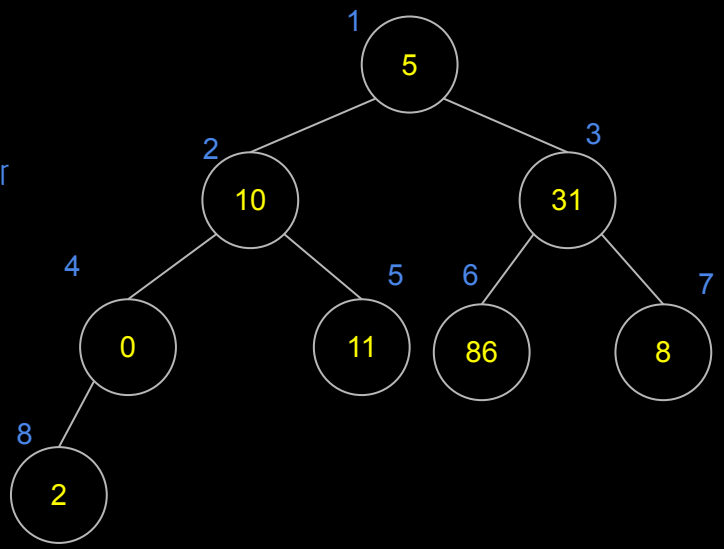
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	3

min-heapify(v,3,8)

n	i	l	r	menor
8	3	6	7	7



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

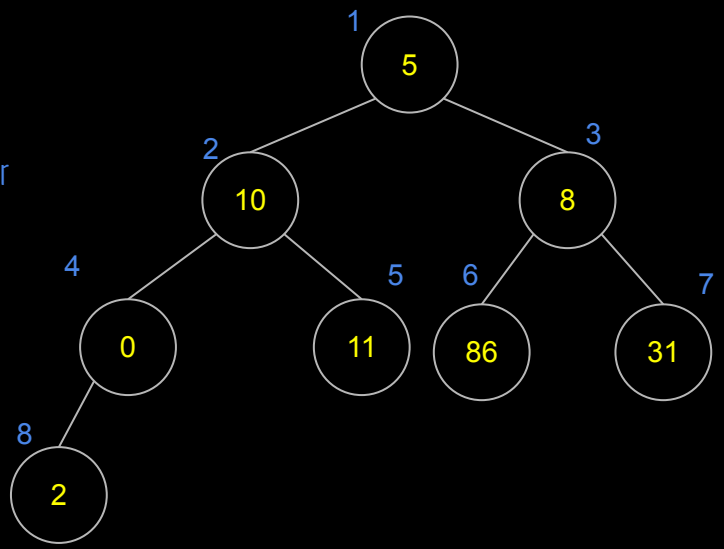
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	3

min-heapify(v,3,8)

n	i	l	r	menor
8	3	6	7	7



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

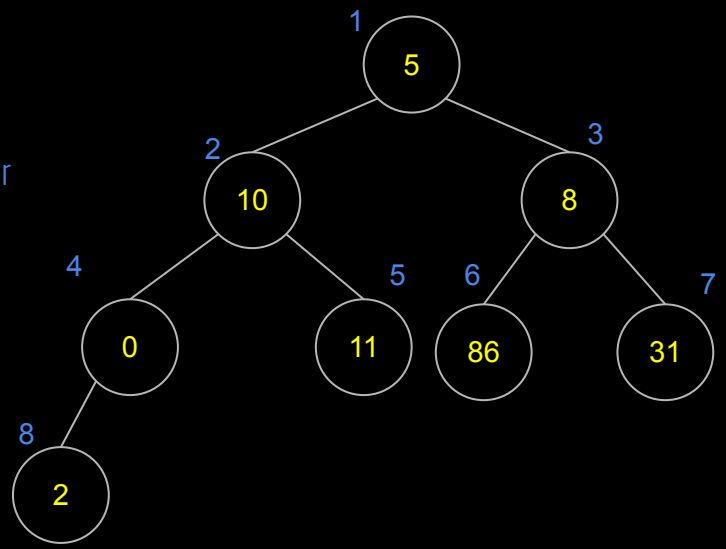
construir-min-heap(v,8)

n	i
8	3

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

min-heapify(v,3,8)

n	i	l	r	menor
8	3	6	7	7



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

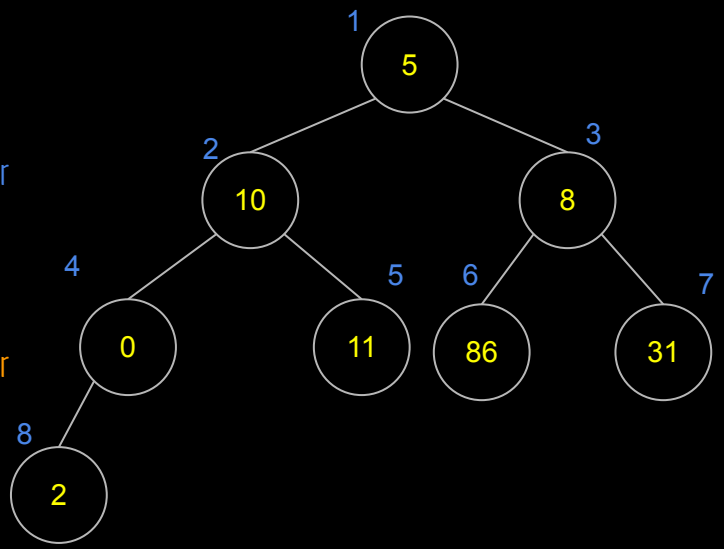
n	i
8	3

min-heapify(v,3,8)

n	i	l	r	menor
8	3	6	7	7

min-heapify(v,3,7)

n	i	l	r	menor
8	7	14	15	



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

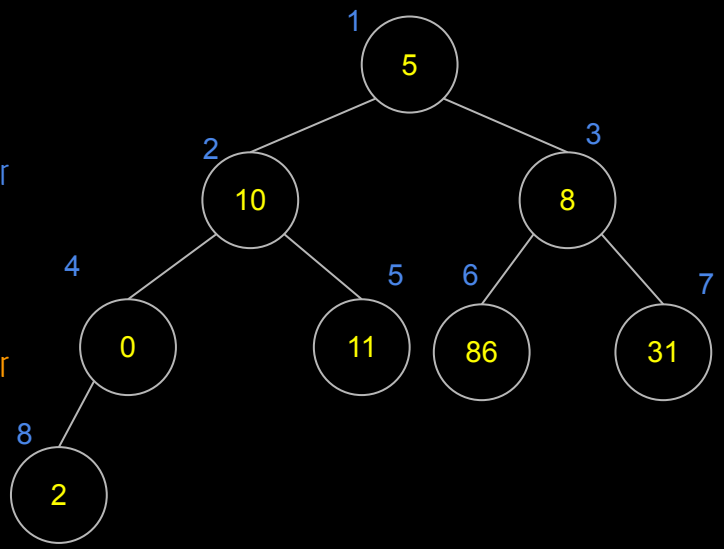
n	i
8	3

min-heapify(v,3,8)

n	i	l	r	menor
8	3	6	7	7

min-heapify(v,3,7)

n	i	l	r	menor
8	7	14	15	7



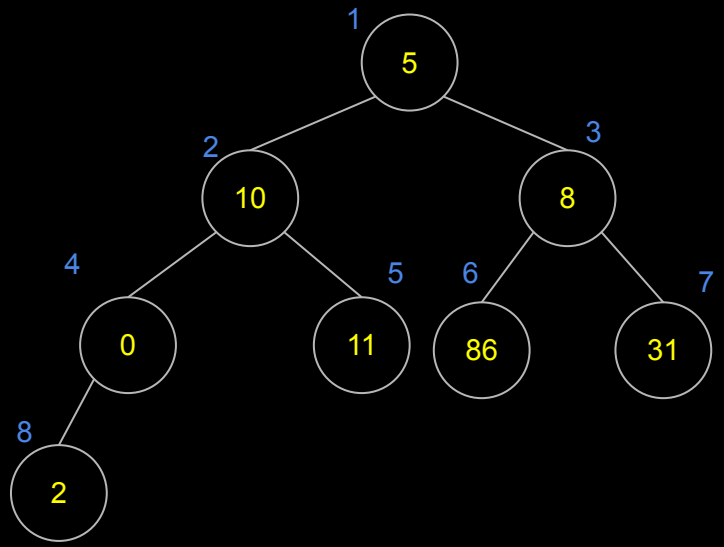
R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

construir-min-heap(v,8)

n	i
8	2

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```



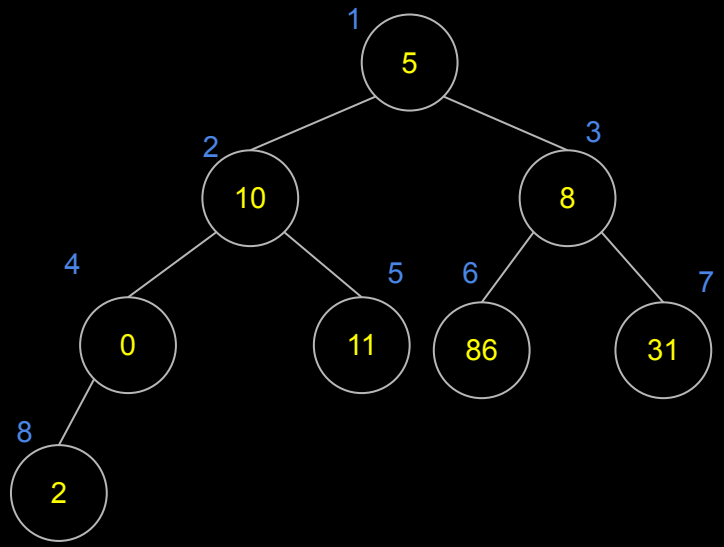
R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

construir-min-heap(v,8)

n	i
8	2

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

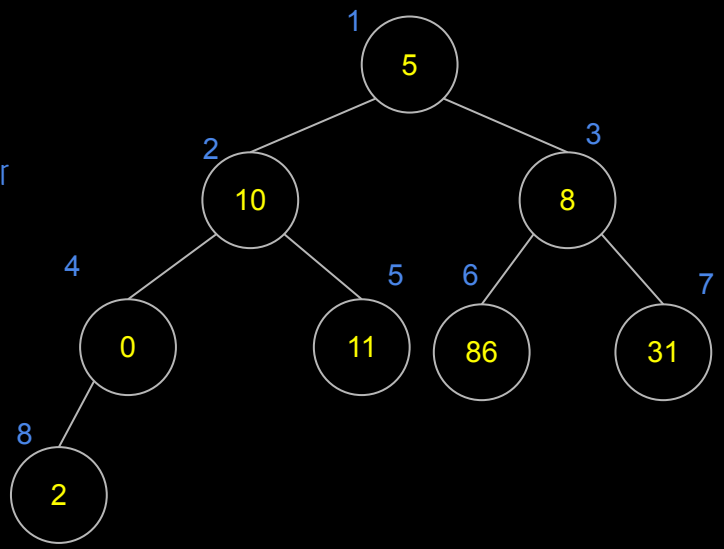
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	2

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

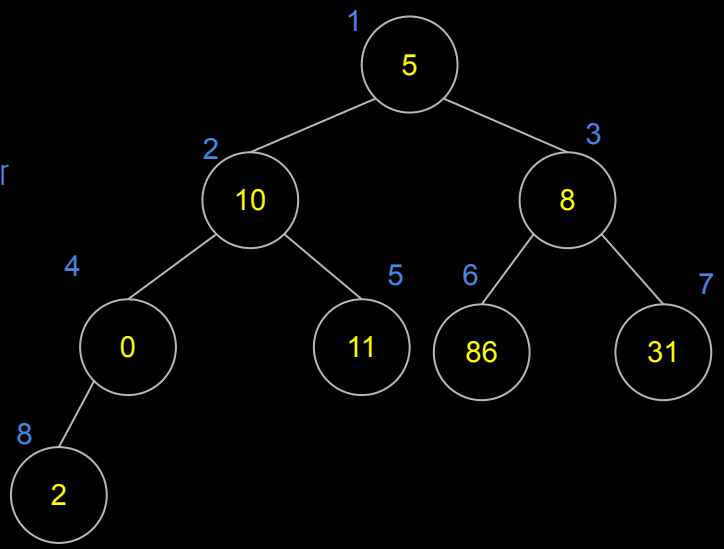
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	2

