ORDENAÇÃO

Aula 22 - 16 de junho de 2009

Mergesort

Mergesort

Veremos, pela primeira vez, um algoritmo ótimo para o problema da ordenação usando o modelo de comparação de chaves: o *mergesort*.

Mergesort

Veremos, pela primeira vez, um algoritmo ótimo para o problema da ordenação usando o modelo de comparação de chaves: o *mergesort*.

Segundo Donald Knuth, o matemático John von Neumann, um dos pioneiros da computação, foi o inventor do *mergesort*. Ele implementou o *mergesort* em 1945 no computador EDVAC.

Mergesort

Veremos, pela primeira vez, um algoritmo ótimo para o problema da ordenação usando o modelo de comparação de chaves: o *mergesort*.

Segundo Donald Knuth, o matemático John von Neumann, um dos pioneiros da computação, foi o inventor do *mergesort*. Ele implementou o *mergesort* em 1945 no computador EDVAC.

O *mergesort* é, de alguma forma, semelhante ao *quicksort*, pois ambos são fundamentados no mesmo paradigma de resolução de problemas: o popular paradigma **divisão e conquista**.

Mergesort

Mergesort

Usando divisão e conquista, o *mergesort* pode ser descrito como:

Mergesort

Usando divisão e conquista, o *mergesort* pode ser descrito como:

- **Divida**: divida a seqüência de n elementos a ser ordenada em duas subseqüências, cada qual com $n/2$ elementos.

Mergesort

Usando divisão e conquista, o *mergesort* pode ser descrito como:

- **Divida**: divida a seqüência de n elementos a ser ordenada em duas subseqüências, cada qual com $n/2$ elementos.

- **Conquiste**: ordene as duas subseqüências, recursivamente, usando *mergesort*.

Mergesort

Usando divisão e conquista, o *mergesort* pode ser descrito como:

- **Divida**: divida a seqüência de n elementos a ser ordenada em duas subseqüências, cada qual com $n/2$ elementos.

- **Conquiste**: ordene as duas subseqüências, recursivamente, usando *mergesort*.

- **Combine**: intercale as duas subseqüências ordenadas anteriormente para gerar a seqüência de n elementos ordenada.

Mergesort

Mergesort

A condição de parada da recursão do *mergesort* ocorre quando a seqüência a ser ordenada possui tamanho 1, pois ela já está ordenada.

Mergesort

A condição de parada da recursão do *mergesort* ocorre quando a seqüência a ser ordenada possui tamanho 1, pois ela já está ordenada.

(01) **algoritmo** *mergesort*(**ref** A, n)
(02)   *mergesort_aux*$(A, 1, n)$
(03) **fimalgoritmo**

A condição de parada da recursão do *mergesort* ocorre quando a seqüência a ser ordenada possui tamanho 1, pois ela já está ordenada.

(01) **algoritmo** *mergesort_aux*(**ref** A, l, r)
(02) **se** $l < r$ **então**
(03) $m \leftarrow \left\lfloor \frac{(l+r)}{2} \right\rfloor$
(04) *mergesort_aux*(A, l, m)
(05) *mergesort_aux*$(A, m+1, r)$
(06) *intercale*(A, l, m, r)
(07) **fimse**
(08) **fimalgoritmo**

Mergesort

Mergesort

Exemplo:

Mergesort

Exemplo:

$l = 1$ $r = 8$

5	2	4	7	1	3	8	6

Mergesort

Exemplo:

$l = 1$ $\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ $r = 8$

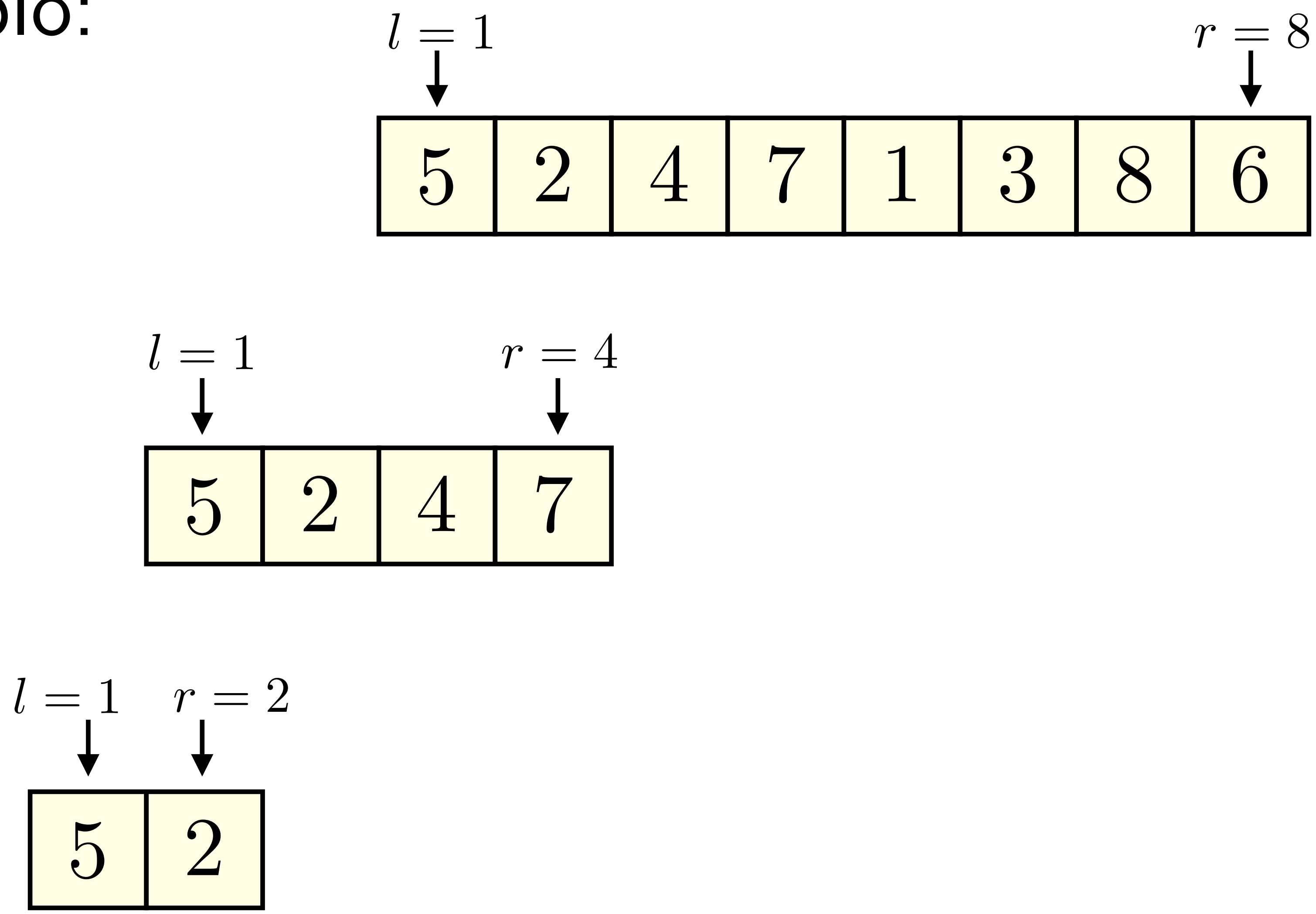
5	2	4	7	1	3	8	6

$l = 1$ $\quad\quad\quad\quad\quad\quad$ $r = 4$

5	2	4	7

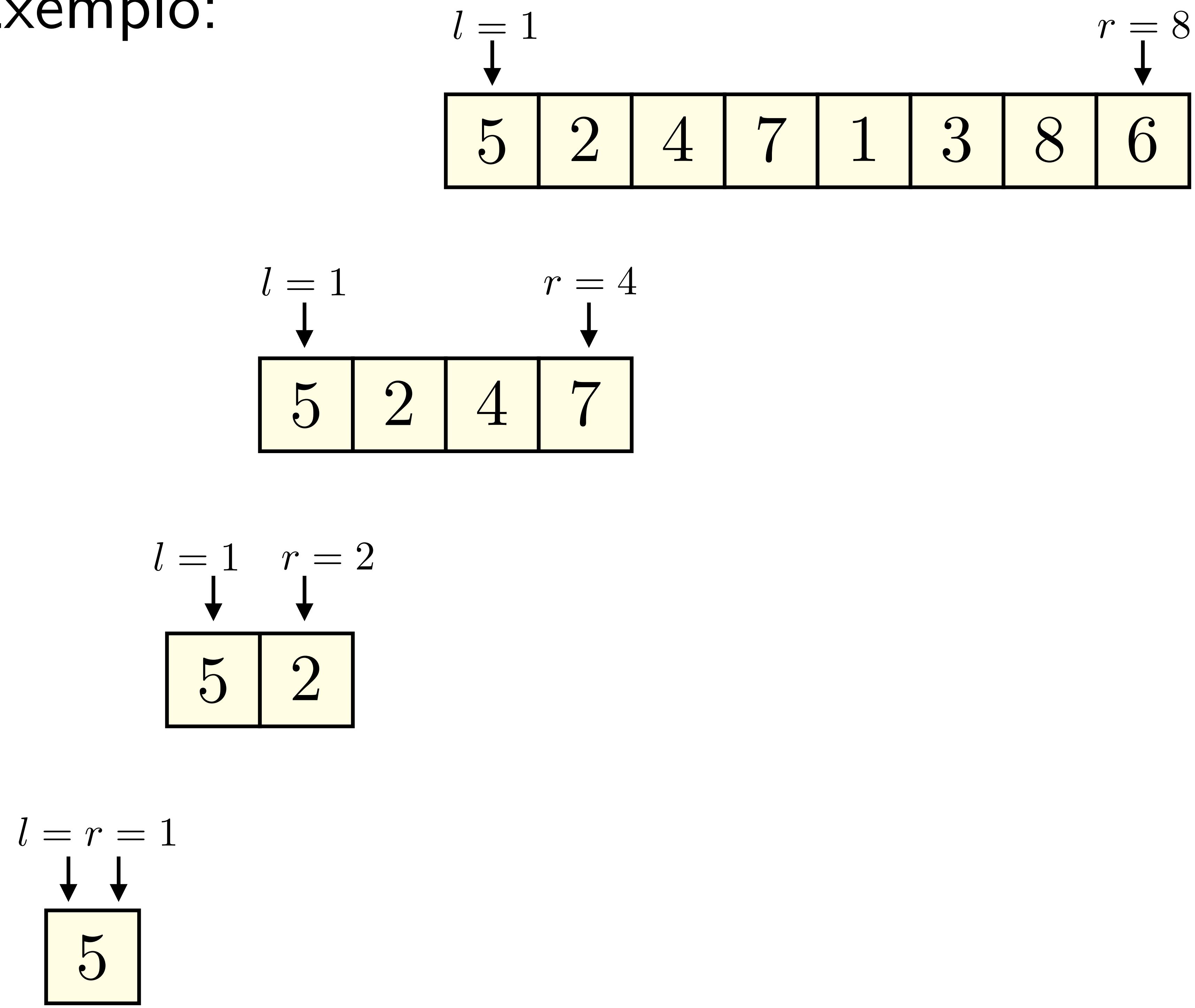
Mergesort

Exemplo:



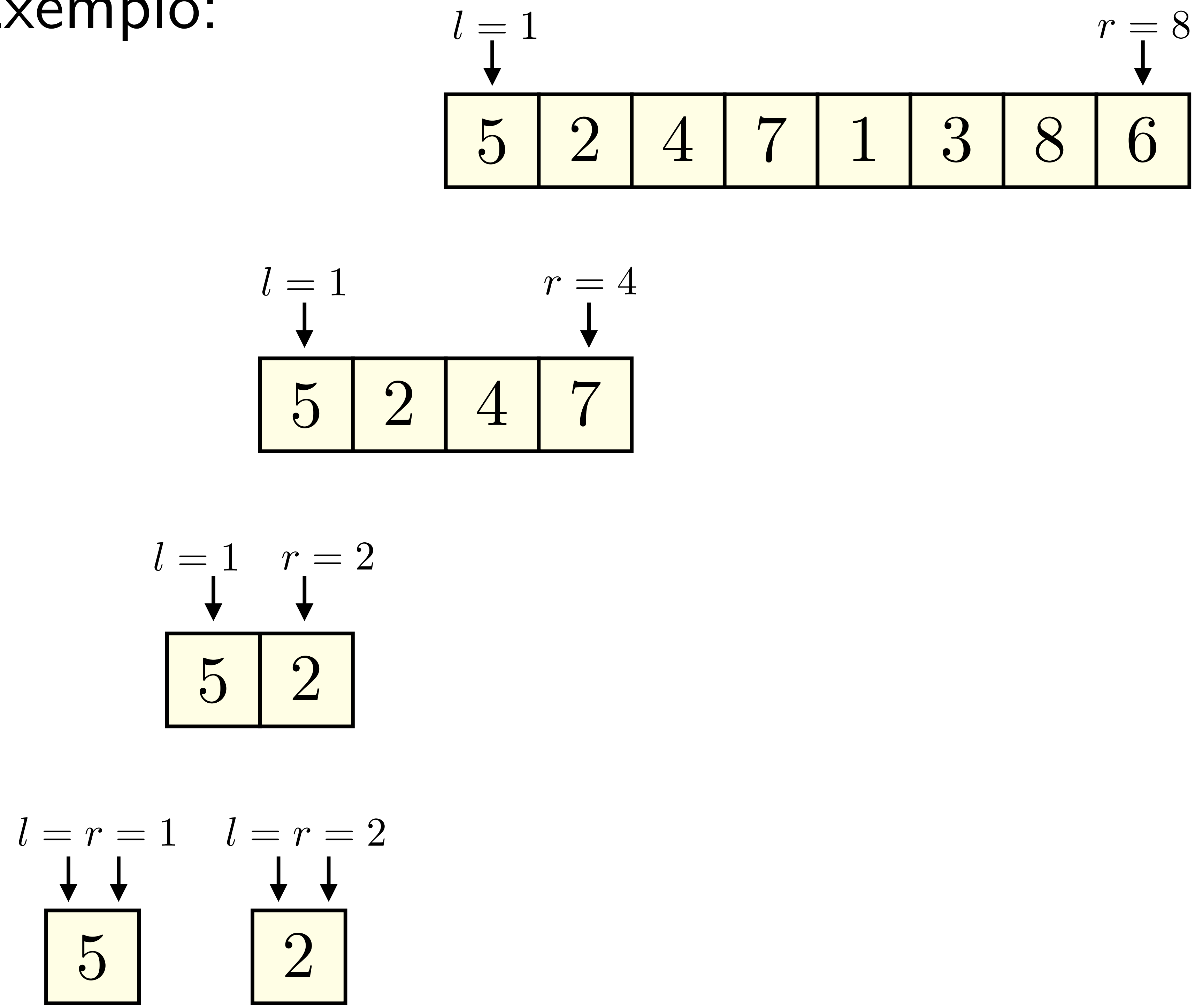
Mergesort

Exemplo:



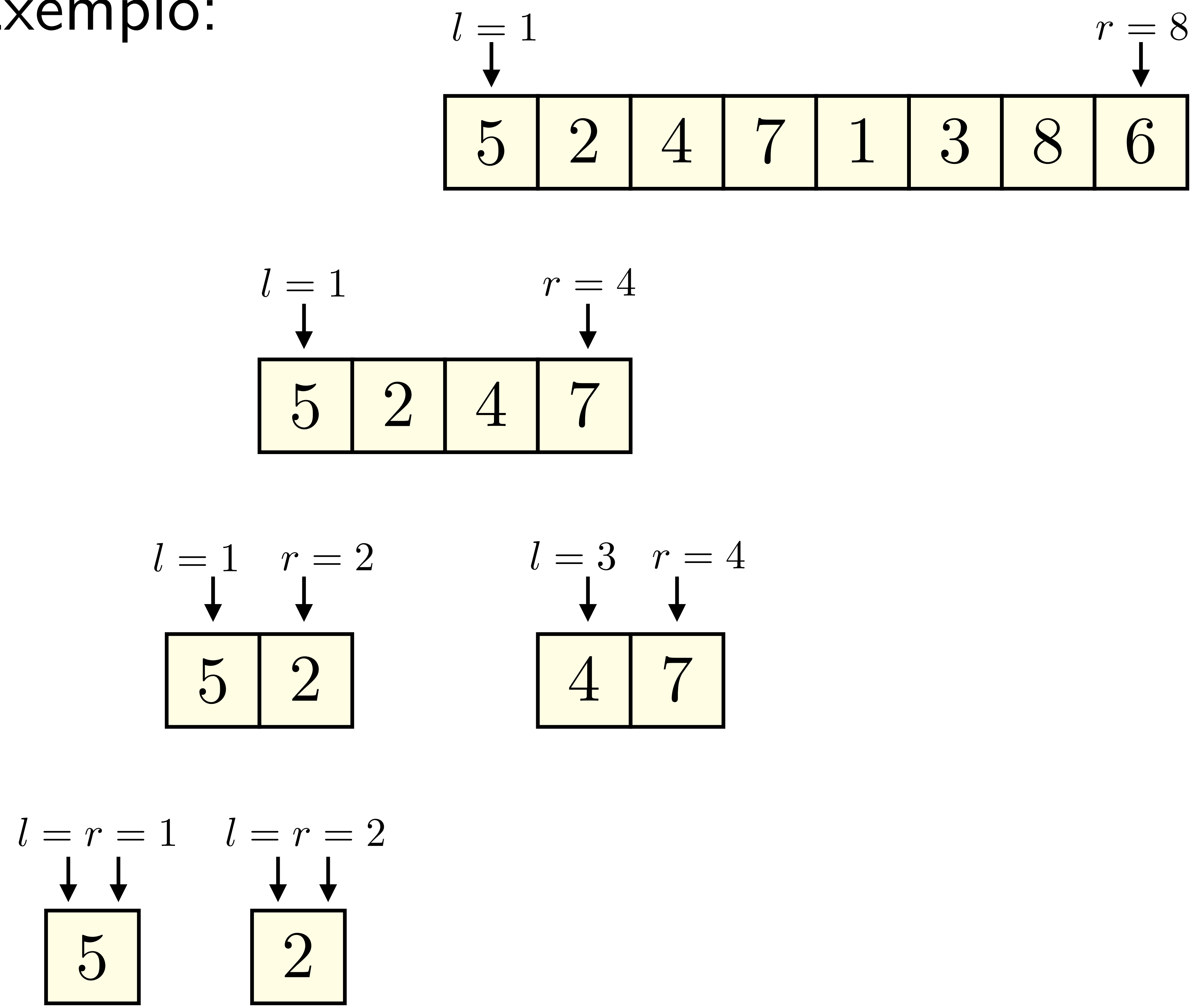
Mergesort

Exemplo:



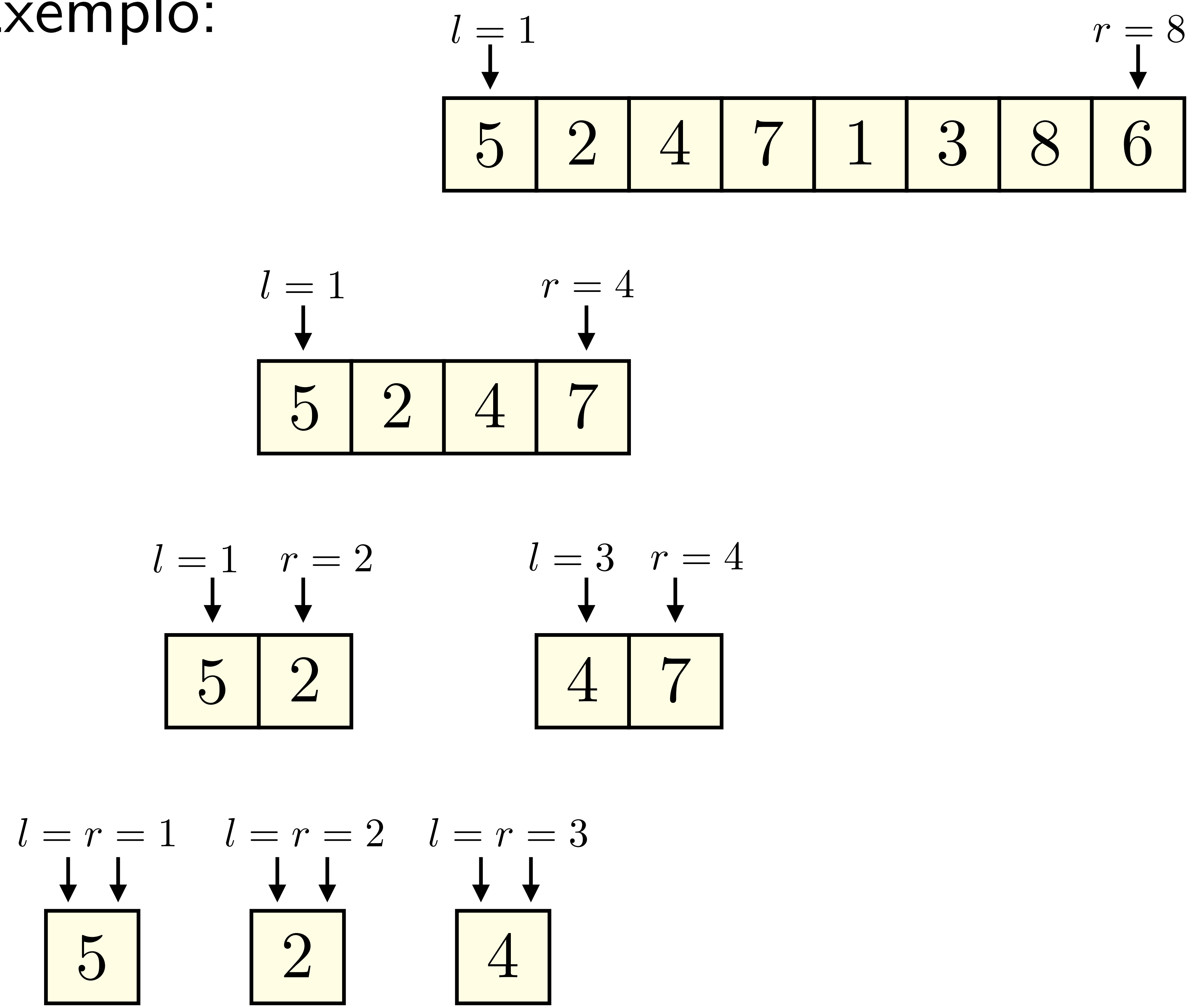
Mergesort

Exemplo:



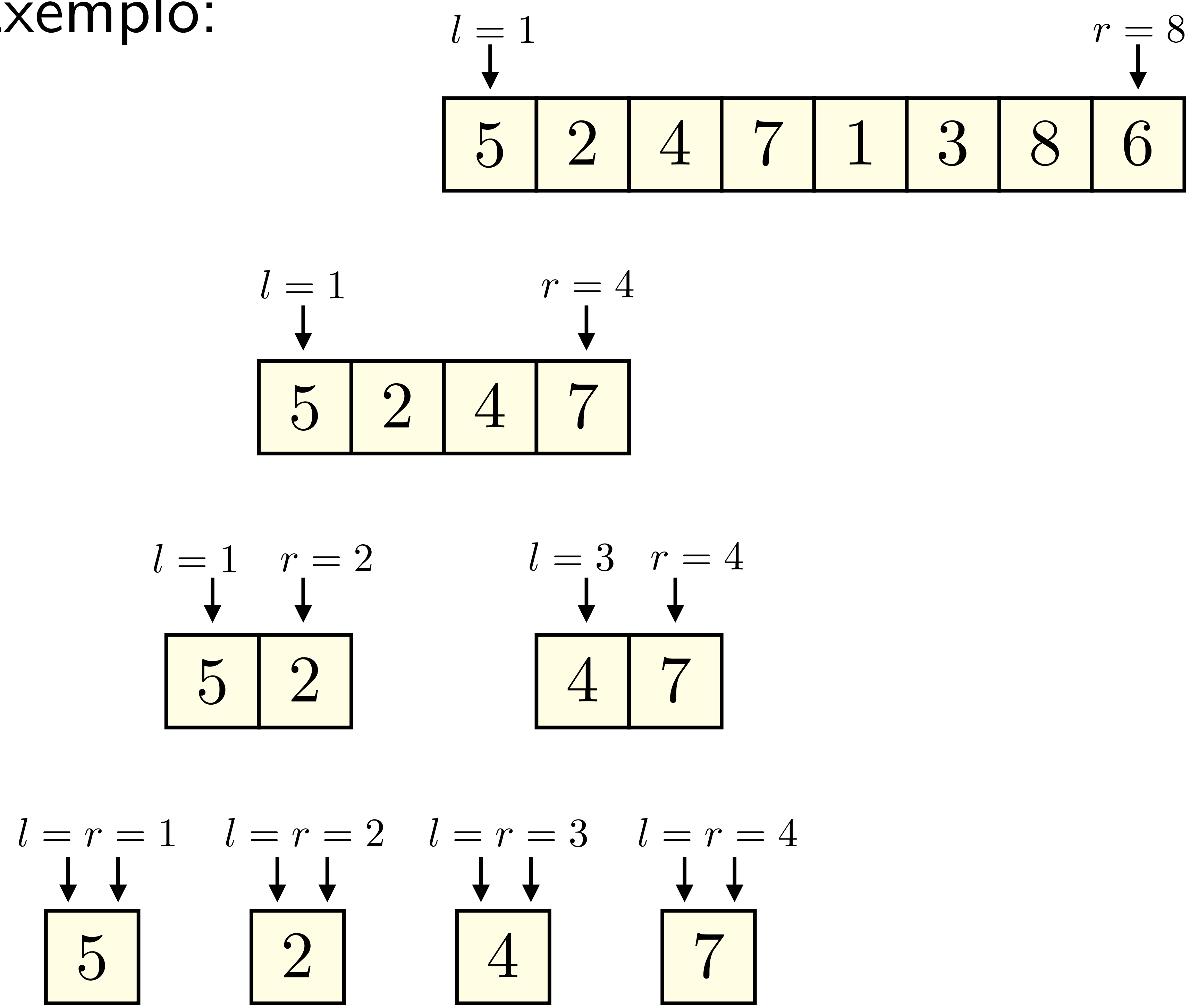
Mergesort

Exemplo:



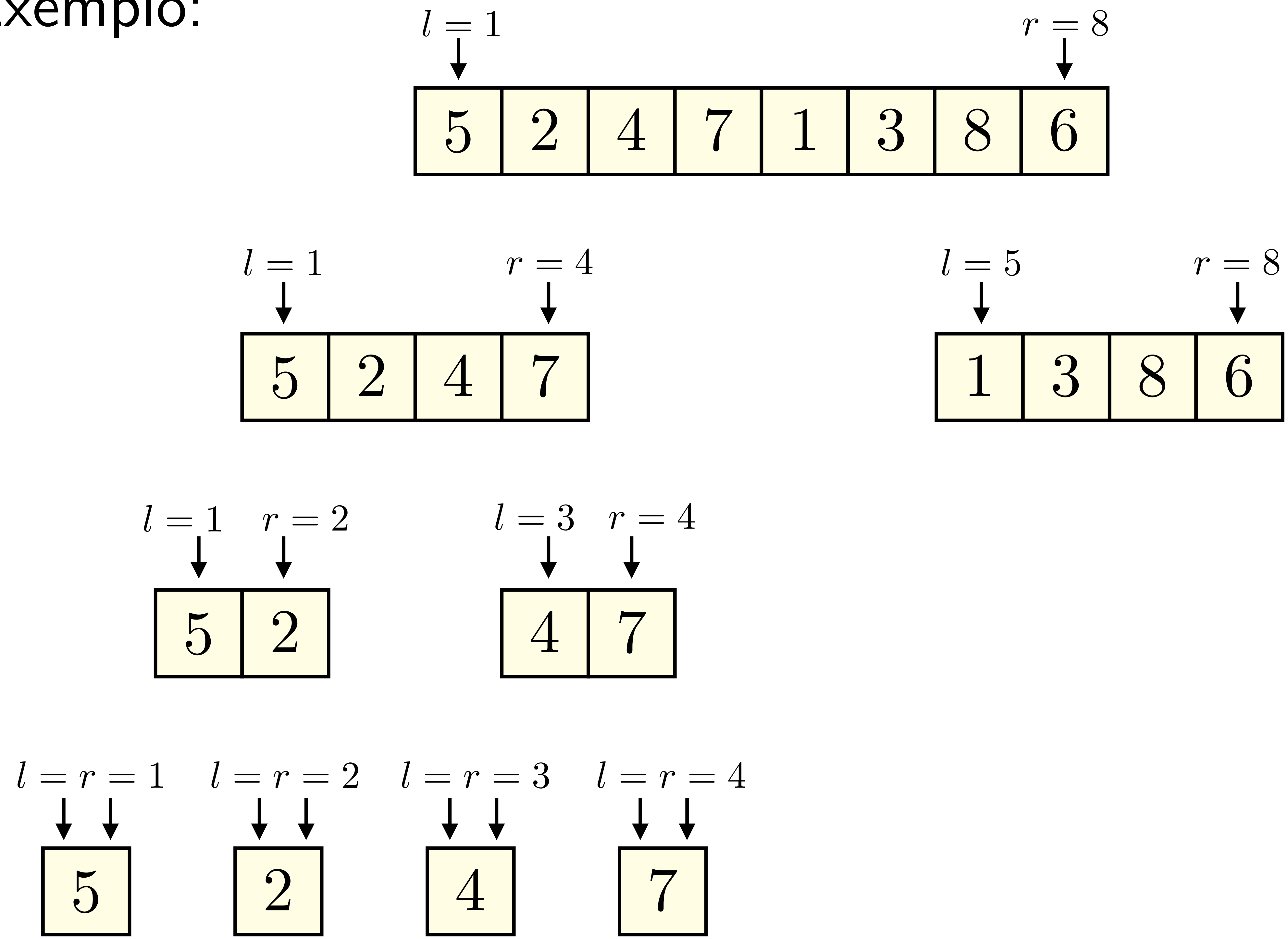
Mergesort

Exemplo:



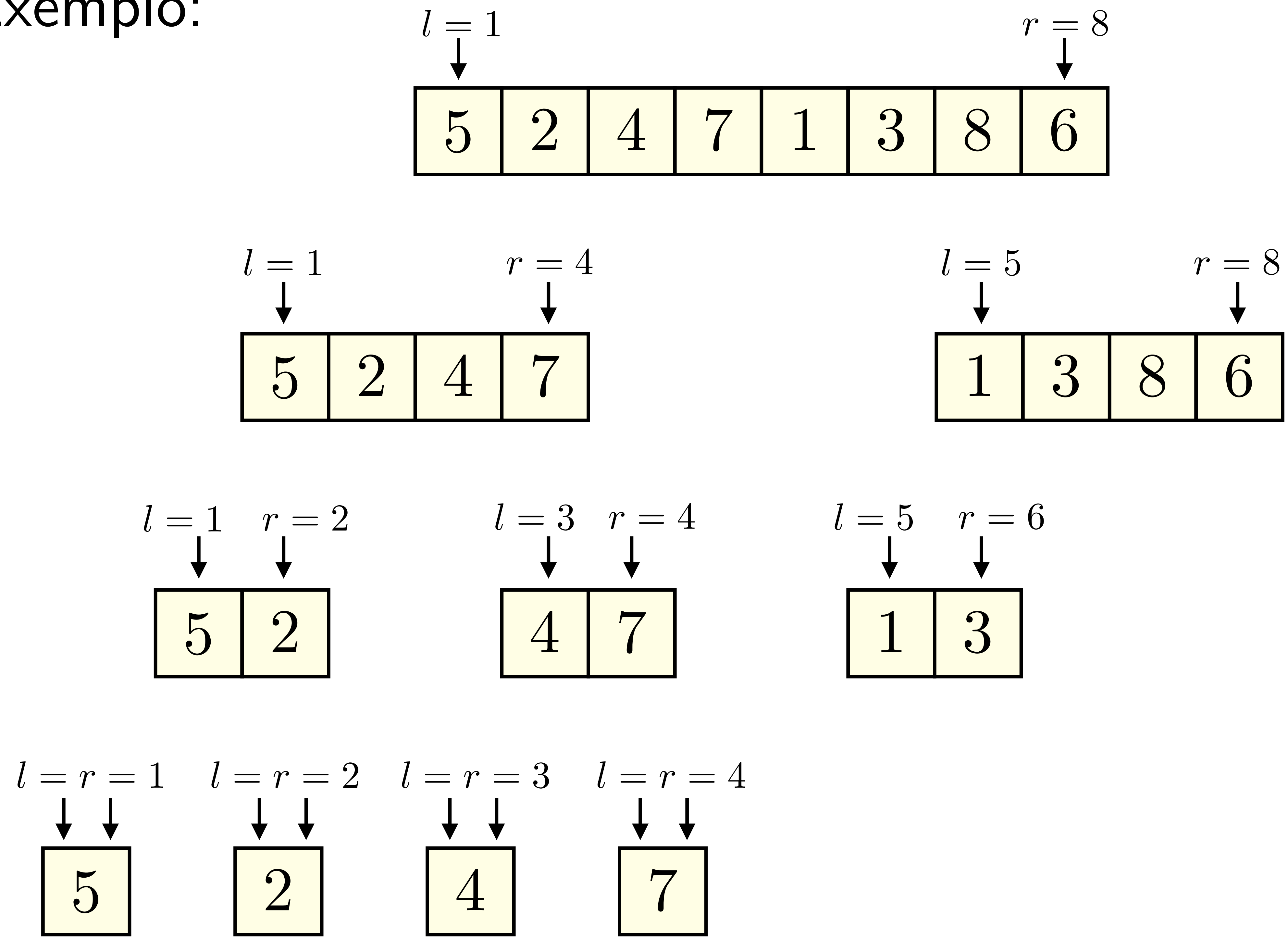
Mergesort

Exemplo:



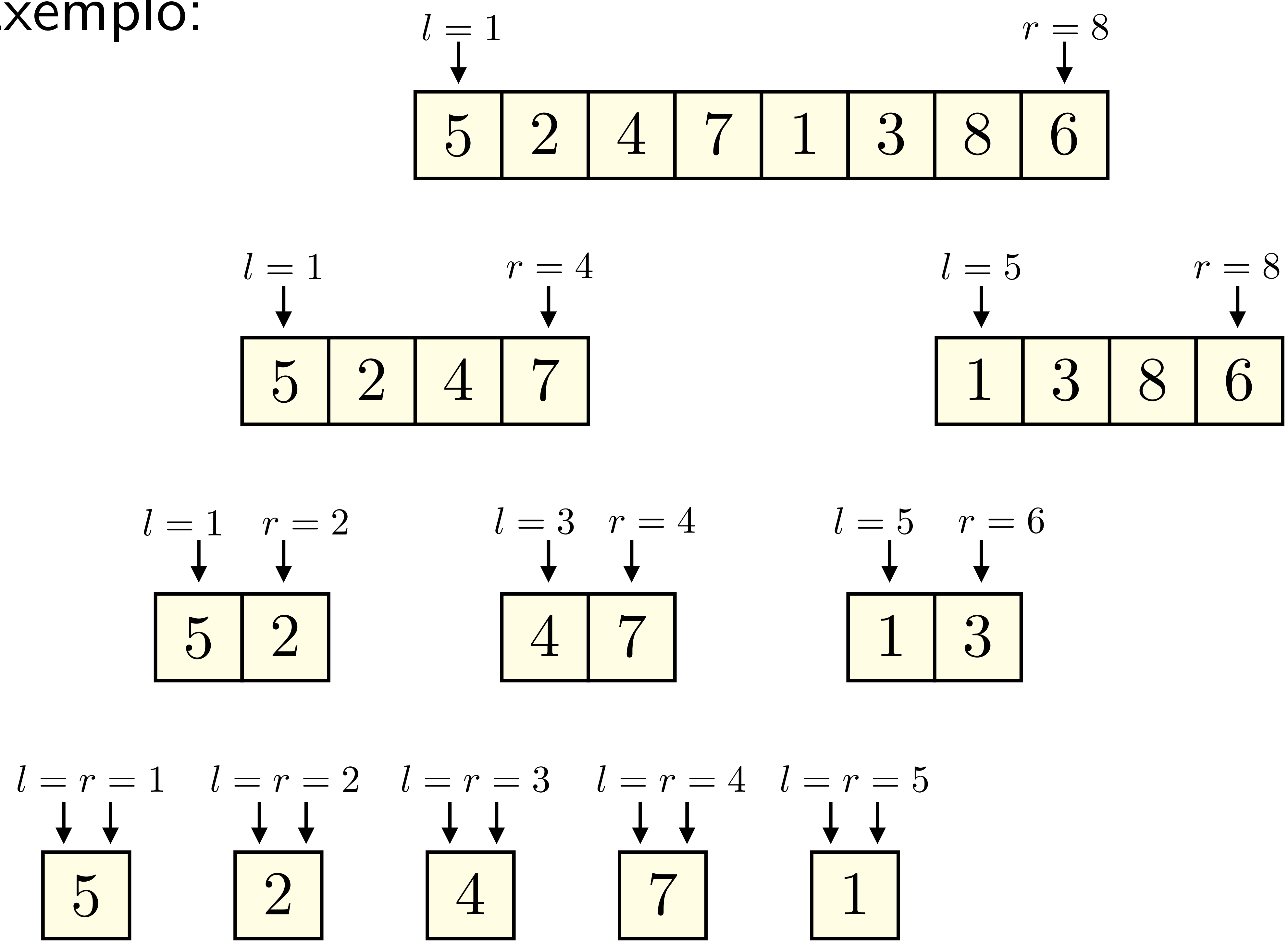
Mergesort

Exemplo:



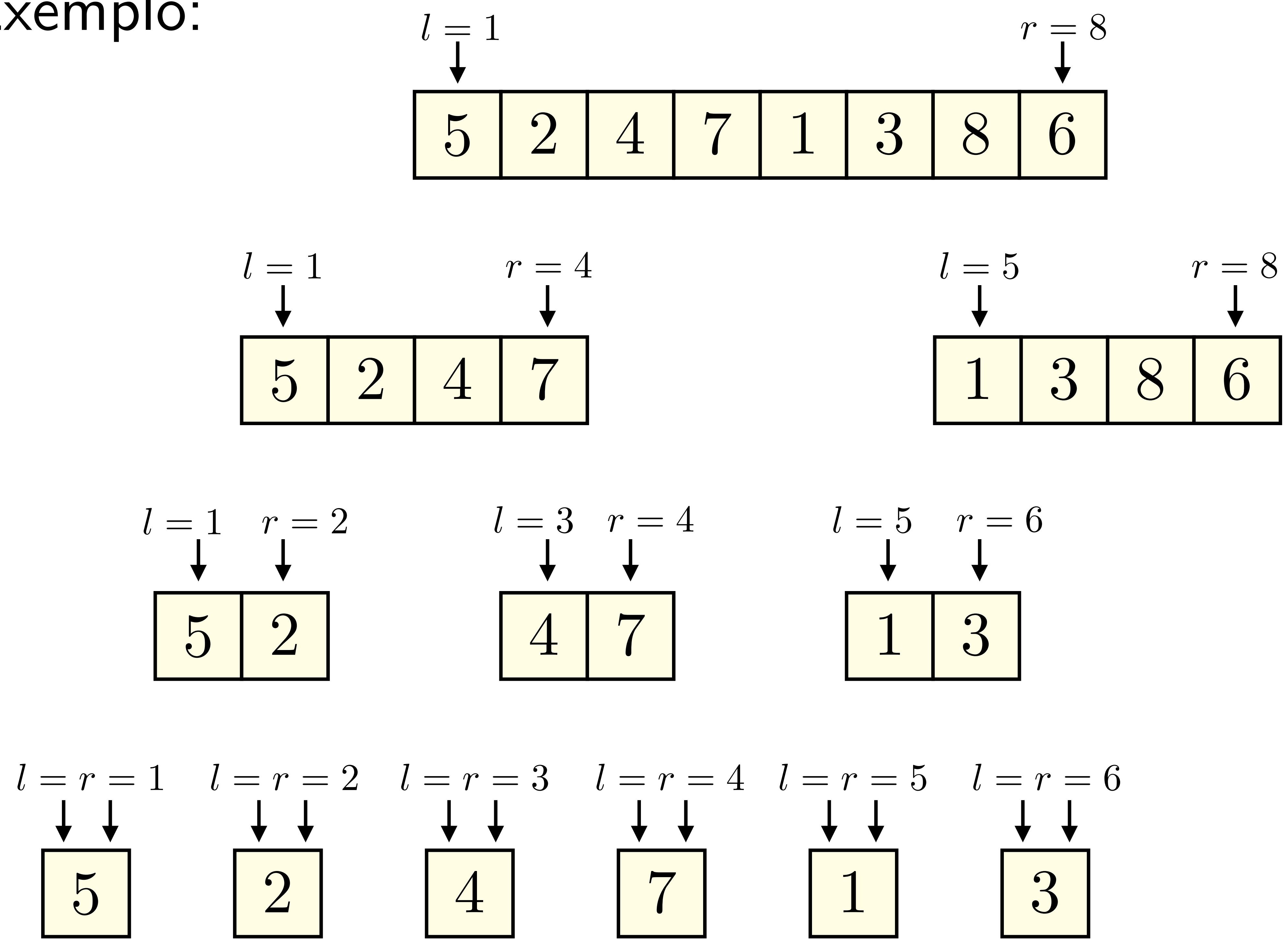
Mergesort

Exemplo:



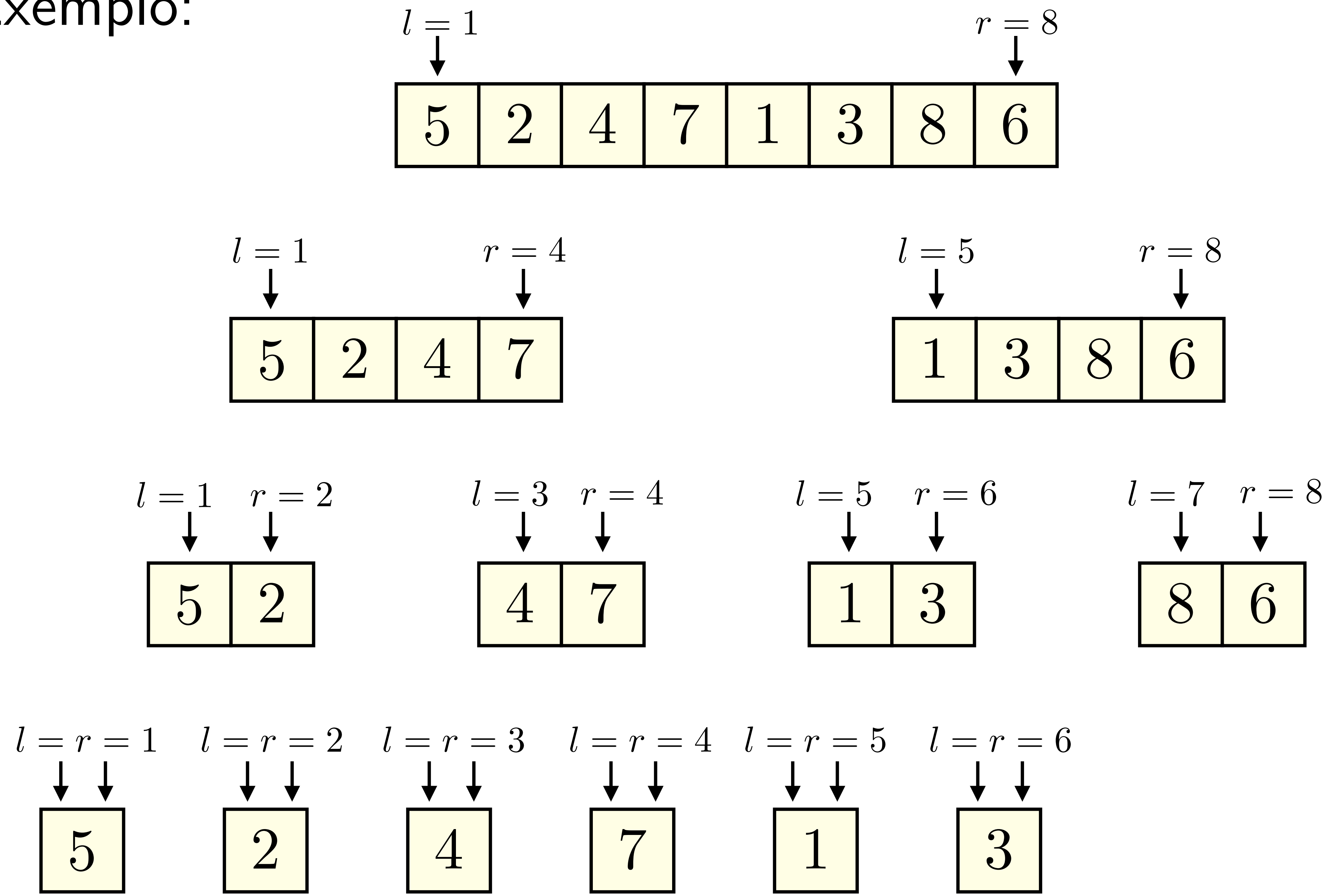
Mergesort

Exemplo:

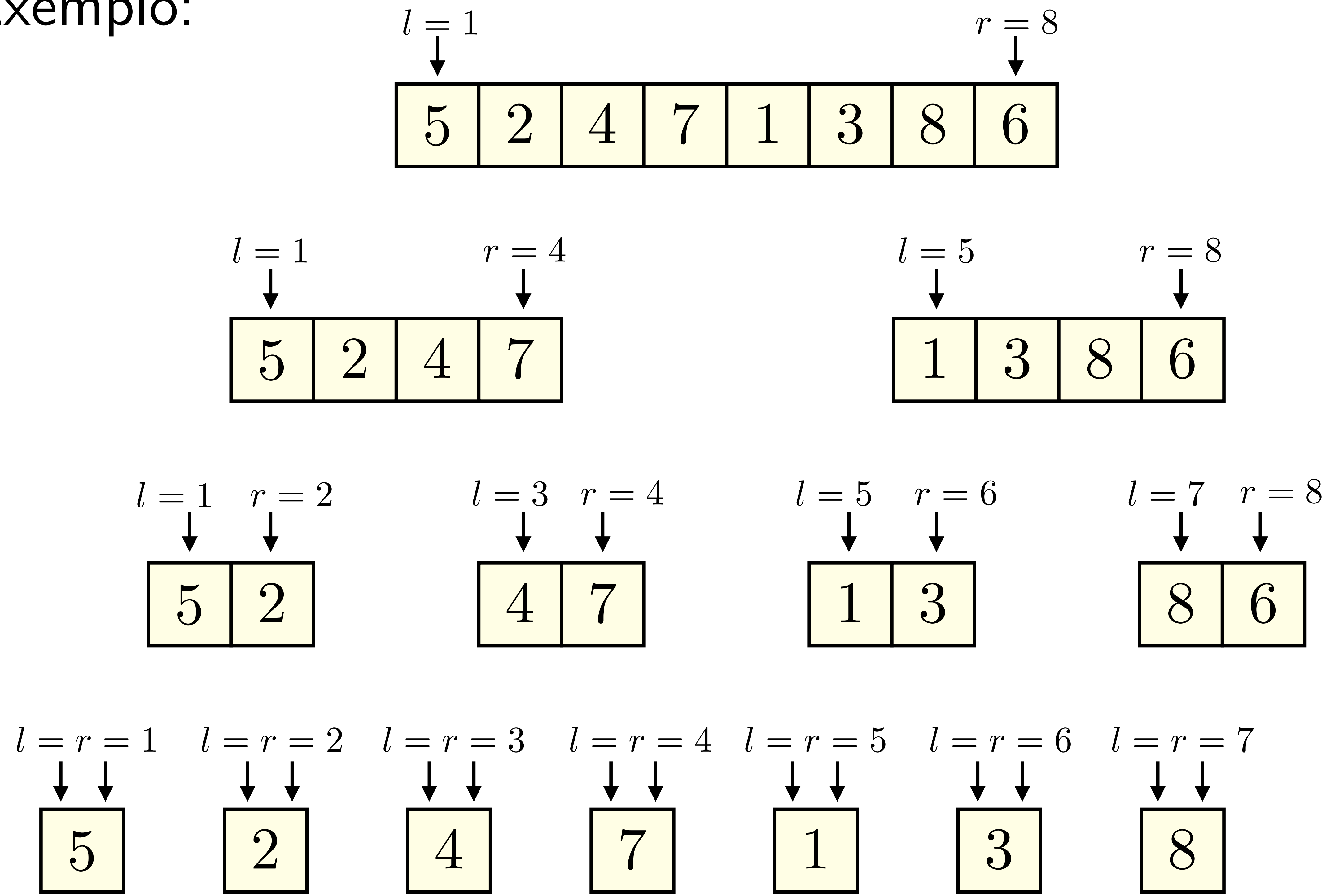


Mergesort

Exemplo:

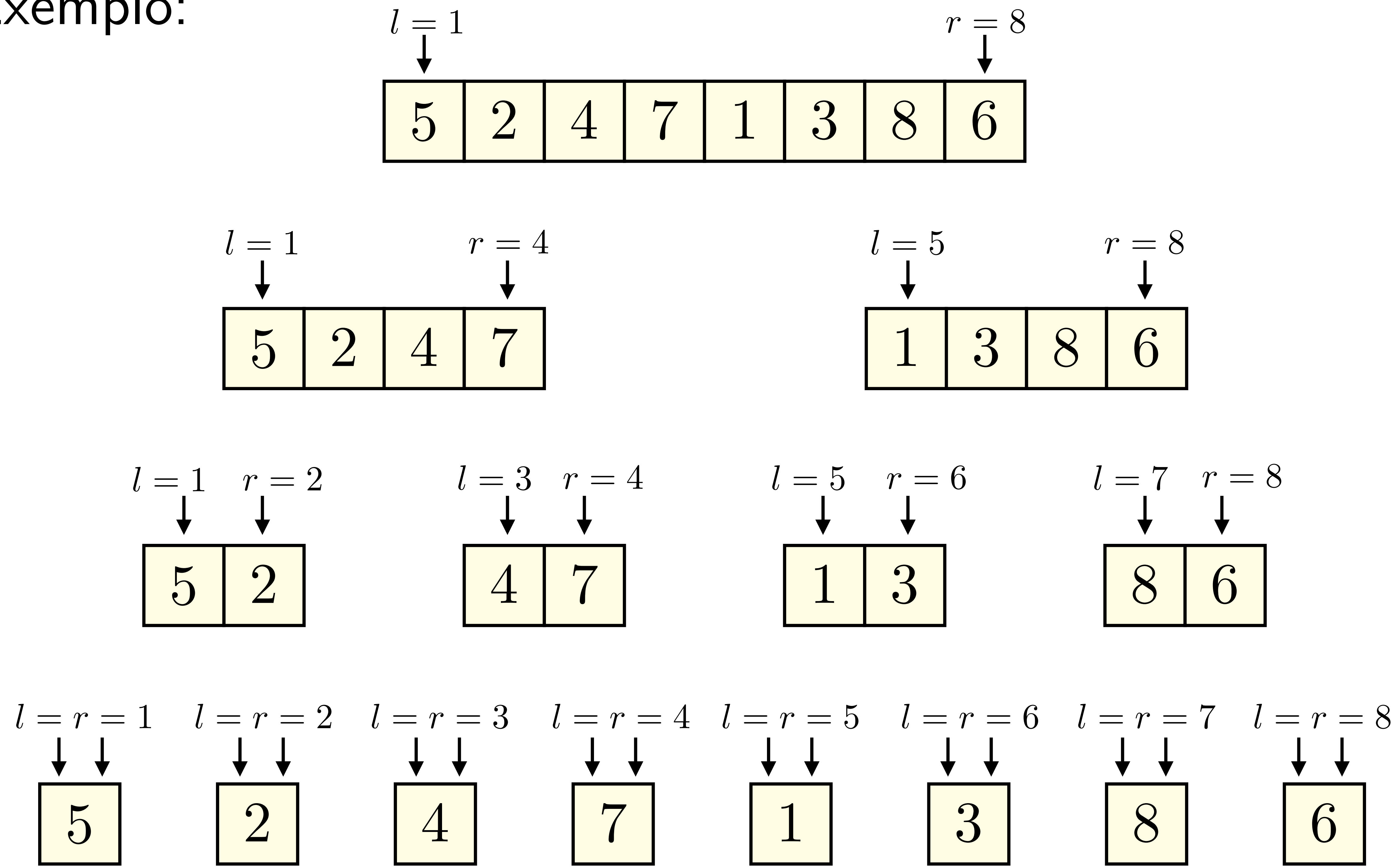


Exemplo:


$l = 1$
$r = 8$
5 2 4 7 1 3 8 6
$l = 1$
$r = 4$
5 2 4 7
$l = 5$
$r = 8$
1 3 8 6
$l = 1$
$r = 2$
5 2
$l = 3$
$r = 4$
4 7
$l = 5$
$r = 6$
1 3
$l = 7$
$r = 8$
8 6
$l = r = 1$
5
$l = r = 2$
2
$l = r = 3$
4
$l = r = 4$
7
$l = r = 5$
1
$l = r = 6$
3
$l = r = 7$
8

Mergesort

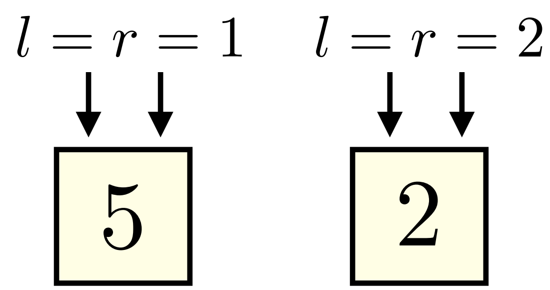
Exemplo:



Mergesort

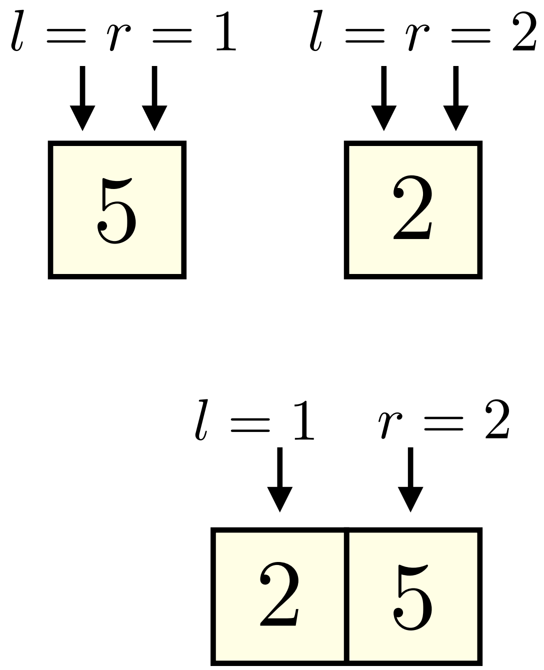
Exemplo:

Exemplo:



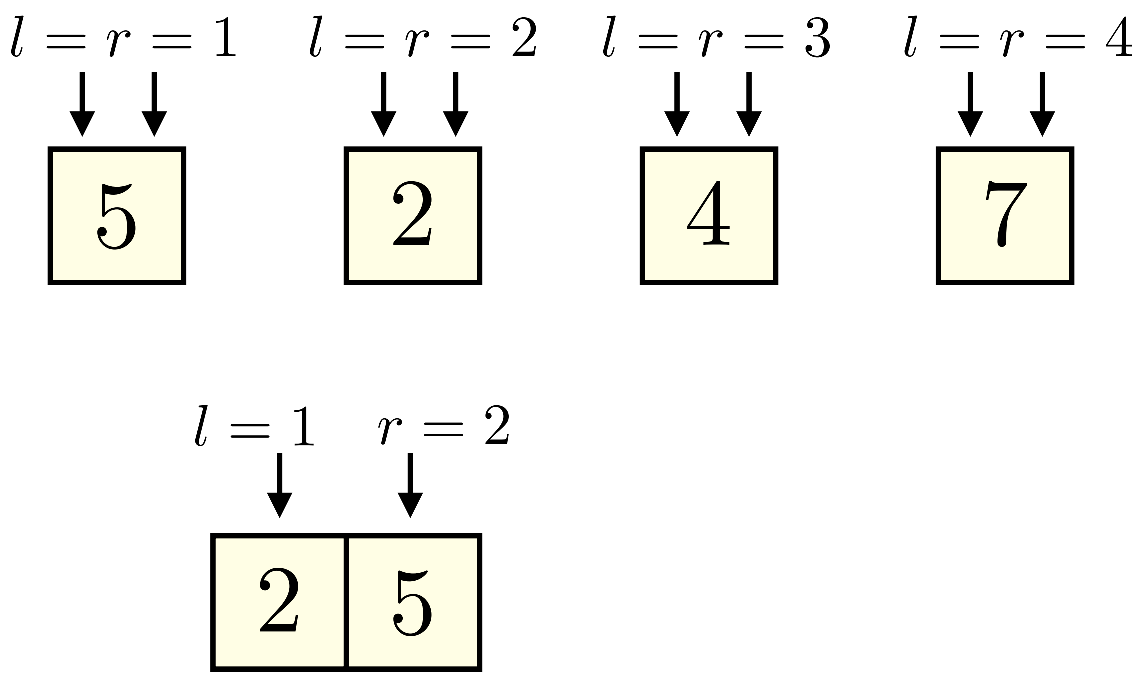
Mergesort

Exemplo:



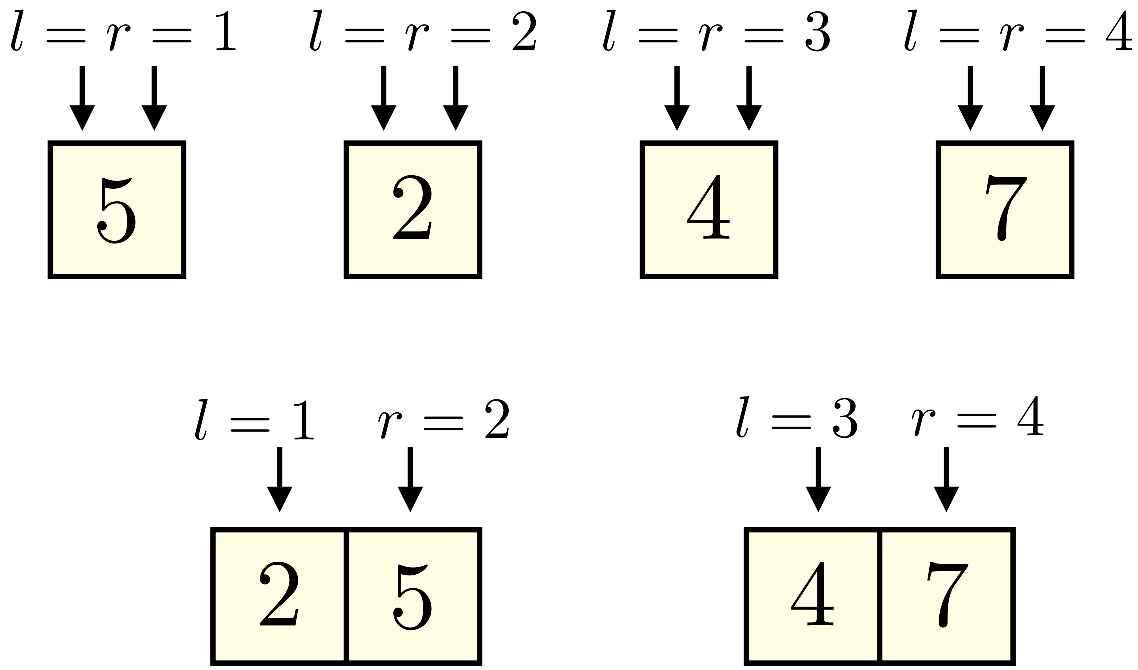
Mergesort

Exemplo:

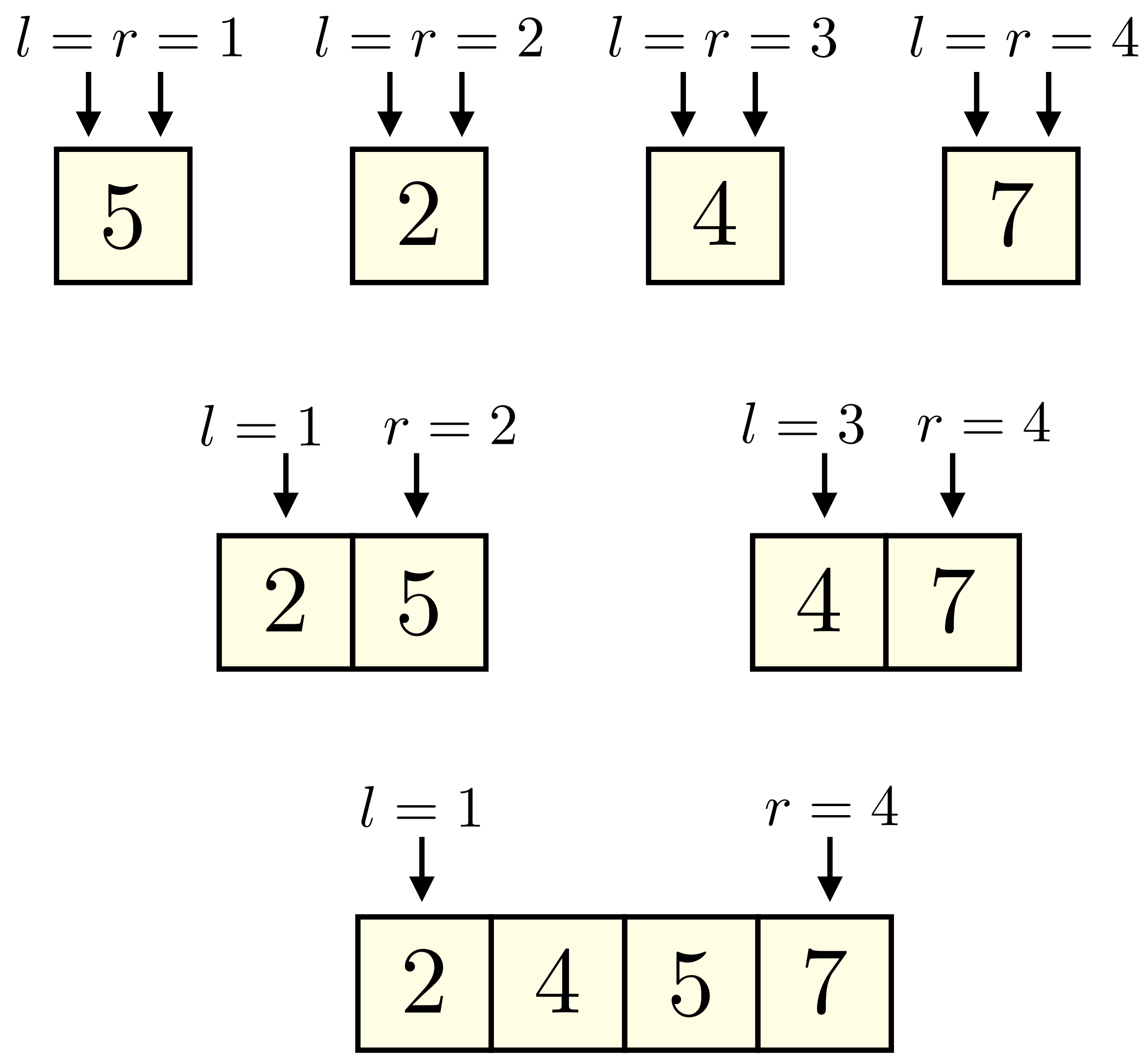


Mergesort

Exemplo:

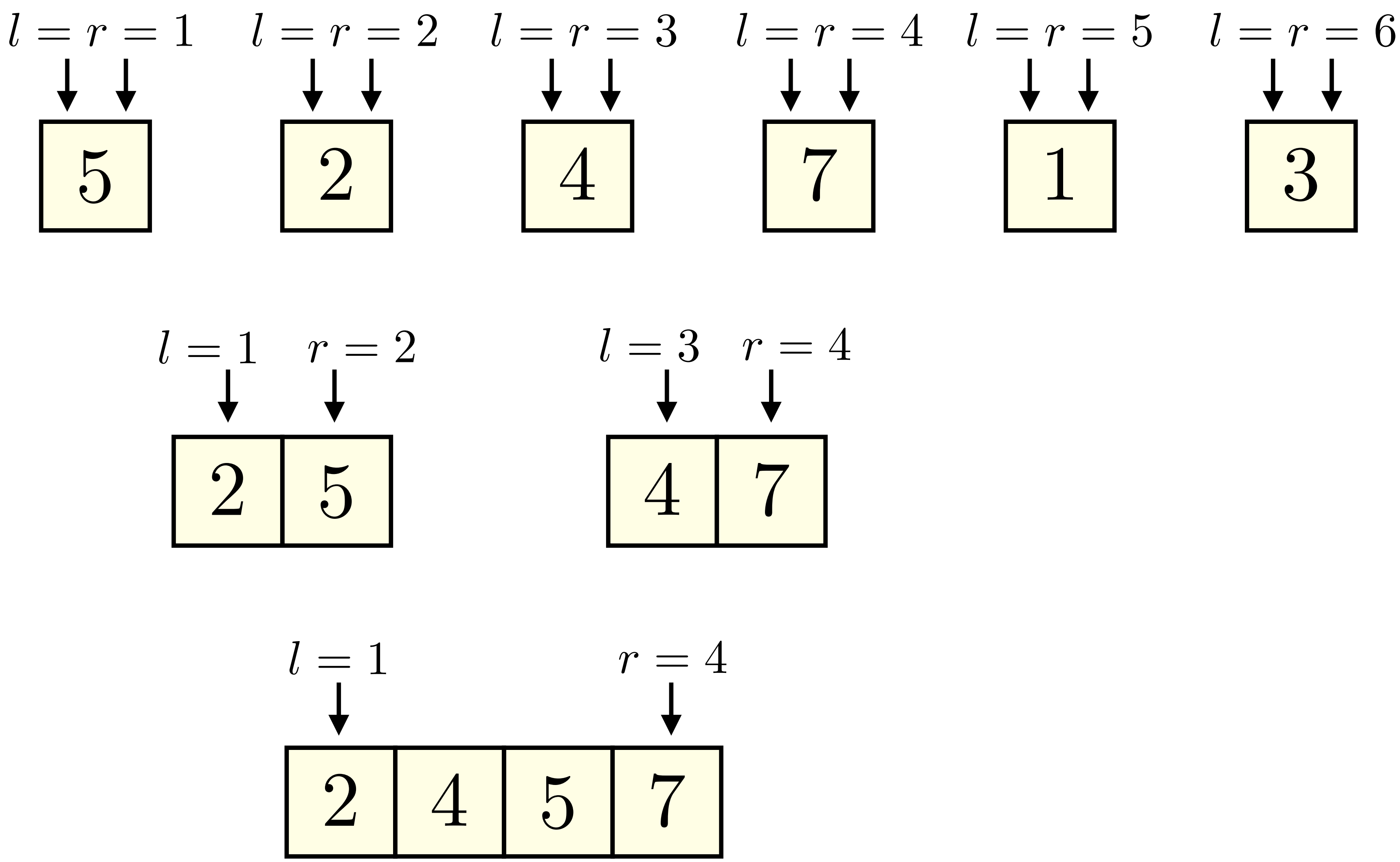


Exemplo:



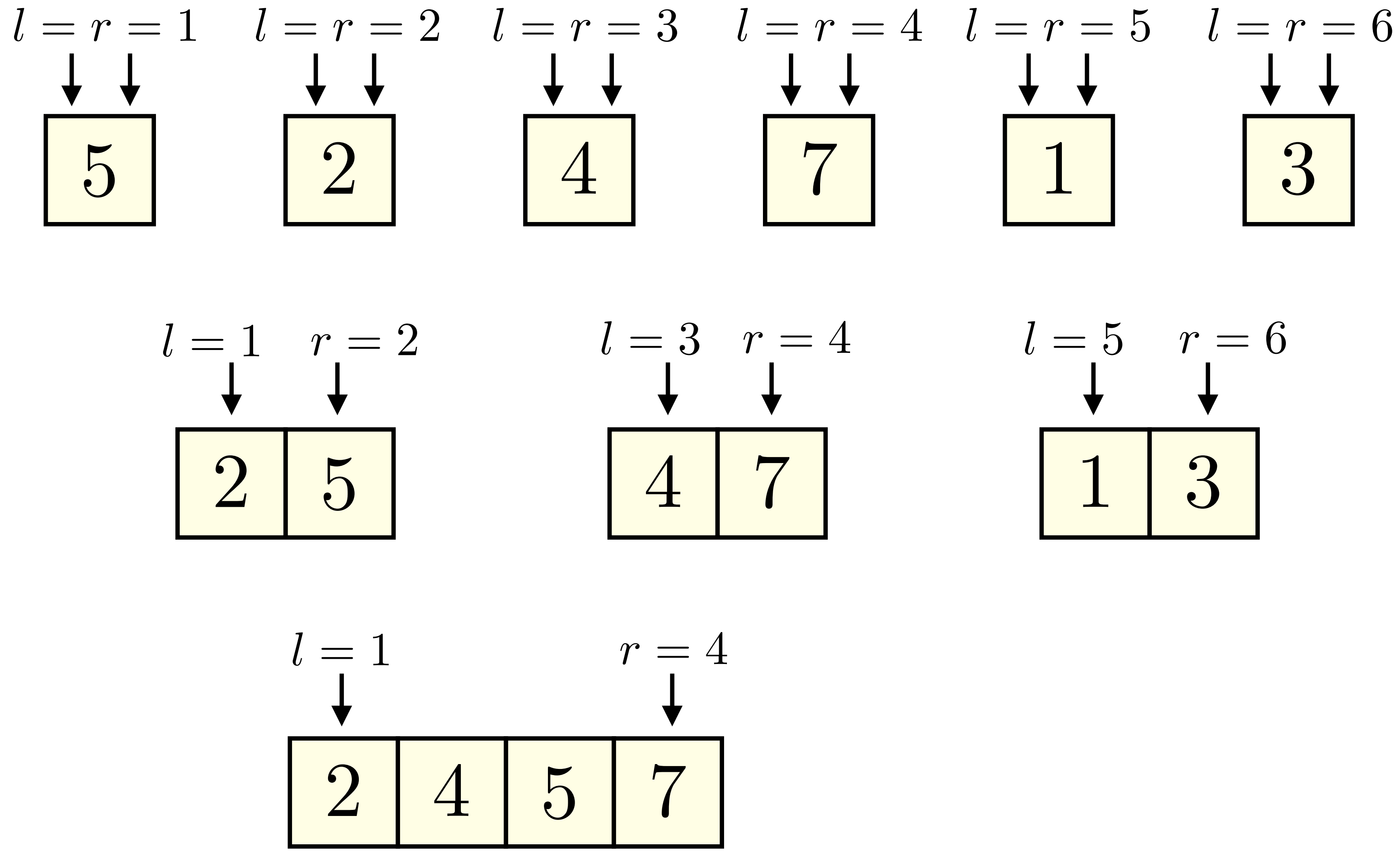
Mergesort

Exemplo:



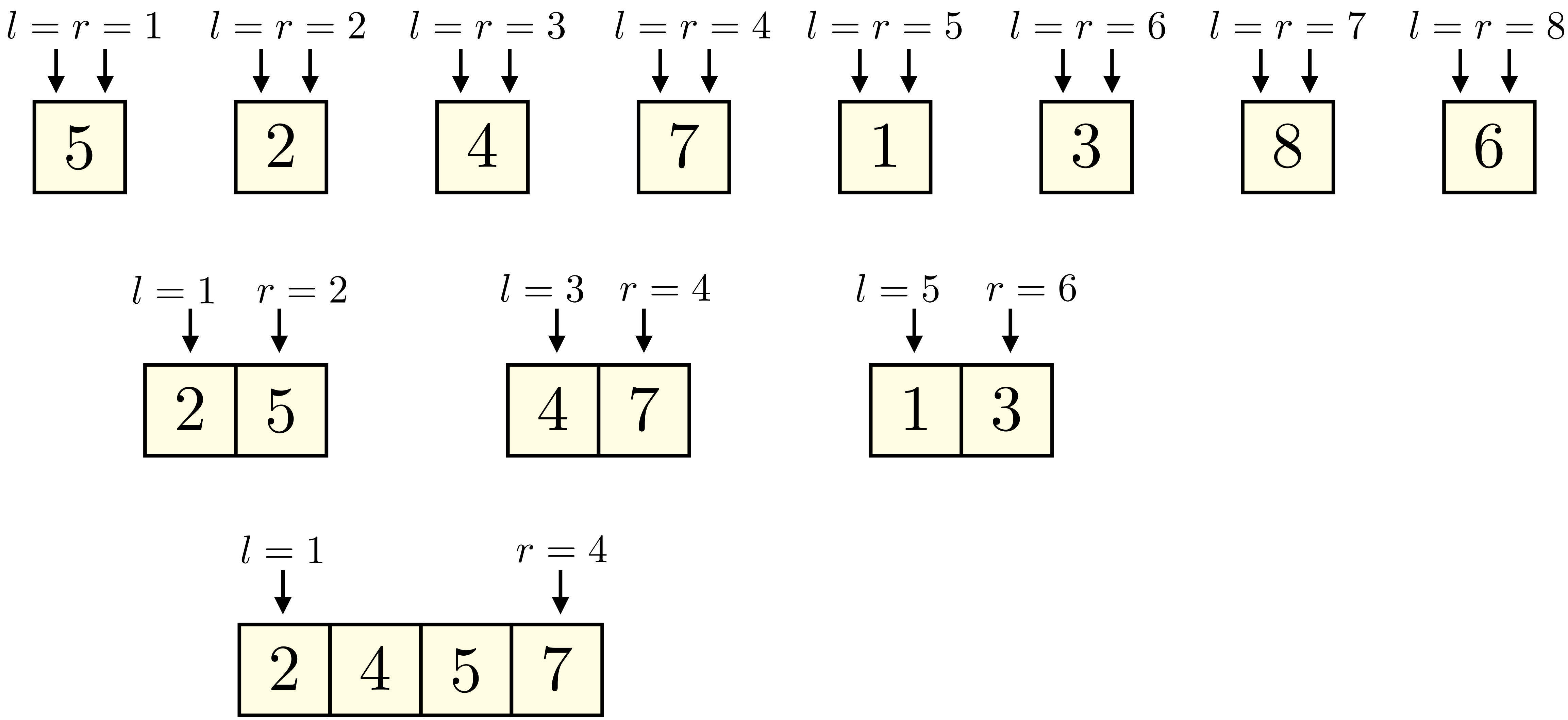
Mergesort

Exemplo:



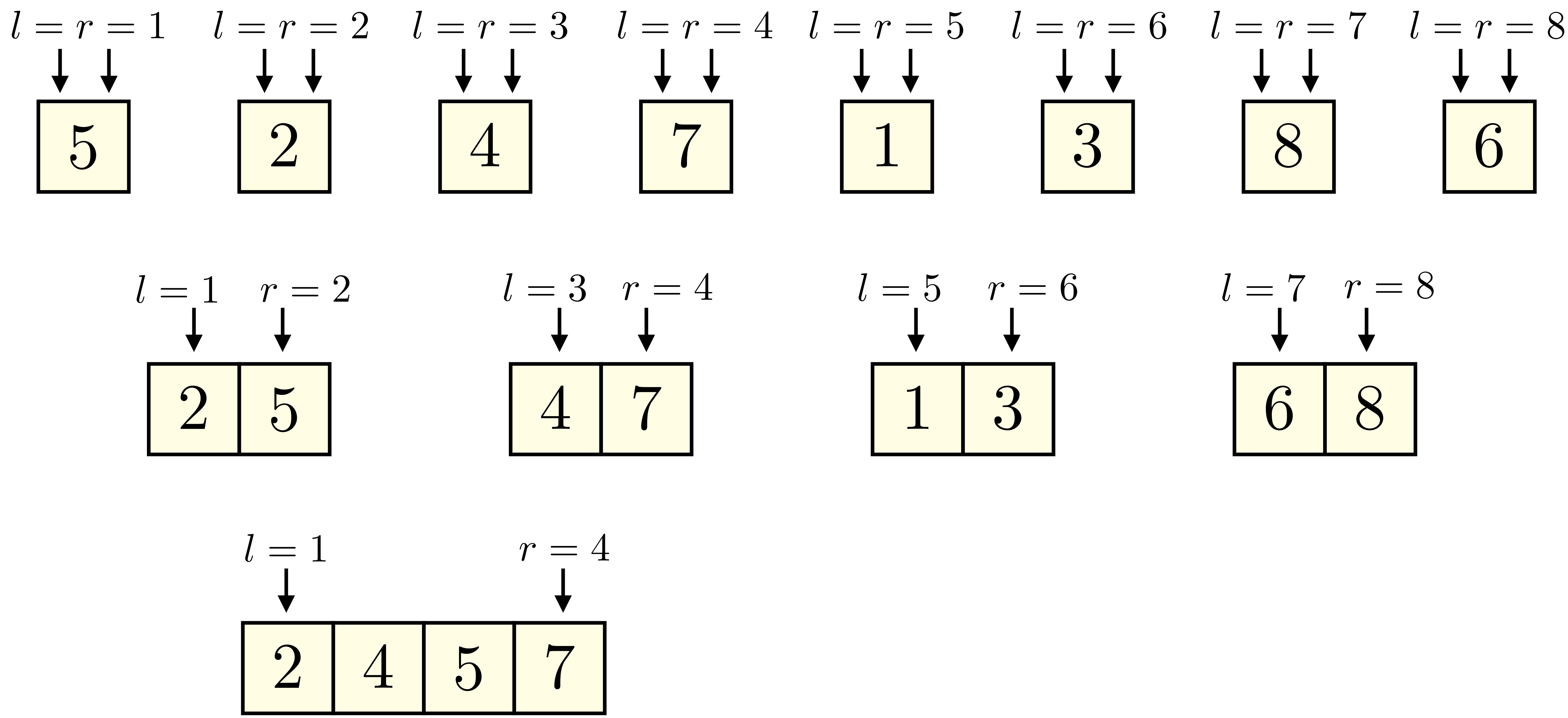
Mergesort

Exemplo:



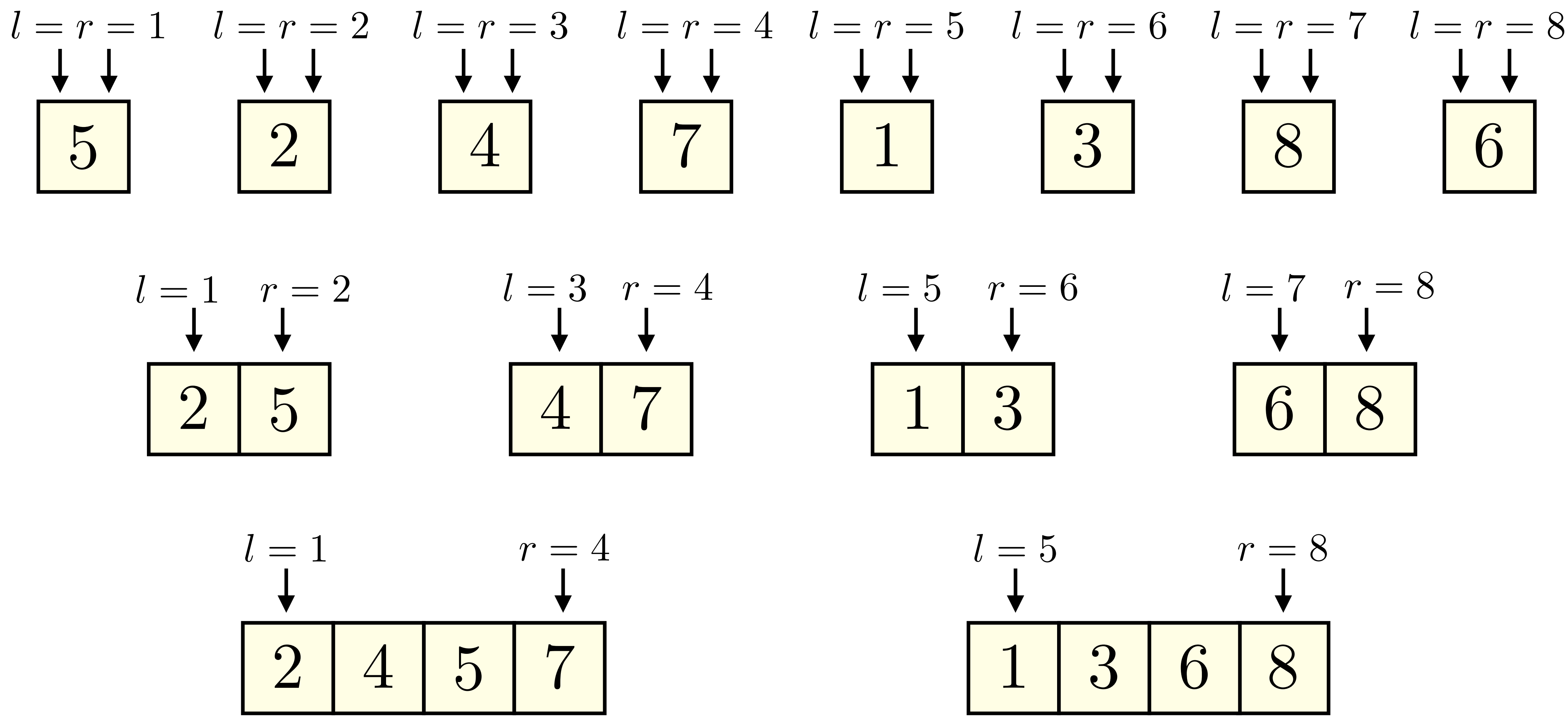
Mergesort

Exemplo:



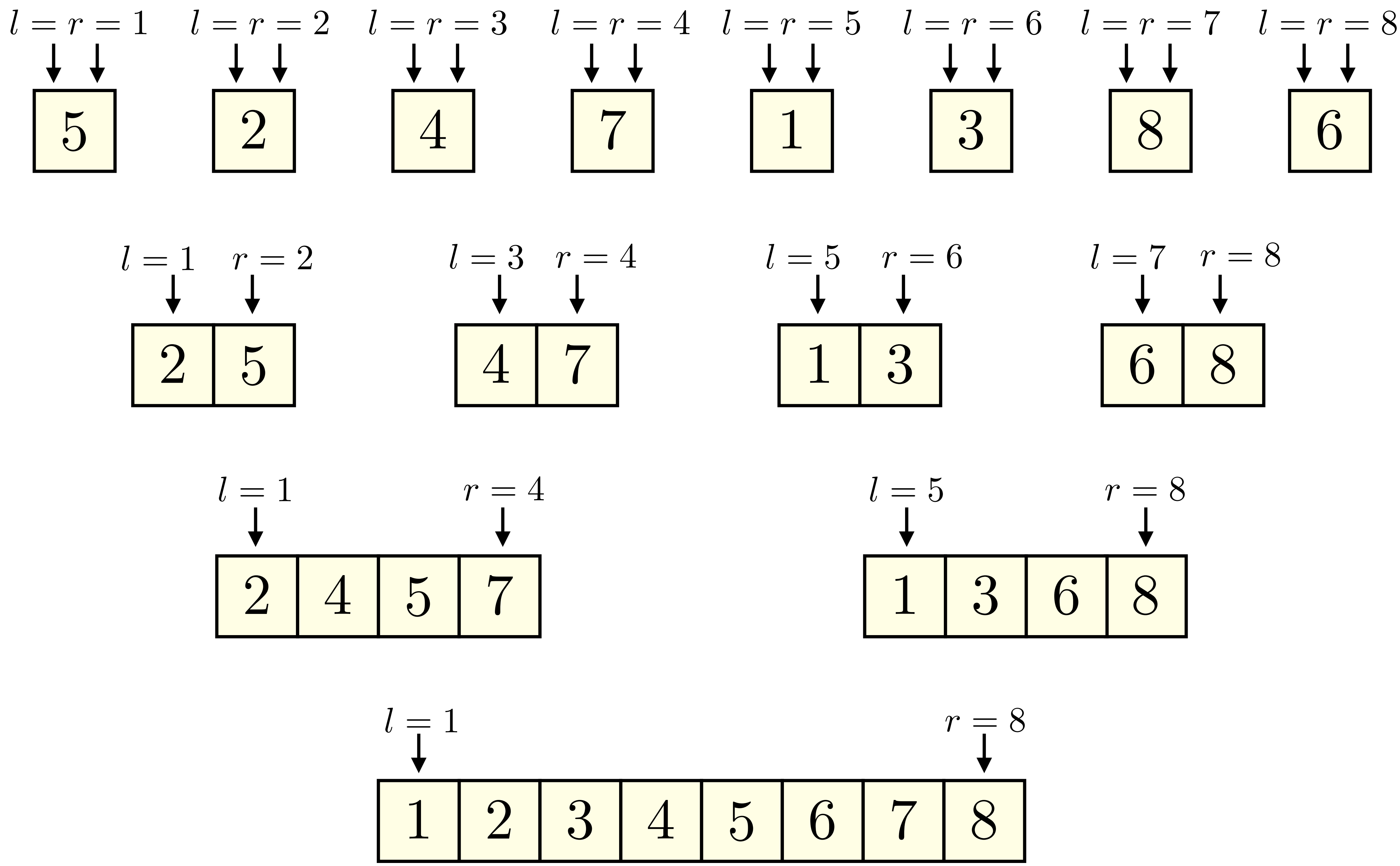
Mergesort

Exemplo:



Mergesort

Exemplo:



Mergesort

Mergesort

Assim como *partição* é a operação chave do algoritmo *quick-sort*, a *intercalação* de duas subseqüências é a operação chave de *mergesort*.

Mergesort

Assim como *partição* é a operação chave do algoritmo *quick-sort*, a *intercalação* de duas subseqüências é a operação chave de *mergesort*.

A *intercalação* de duas subseqüências é realizada por *intercala*$()$.

Mergesort

Assim como *partição* é a operação chave do algoritmo *quick-sort*, a *intercalação* de duas subseqüências é a operação chave de *mergesort*.

A *intercalação* de duas subseqüências é realizada por *intercala*$()$.

O procedimento *intercala*$()$ recebe um vetor, A, e três índices, l, m e r, tais que $l \leq m$ e $m < r$. Assumimos que as subseqüências $A[l], \ldots, A[m]$ e $A[m+1], \ldots, A[r]$ estão ordenadas.

Mergesort

Mergesort

algoritmo *intercala*(A, l, m, r)
    $n_1 \leftarrow m - l + 1$
    $n_2 \leftarrow r - m$
    defina vetores $L[1, \ldots, n_1 + 1]$ e $R[1, \ldots, n_2 + 1]$
    para $i = 1$ **até** n_1 **faça**
        $L[i] \leftarrow A[l + i - 1]$
    fimpara
    para $j = 1$ **até** n_2 **faça**
        $R[j] \leftarrow A[m + j]$
    fimpara
    $L[n_1 + 1] \leftarrow X$, onde $a \preceq X$, para todo $a \in A$
    $R[n_2 + 1] \leftarrow X$, onde $a \preceq X$, para todo $a \in A$
    $i \leftarrow 1$
    $j \leftarrow 1$
    para $k = l$ **até** r **faça**
        se *compara*$(L[j], R[j]) \neq 1$ **então**
            $A[k] \leftarrow L[i]$
            $i \leftarrow i + 1$
        senão
            $A[k] \leftarrow R[j]$
            $j \leftarrow j + 1$
        fimse
    fimpara
fimalgoritmo

Mergesort

Mergesort

A complexidade de tempo de *intercala* é $\Theta(r - l + 1)$.

Mergesort

A complexidade de tempo de *intercala* é $\Theta(r - l + 1)$.

Qual é a complexidade de tempo de *mergesort*?

Mergesort

A complexidade de tempo de *intercala* é $\Theta(r - l + 1)$.

Qual é a complexidade de tempo de *mergesort*?

Se A possui n elementos, então a complexidade de *mergesort* é aquela de *mergesort_aux* quando chamado com A, $l = 1$ e $r = n$.

Mergesort

A complexidade de tempo de *intercala* é $\Theta(r - l + 1)$.

Qual é a complexidade de tempo de *mergesort*?

Se A possui n elementos, então a complexidade de *mergesort* é aquela de *mergesort_aux* quando chamado com A, $l = 1$ e $r = n$.

Mas, *mergesort_aux* faz duas chamadas a si mesmo. Na primeira, o tamanho do vetor é $\lfloor n/2 \rfloor$ e, na outra, $\lceil n/2 \rceil$. Em seguida, *intercala*$()$ é chamado com A, $l = 1$, $m = \lfloor n/2 \rfloor$ e $r = n$.

Mergesort

Mergesort

Logo, se $t(n)$ é o número de operações primitivas de *mergesort_aux* quando $r - l + 1 = n$, podemos expressar $t(n)$ como segue:

$$t(n) = \begin{cases} \Theta(1) & \text{se } n = 1, \\ t\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + t\left(\left\lceil \frac{n}{2} \right\rceil\right) + \Theta(n) & \text{caso contrário.} \end{cases}$$

Logo, se $t(n)$ é o número de