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	4



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

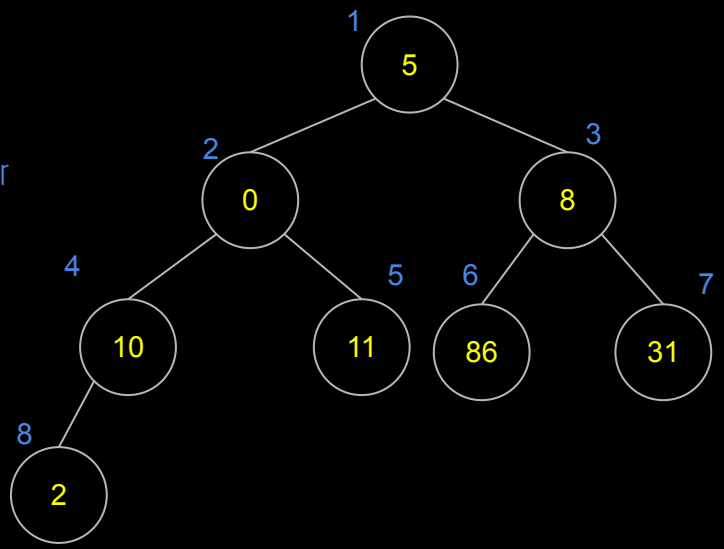
construir-min-heap(v,8)

n	i
8	2

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	4



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

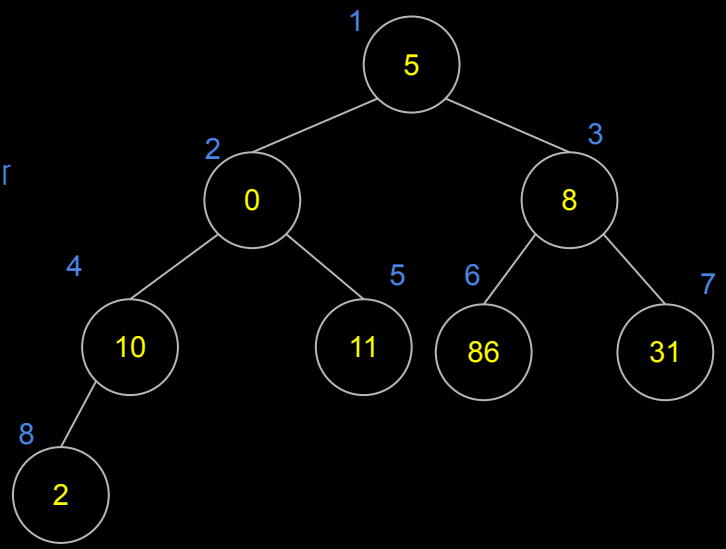
construir-min-heap(v,8)

n	i
8	2

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	4



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

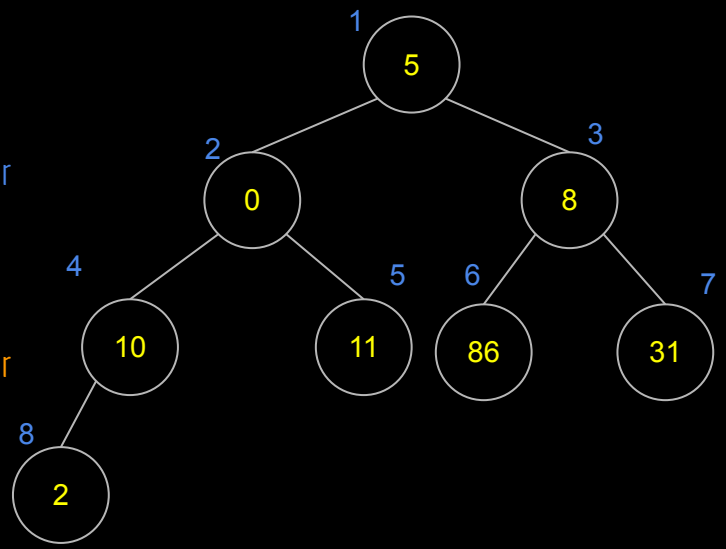
n	i
8	2

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	4

min-heapify(v,4,8)

n	i	l	r	menor
8	4	8	9	



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

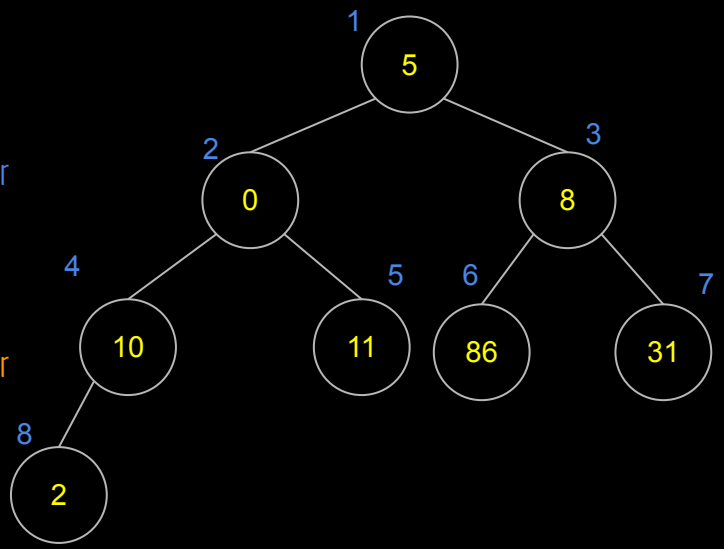
n	i
8	2

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	4

min-heapify(v,4,8)

n	i	l	r	menor
8	4	8	9	8



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

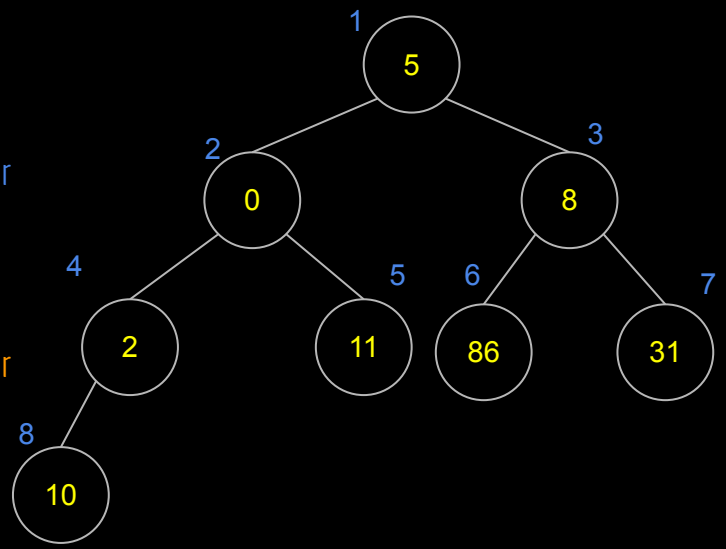
n	i
8	2

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	4

min-heapify(v,4,8)

n	i	l	r	menor
8	4	8	9	8



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

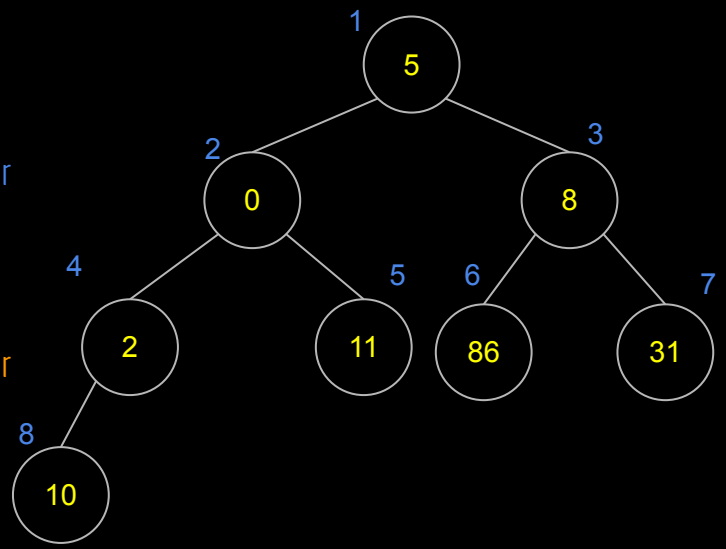
n	i
8	2

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	4

min-heapify(v,4,8)

n	i	l	r	menor
8	4	8	9	8



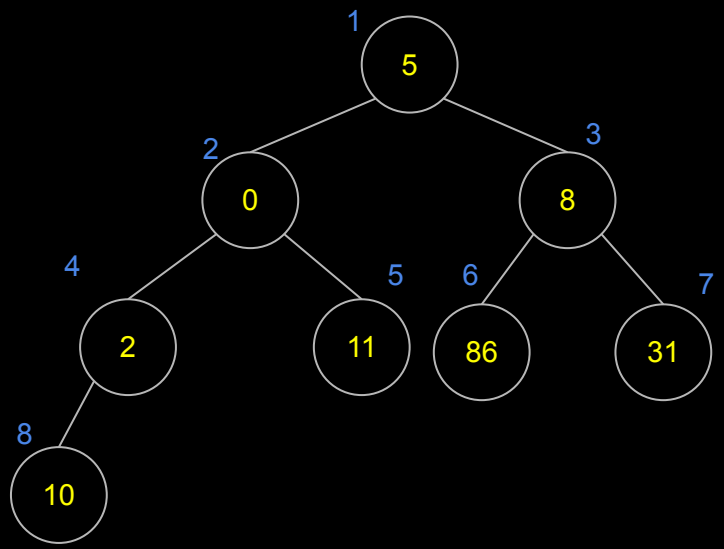
R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

construir-min-heap(v,8)

n	i
8	1

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```



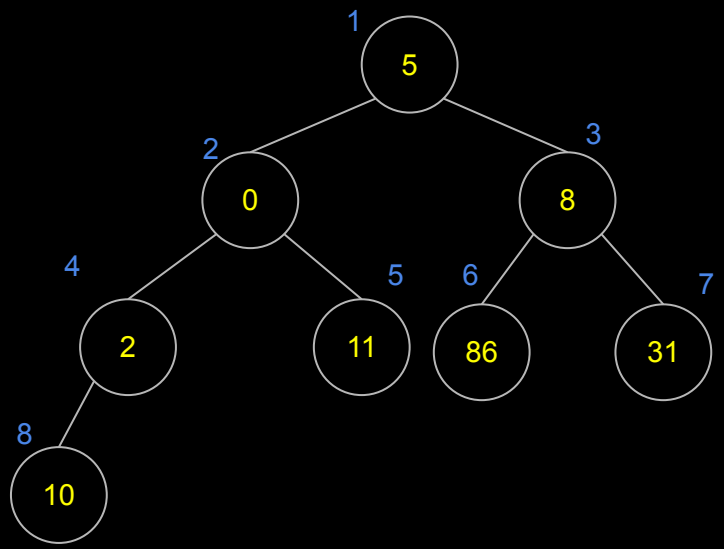
R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

construir-min-heap(v,8)

n	i
8	1

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

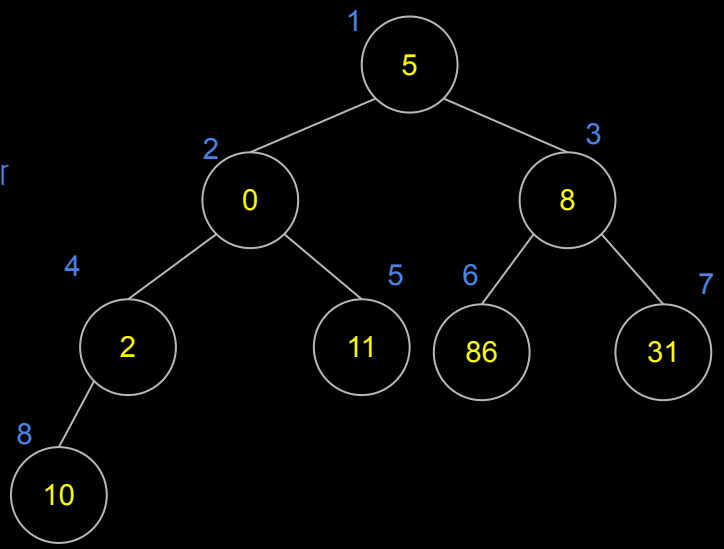
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	1

min-heapify(v,1,8)

n	i	l	r	menor
8	1	2	3	



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

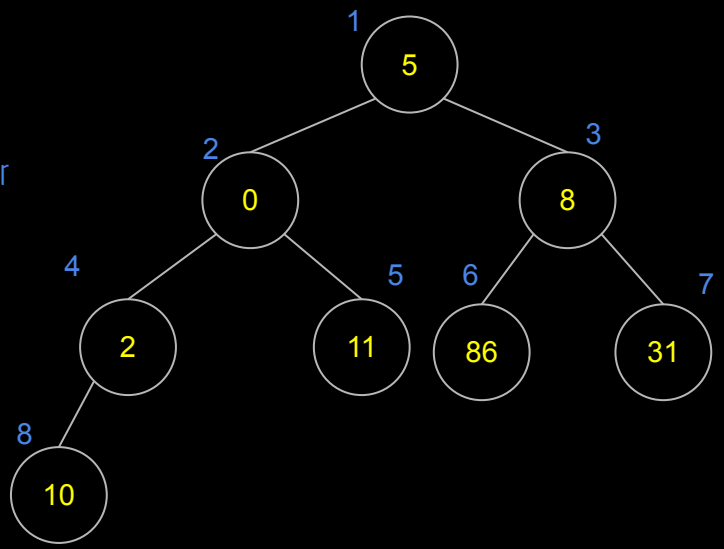
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	1

min-heapify(v,1,8)

n	i	l	r	menor
8	1	2	3	2



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

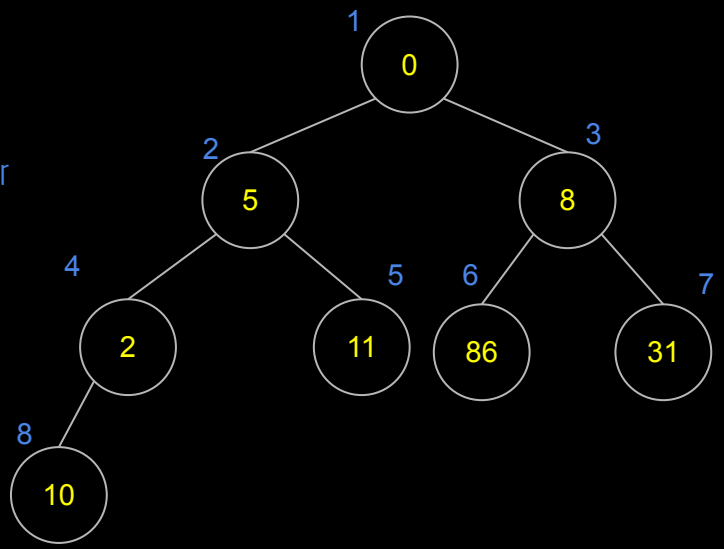
construir-min-heap(v,8)

n	i
8	1

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

min-heapify(v,1,8)

n	i	l	r	menor
8	1	2	3	2



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

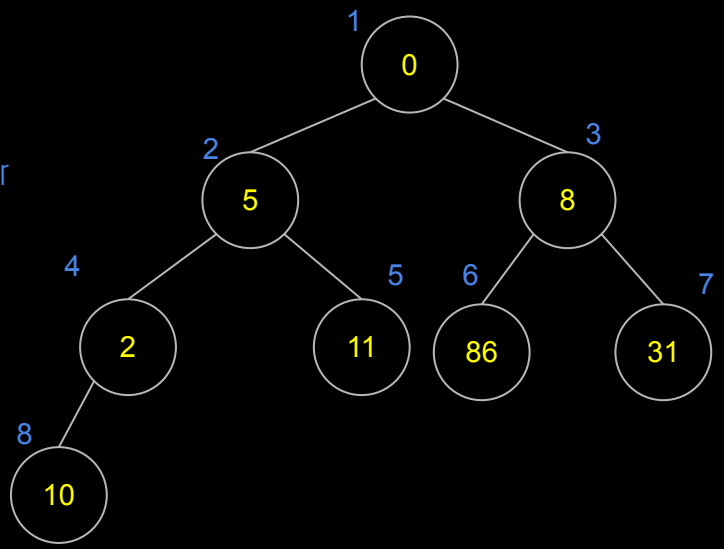
```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

n	i
8	1

min-heapify(v,1,8)

n	i	l	r	menor
8	1	2	3	2



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

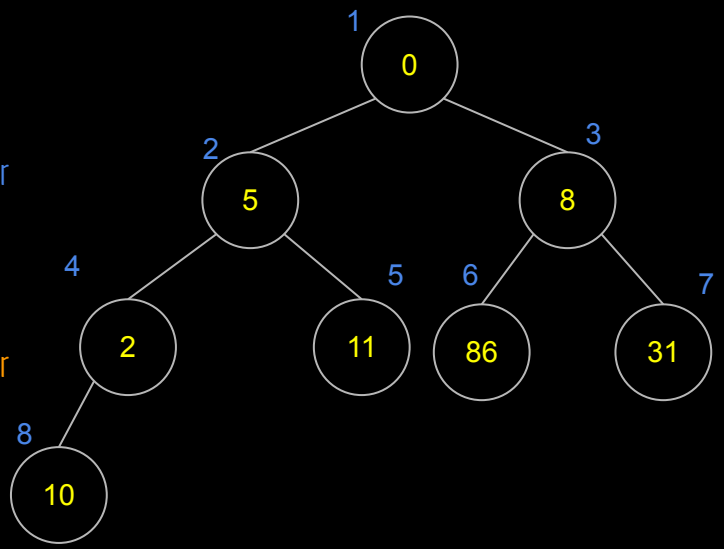
n	i
8	1

min-heapify(v,1,8)

n	i	l	r	menor
8	1	2	3	2

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

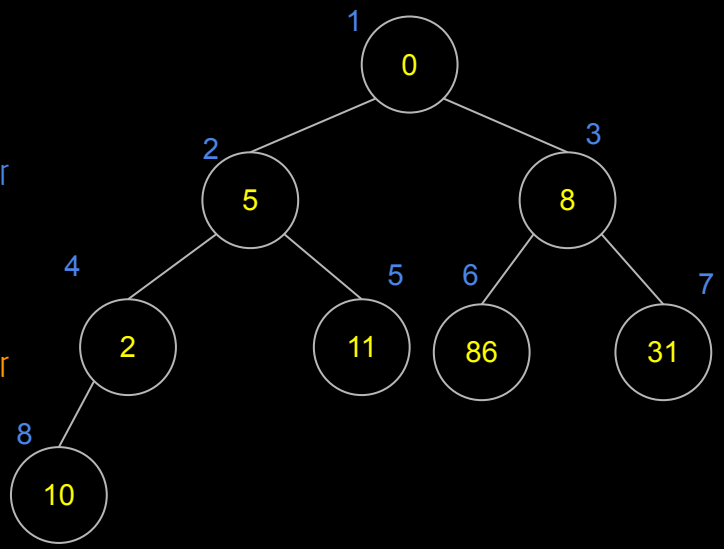
n	i
8	1

min-heapify(v,1,8)

n	i	l	r	menor
8	1	2	3	2

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	4



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

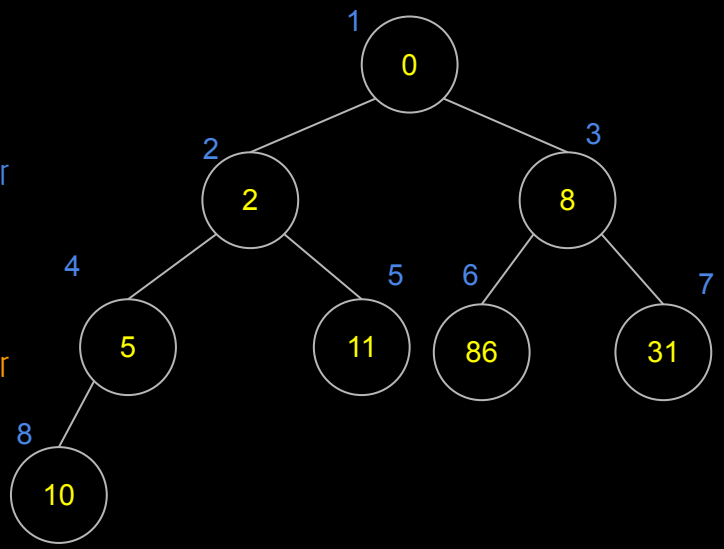
n	i
8	1

min-heapify(v,1,8)

n	i	l	r	menor
8	1	2	3	2

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	4



R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

```
função construir-min-heap(v,n)
para i = n/2 até 1 passo -1
      min-heapify(v,i,n)
```

```
função min-heapify(h,i,n)
l = esquerda(i)
r = direita(i)
se l ≤ n e h[l] < h[i]
      menor = l
senão
      menor = i
se r ≤ n e h[r] < h[menor]
      menor = r
se menor ≠ i
      trocar(h,i,menor)
      min-heapify(h,menor,n)
```

construir-min-heap(v,8)

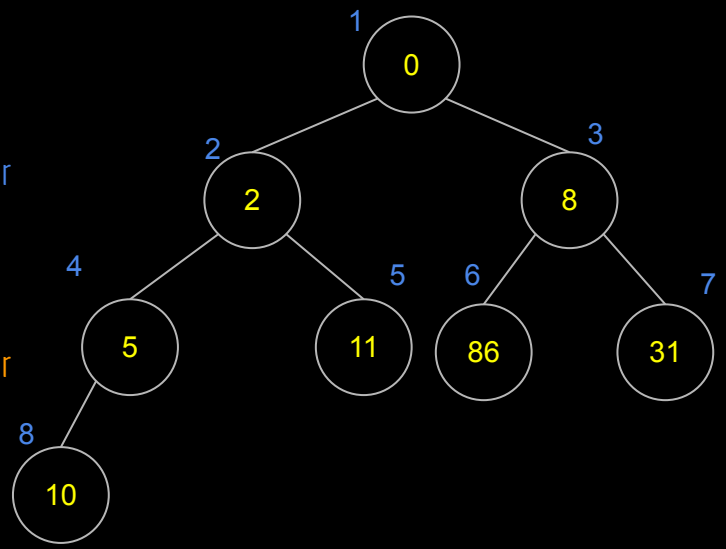
n	i
8	1

min-heapify(v,1,8)

n	i	l	r	menor
8	1	2	3	2

min-heapify(v,2,8)

n	i	l	r	menor
8	2	4	5	4

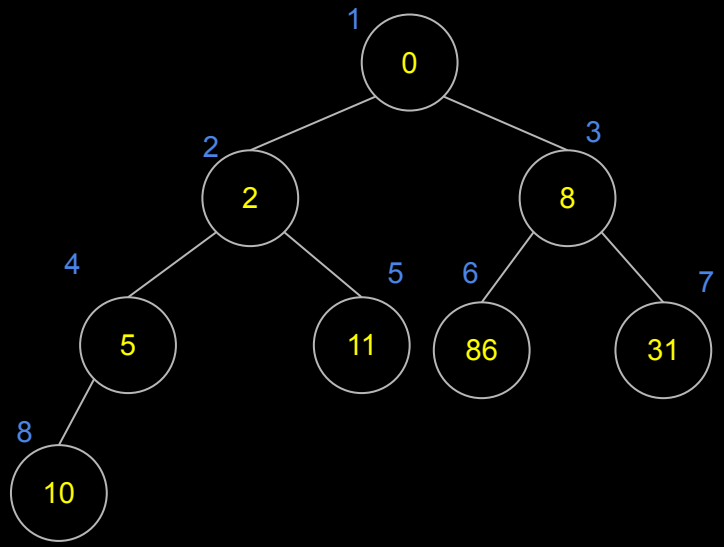


R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

Custo

Min-Heap construída.

Qual foi o custo?



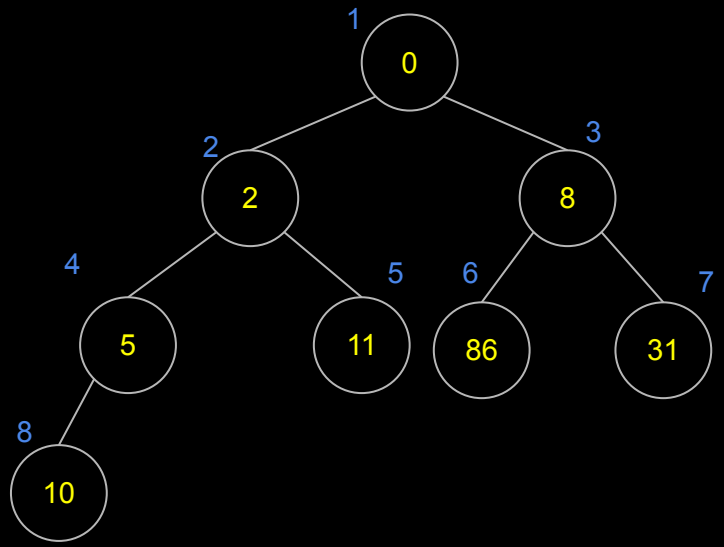
5 ← R_1	10 ← R_2	31 ← R_3	0 ← R_4	11 ← R_5	86 ← R_6	8 ← R_7	2 ← R_8
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

Custo

Min-Heap construída.

Qual foi o custo?

Custou P para copiar a cabeça de cada R_x para a heap,
+ P para construir a Min-Heap.
Logo, $C(P) = 2P$.



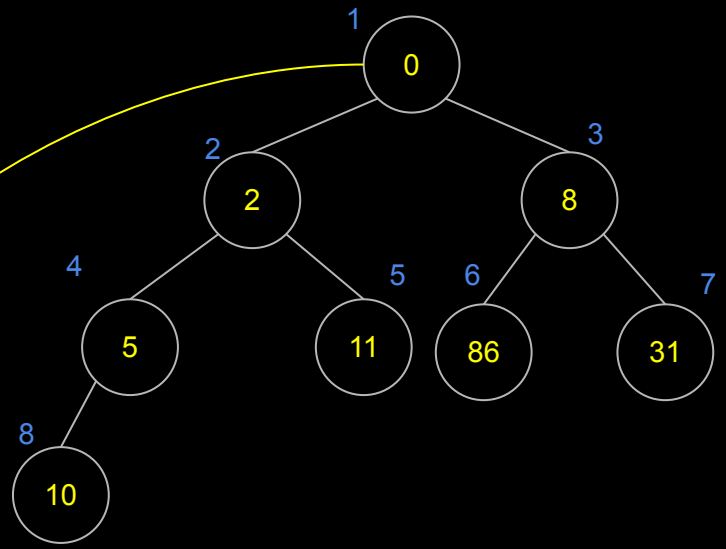
5	10	31	0	11	86	8	2
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85
← R_1	← R_2	← R_3	← R_4	← R_5	← R_6	← R_7	← R_8

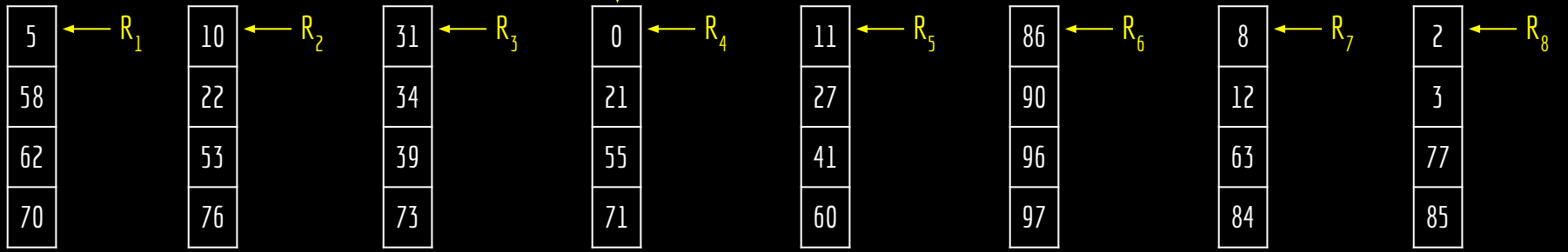
E agora?

Cada nodo da Min-Heap aponta para o seu arquivo.

Copiar a cabeça da Min-Heap para o arquivo final, mover o ponteiro do arquivo R_x correspondente, copiar a nova cabeça de R_x para a cabeça da Min-Heap, e chamar Min-Heapify com um custo $O(log_2 P)$.

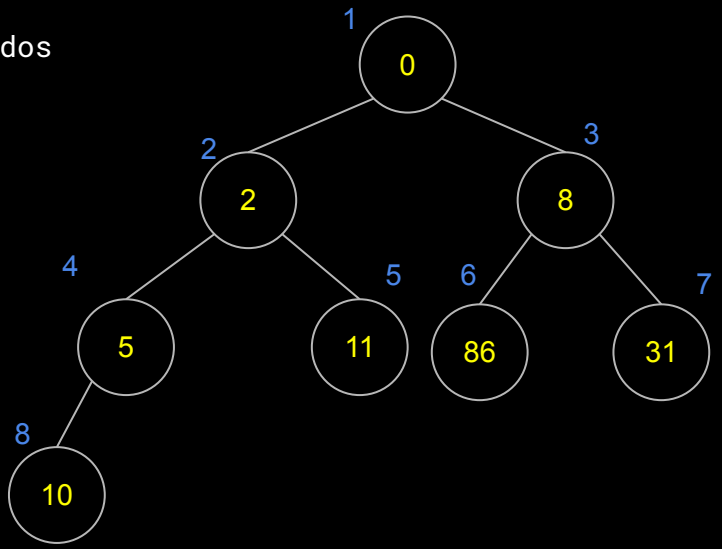
Repetir o processo.





E agora?

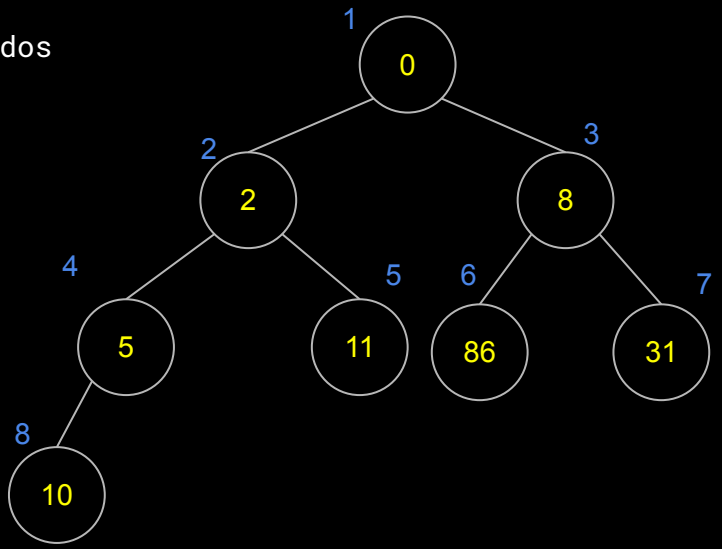
Copiar dado da cabeça da heap
Ajustar Ponteiros
Substituir cabeça da heap
Min-Heapify
repetir enquanto existirem dados



5 ← R_1	10 ← R_2	31 ← R_3	0 ← R_4	11 ← R_5	86 ← R_6	8 ← R_7	2 ← R_8
58	22	34	21	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

E agora?

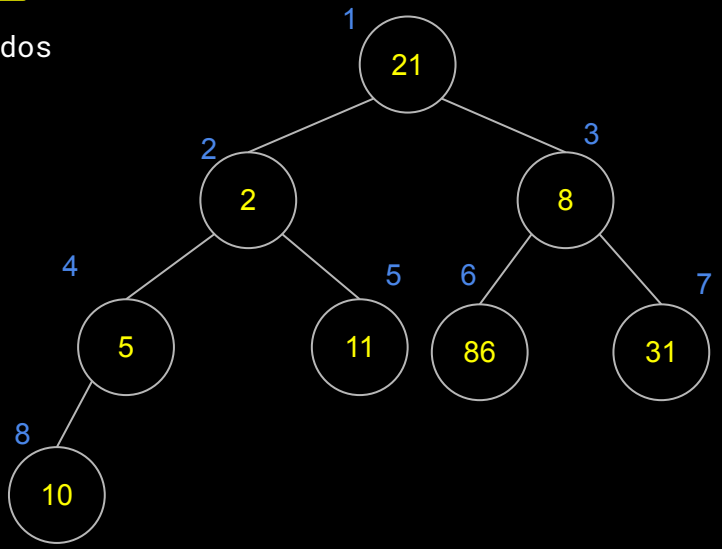
Copiar dado da cabeça da heap
Ajustar Ponteiros
Substituir cabeça da heap
Min-Heapify
repetir enquanto existirem dados

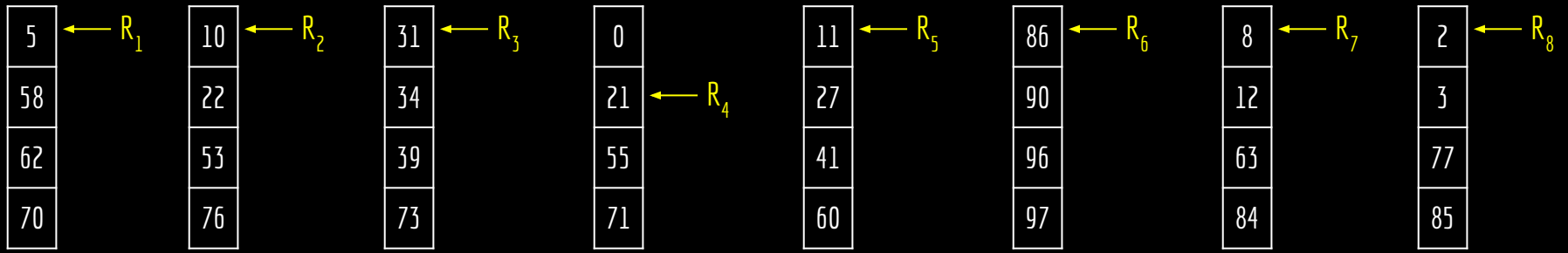


5 ← R_1	10 ← R_2	31 ← R_3	0	11 ← R_5	86 ← R_6	8 ← R_7	2 ← R_8
58	22	34	21 ← R_4	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

E agora?

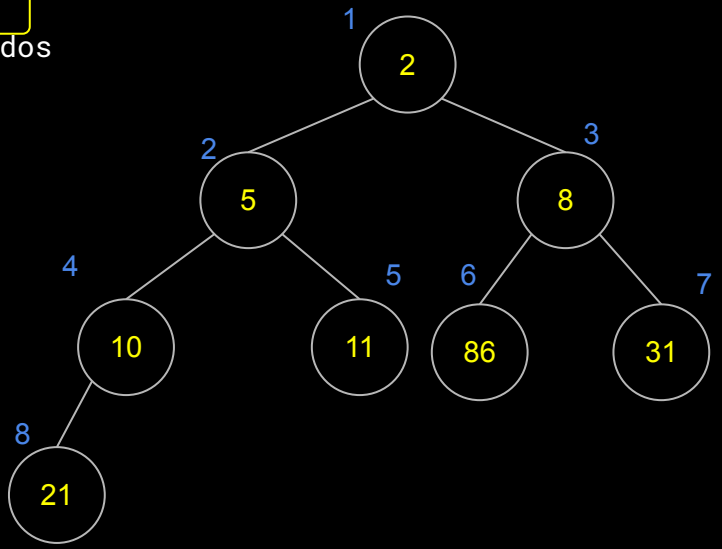
Copiar dado da cabeça da heap
Ajustar Ponteiros
Substituir cabeça da heap
Min-Heapify
repetir enquanto existirem dados

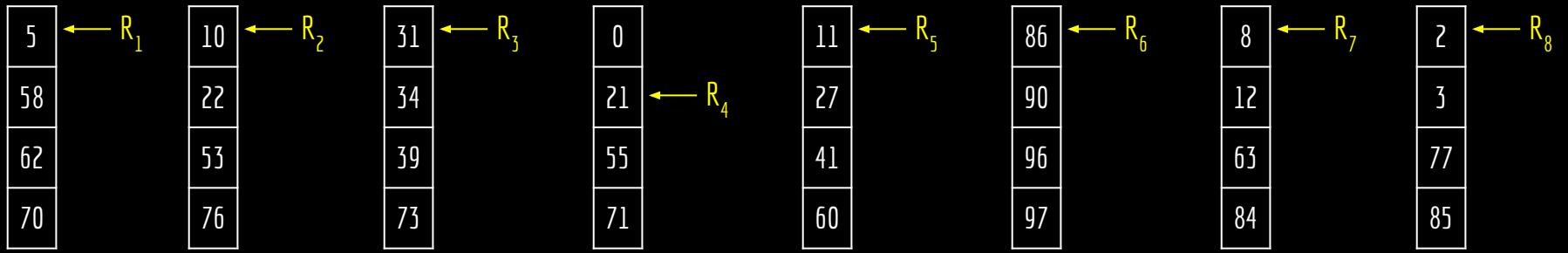




E agora?

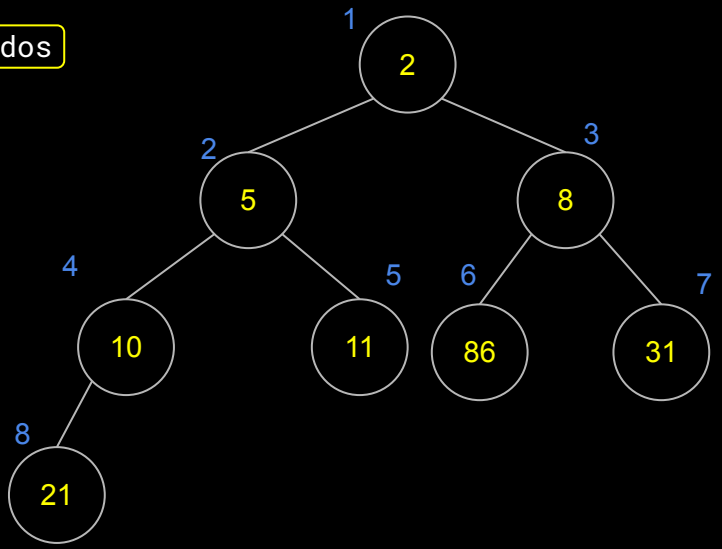
Copiar dado da cabeça da heap
Ajustar Ponteiros
Substituir cabeça da heap
Min-Heapify
repetir enquanto existirem dados





E agora?

Copiar dado da cabeça da heap
Ajustar Ponteiros
Substituir cabeça da heap
Min-Heapify
repetir enquanto existirem dados



S

0

5 ← R_1	10 ← R_2	31 ← R_3	0	11 ← R_5	86 ← R_6	8 ← R_7	2 ← R_8
58	22	34	21 ← R_4	27	90	12	3
62	53	39	55	41	96	63	77
70	76	73	71	60	97	84	85

Seleção por Substituição

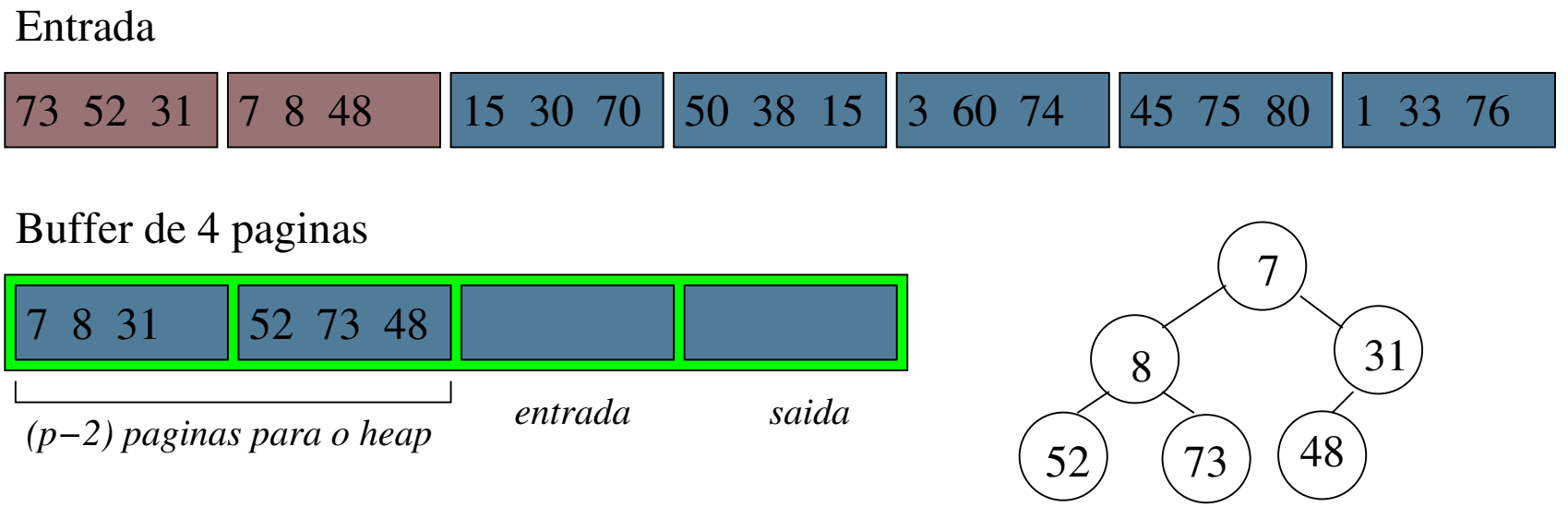
- Estratégia para diminuir a quantidade de conjuntos ordenados no Passo de Ordenação
- Utiliza a estrutura de heap (ao invés do método de seleção)
- Segundo Knuth:
  Para números randômicos o tamanho dos conjuntos ordenados é em média $2 * b$

Passos da Seleção por Substituição

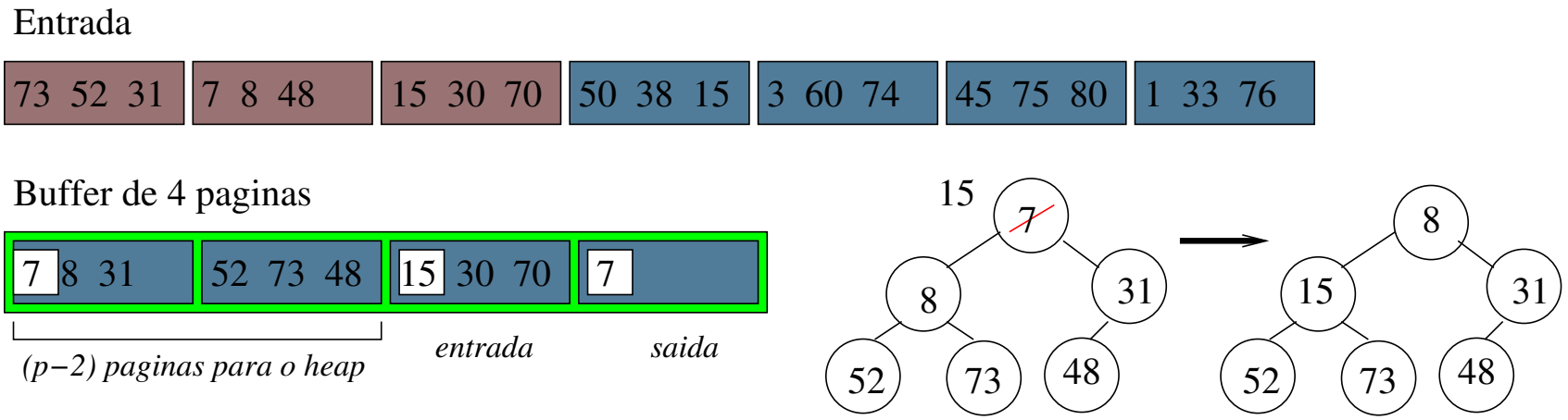
Idéia:

- Mantém os registros no buffer em ordem min-heap
- Retira o menor e insere o próximo registro x
- Se x for menor que o último elemento retirado, ele é marcado como maior que todos os demais no heap
- Quando todos os elementos no heap estiverem marcados, inicia-se um novo conjunto ordenado.

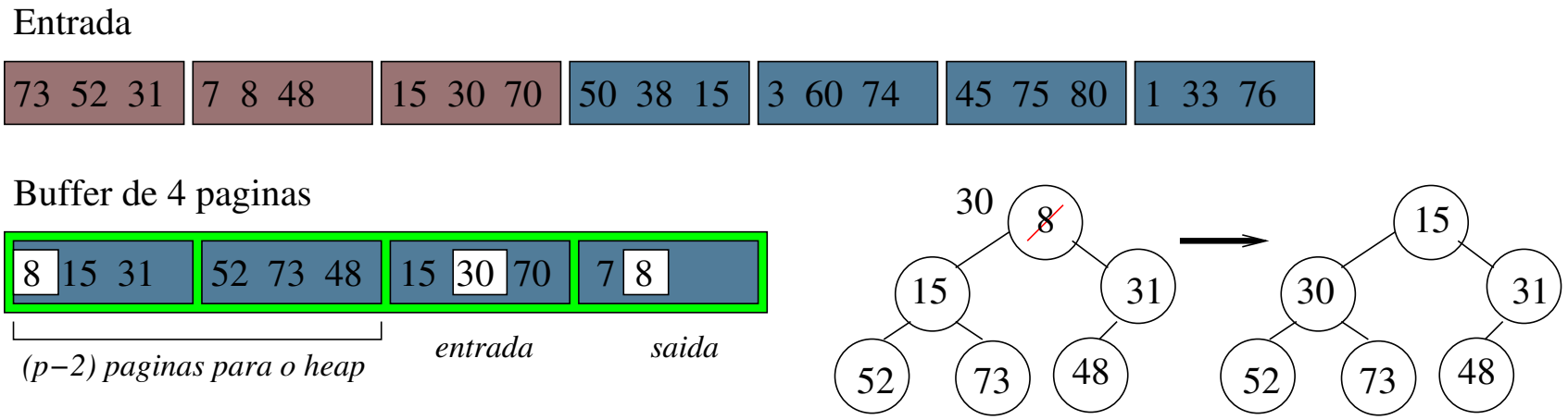
Exemplo


Entrada
73 52 31
7 8 48
15 30 70
50 38 15
3 60 74
45 75 80
1 33 76
Buffer de 4 paginas
7 8 31
52 73 48
(p−2) paginas para o heap
entrada
saida
7
8
31
52
73
48

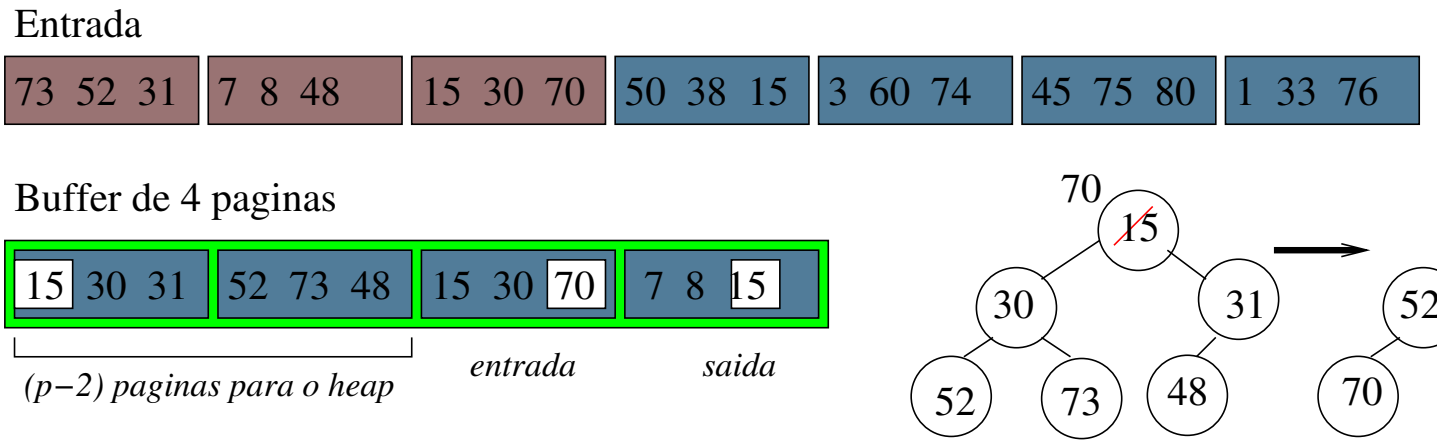
Exemplo


Entrada
73 52 31
7 8 48
15 30 70
50 38 15
3 60 74
45 75 80
1 33 76
Buffer de 4 paginas
7 8 31
52 73 48
15 30 70
7
(p−2) paginas para o heap
entrada
saida
15
7
8
31
52
73
48
8
15
31
52
73
48

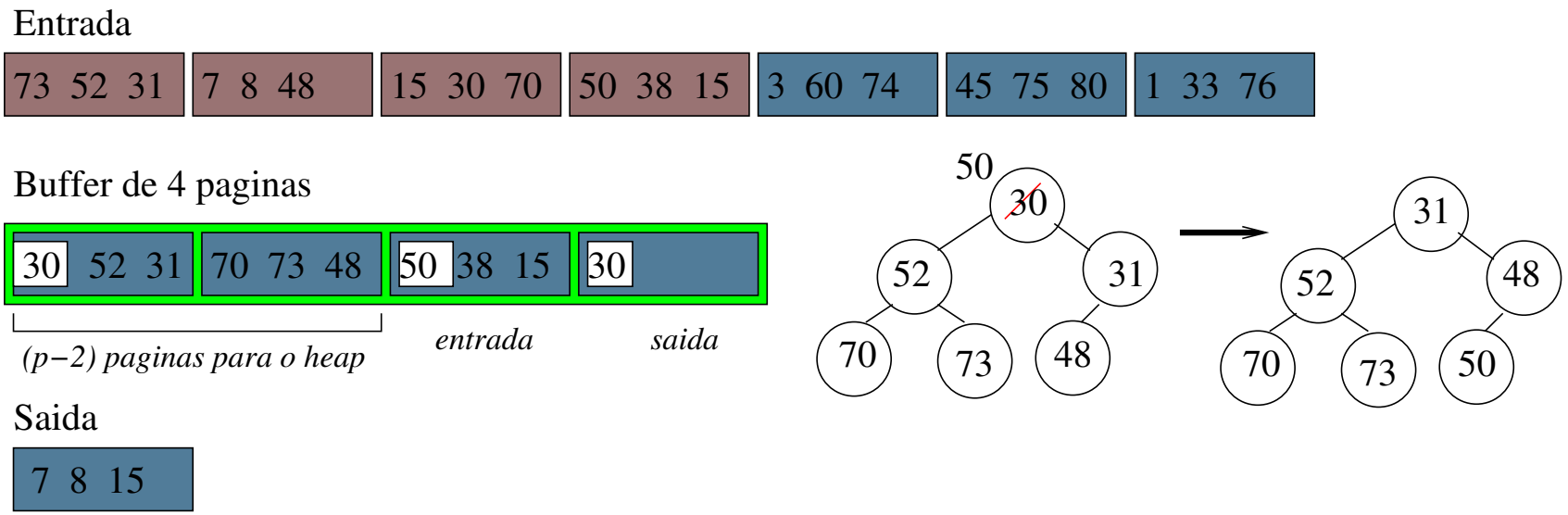
Exemplo


Entrada
73 52 31
7 8 48
15 30 70
50 38 15
3 60 74
45 75 80
1 33 76
Buffer de 4 paginas
8 15 31
52 73 48
15 30 70
7 8
(p−2) paginas para o heap
entrada
saida
30
8
15
31
52
73
48
15
30
31
52
73
48

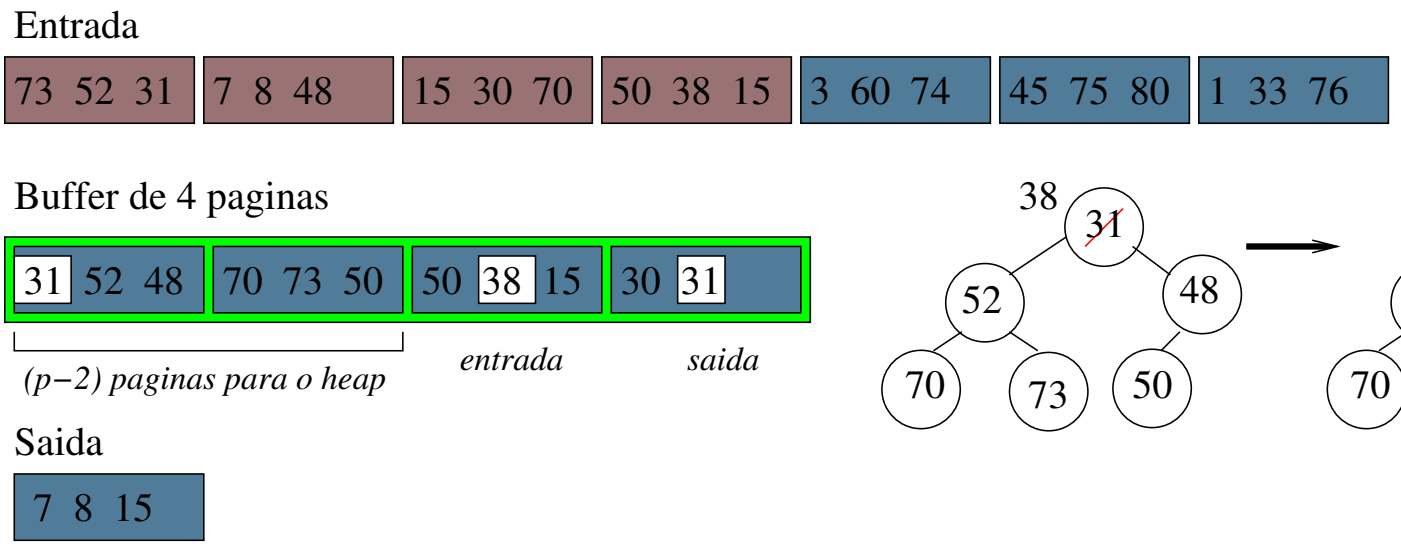
Exemplo


Entrada
73 52 31
7 8 48
15 30 70
50 38 15
3 60 74
45 75 80
1 33 76
Buffer de 4 paginas
15 30 31
52 73 48
15 30 70
7 8 15
(p−2) paginas para o heap
entrada
saida
70
15
30
31
52
73
48
30
52
31
70
73
48

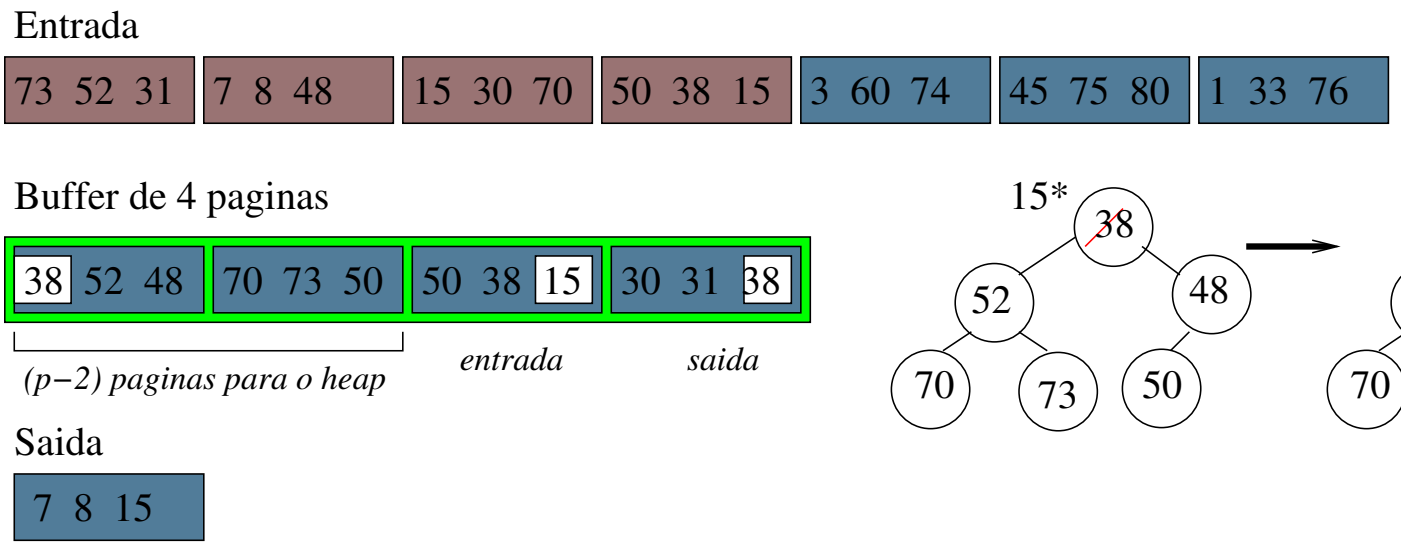
Exemplo


Entrada
73 52 31
7 8 48
15 30 70
50 38 15
3 60 74
45 75 80
1 33 76
Buffer de 4 paginas
30 52 31
70 73 48
50 38 15
30
(p−2) paginas para o heap
entrada
saida
Saida
7 8 15
50
30
52
31
70
73
48
31
52
48
70
73
50

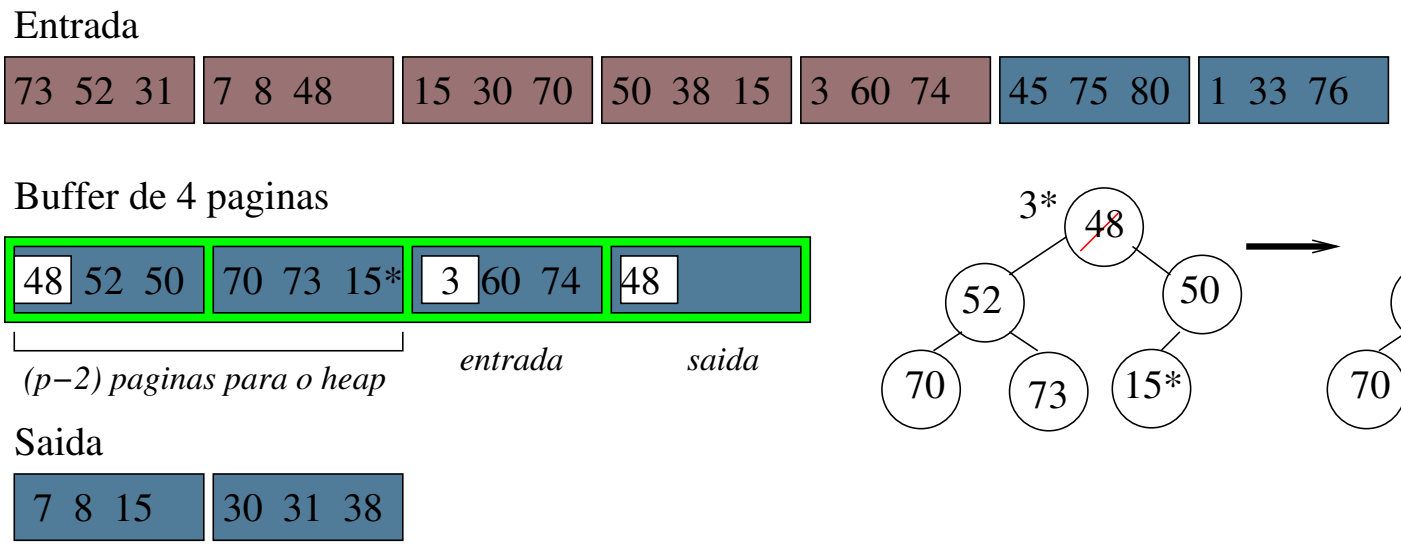
Exemplo


Entrada
73 52 31
7 8 48
15 30 70
50 38 15
3 60 74
45 75 80
1 33 76
Buffer de 4 paginas
31 52 48
70 73 50
50 38 15
30 31
(p−2) paginas para o heap
entrada
saida
Saida
7 8 15
38
31
52
48
70
73
50
38
52
48
70
73
50

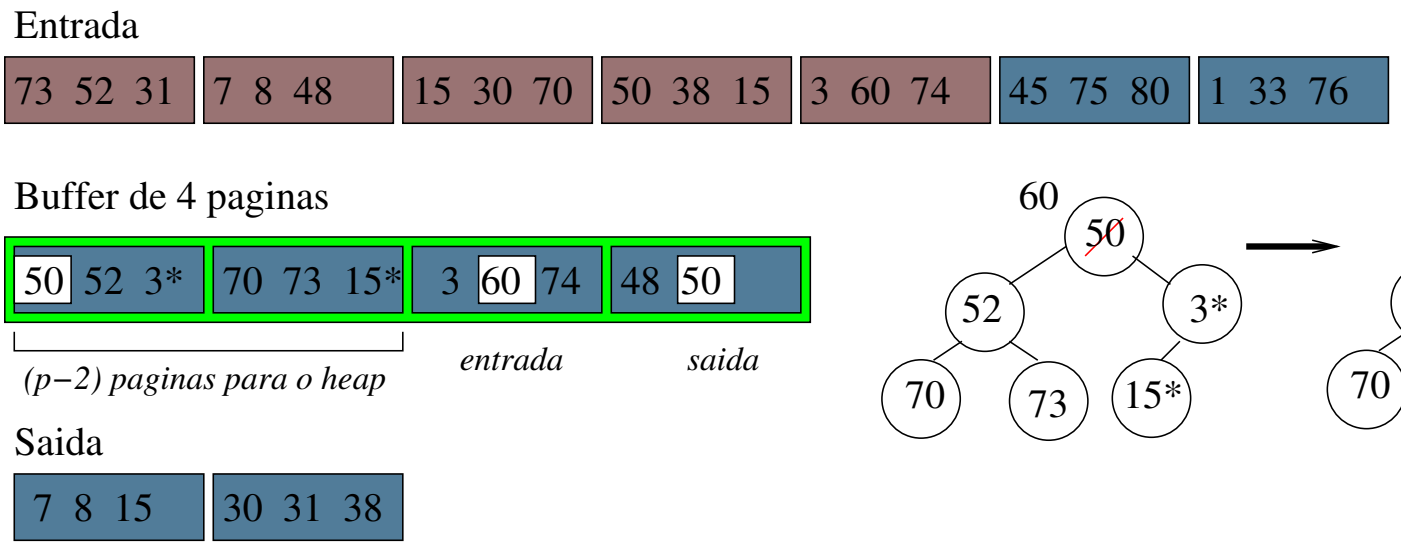
Exemplo


Entrada
73 52 31
7 8 48
15 30 70
50 38 15
3 60 74
45 75 80
1 33 76
Buffer de 4 paginas
38 52 48
70 73 50
50 38 15
30 31 38
(p−2) paginas para o heap
entrada
saida
Saida
7 8 15
15*
38
52
48
70
73
50
48
52
50
70
73
15*

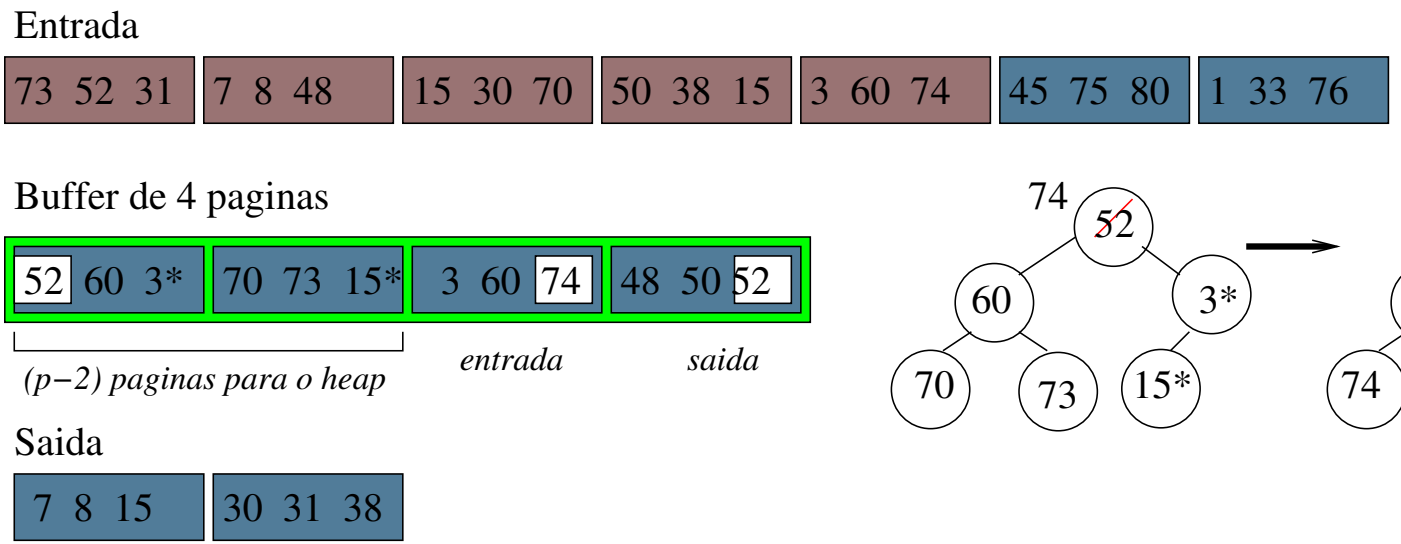
Exemplo


Entrada
73 52 31
7 8 48
15 30 70
50 38 15
3 60 74
45 75 80
1 33 76
Buffer de 4 paginas
48 52 50
70 73 15*
3 60 74
48
(p−2) paginas para o heap
entrada
saida
Saida
7 8 15
30 31 38
3*
48
52
50
70
73
15*
50
52
3*
70
73
15*

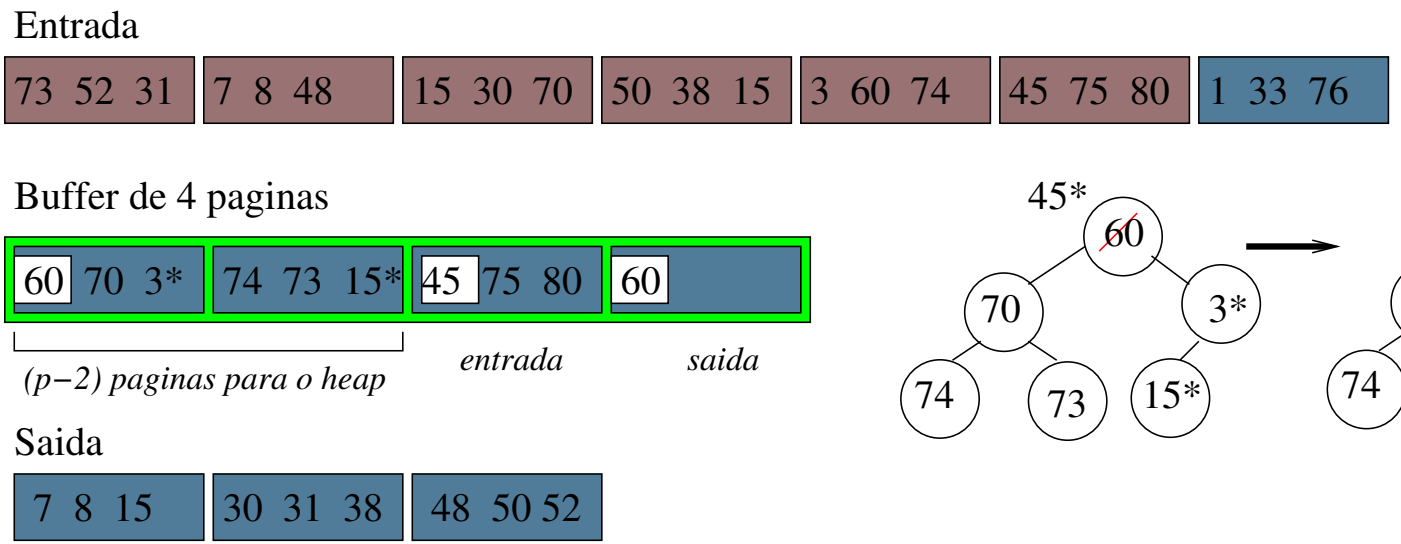
Exemplo


Entrada
73 52 31
7 8 48
15 30 70
50 38 15
3 60 74
45 75 80
1 33 76
Buffer de 4 paginas
50 52 3*
70 73 15*
3 60 74
48 50
(p−2) paginas para o heap
entrada
saida
Saida
7 8 15
30 31 38
60
50
52
3*
70
73
15*
52
60
3*
70
73
15*

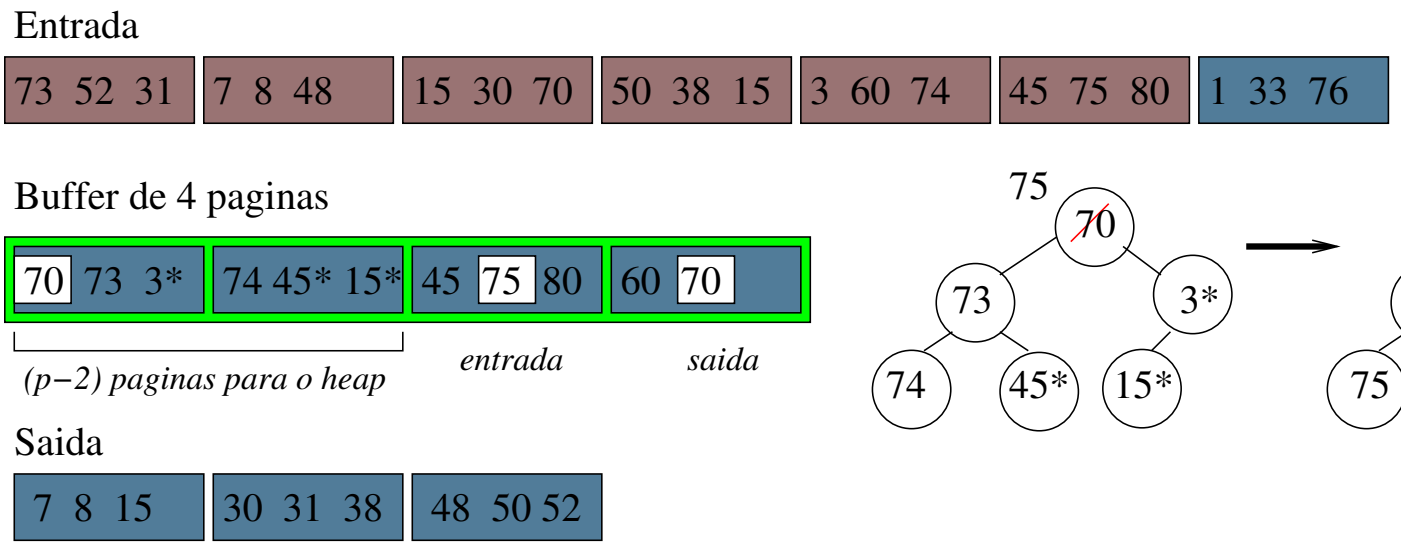
Exemplo


Entrada
73 52 31
7 8 48
15 30 70
50 38 15
3 60 74
45 75 80
1 33 76
Buffer de 4 paginas
52 60 3*
70 73 15*
3 60 74
48 50 52
(p−2) paginas para o heap
entrada
saida
Saida
7 8 15
30 31 38
74
52
60
3*
70
73
15*
60
70
3*
74
73
15*

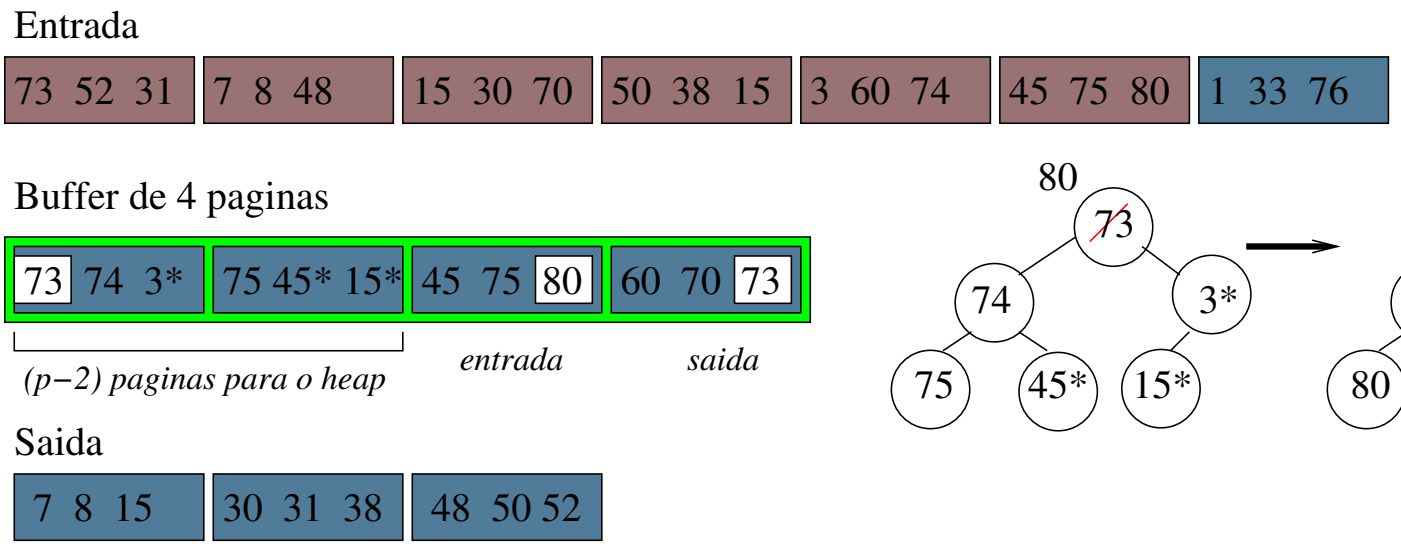
Exemplo


Entrada
73 52 31
7 8 48
15 30 70
50 38 15
3 60 74
45 75 80
1 33 76
Buffer de 4 paginas
60 70 3*
74 73 15*
45 75 80
60
(p−2) paginas para o heap
entrada
saida
Saida
7 8 15
30 31 38
48 50 52
45*
60
70
3*
74
73
15*
70
73
3*
74
45*
15*

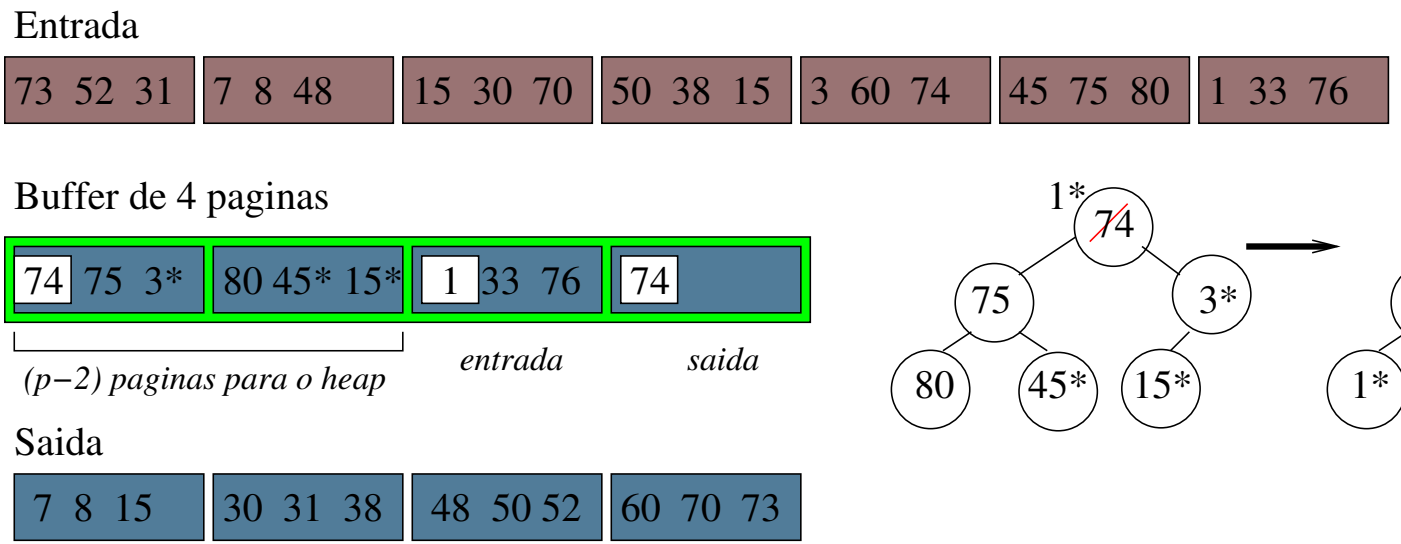
Exemplo


Entrada
73 52 31
7 8 48
15 30 70
50 38 15
3 60 74
45 75 80
1 33 76
Buffer de 4 paginas
70 73 3*
74 45* 15*
45 75 80
60 70
(p−2) paginas para o heap
entrada
saida
Saida
7 8 15
30 31 38
48 50 52
75
70
73
3*
74
45*
15*
73
74
3*
75
45*
15*

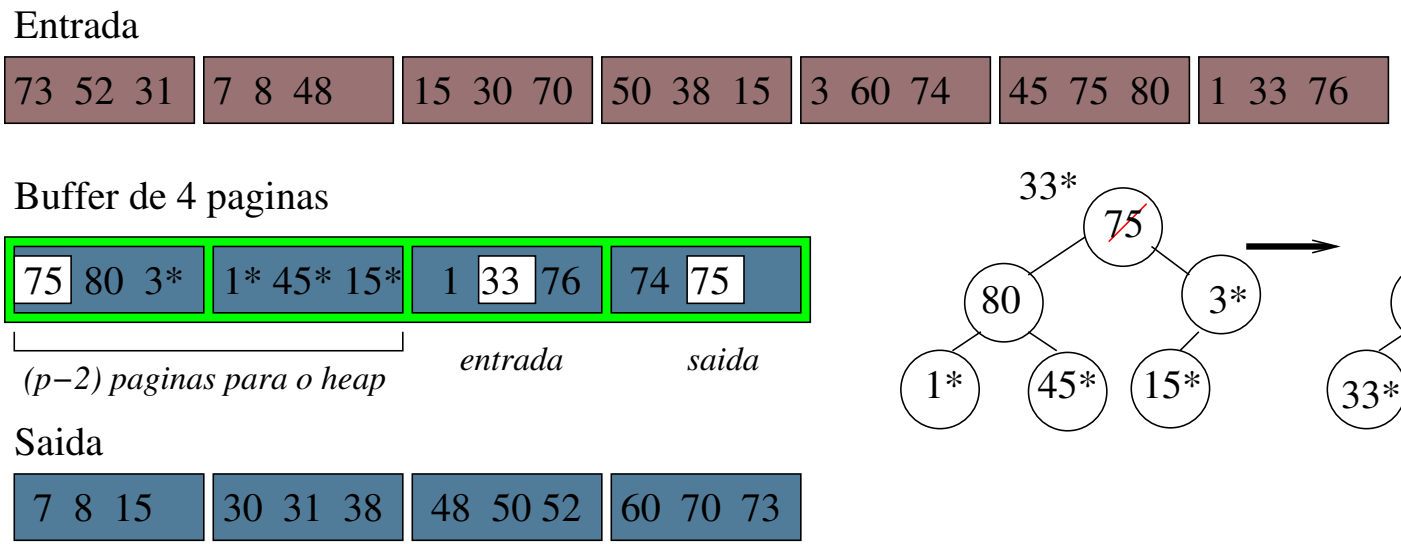
Exemplo


Entrada
73 52 31
7 8 48
15 30 70
50 38 15
3 60 74
45 75 80
1 33 76
Buffer de 4 paginas
73 74 3*
75 45* 15*
45 75 80
60 70 73
(p−2) paginas para o heap
entrada
saida
Saida
7 8 15
30 31 38
48 50 52
80
73
74
3*
75
45*
15*
74
75
3*
80
45*
15*

Exemplo

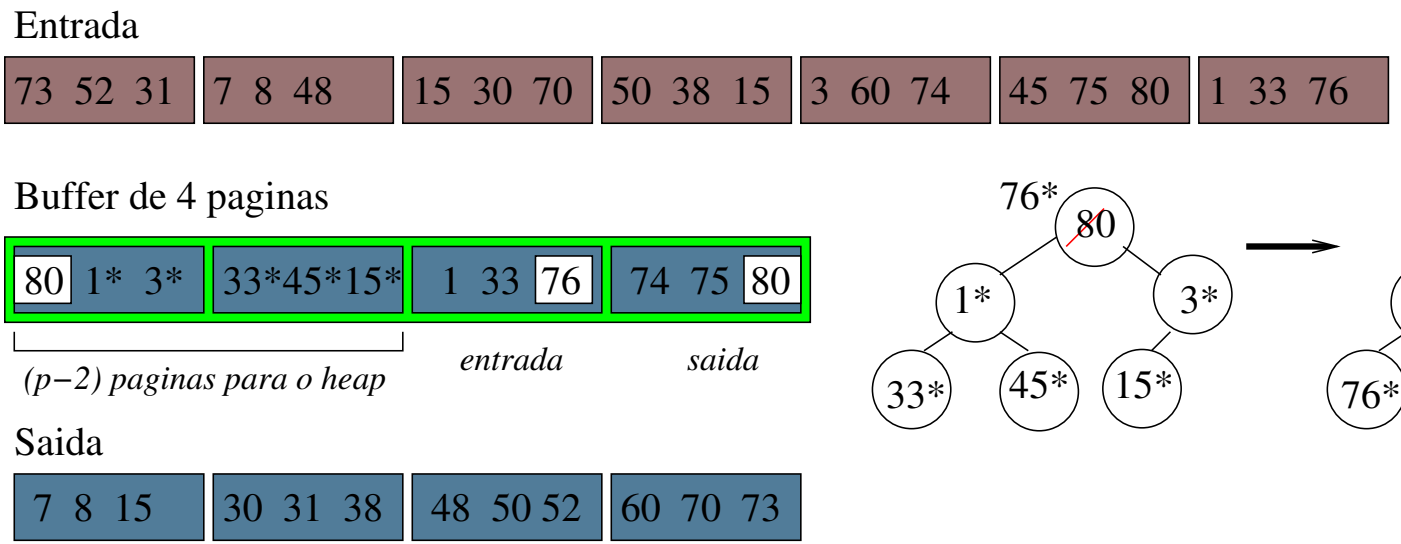

Entrada
73 52 31
7 8 48
15 30 70
50 38 15
3 60 74
45 75 80
1 33 76
Buffer de 4 paginas
74 75 3*
80 45* 15*
1 33 76
74
(p−2) paginas para o heap
entrada
saida
Saida
7 8 15
30 31 38
48 50 52
60 70 73
1*
74
75
3*
80
45*
15*
75
80
3*
1*
45*
15*

Exemplo

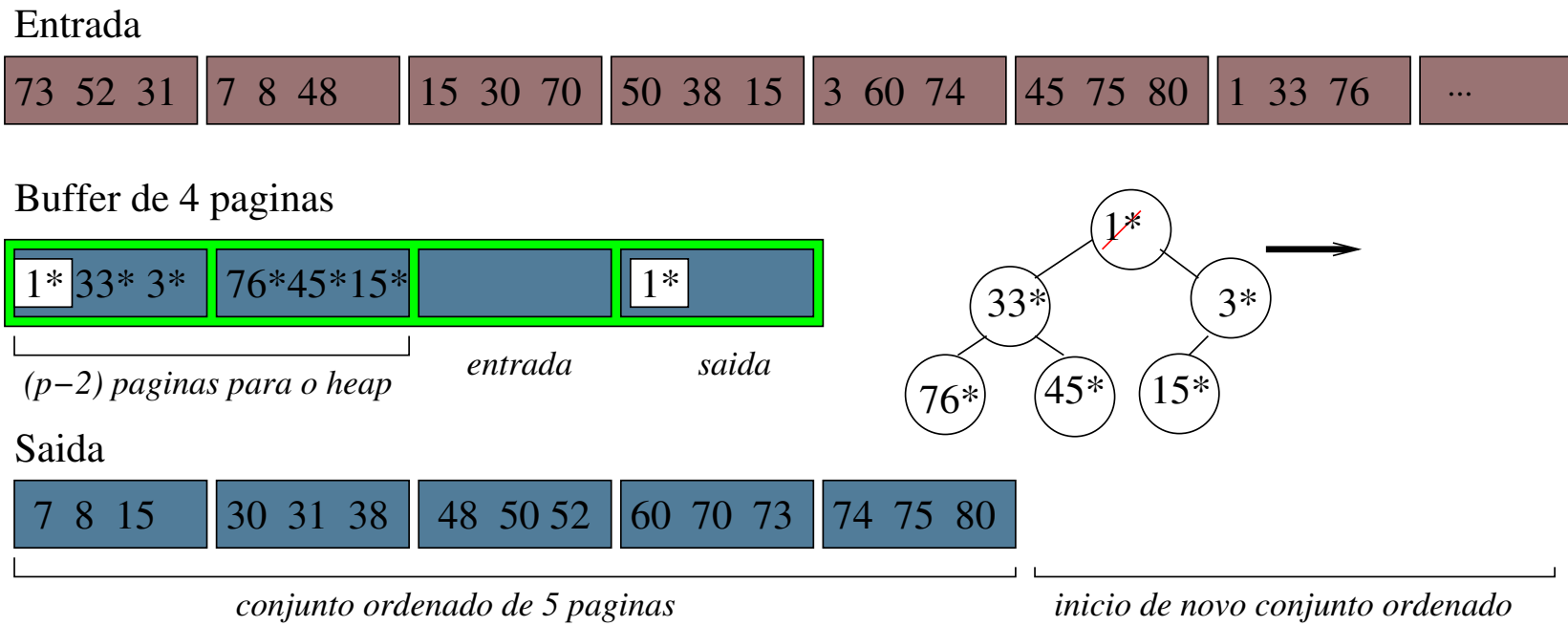

Entrada
73 52 31
7 8 48
15 30 70
50 38 15
3 60 74
45 75 80
1 33 76
Buffer de 4 paginas
75 80 3*
1* 45* 15*
1 33 76
74 75
(p−2) paginas para o heap
entrada
saida
Saida
7 8 15
30 31 38
48 50 52
60 70 73

33*
75
80
3*
1*
45*
15*
80
1*
3*
33*
45*
15*

Exemplo


Entrada
73 52 31
7 8 48
15 30 70
50 38 15
3 60 74
45 75 80
1 33 76
Buffer de 4 paginas
80 1* 3*
33*45*15*
1 33 76
74 75 80
(p−2) paginas para o heap
entrada
saida
Saida
7 8 15
30 31 38
48 50 52
60 70 73
76*
80
1*
3*
33*
45*
15*
1*
33*
3*
76*
45*
15*

Exemplo


Entrada
73 52 31
7 8 48
15 30 70
50 38 15
3 60 74
45 75 80
1 33 76
...
Buffer de 4 paginas
1* 33* 3*
76*45*15*
1*
(p−2) paginas para o heap
entrada
saida
1*
33*
3*
76*
45*
15*
Saida
7 8 15
30 31 38
48 50 52
60 70 73
74 75 80
conjunto ordenado de 5 paginas
inicio de novo conjunto ordenado

Referências

- Livro Ziviani, Seção 4.2
- D. Knuth, *The Art of Computer Programming: Volume 3: Sorting and Searching*, Ed. Addison-Wesley, 1998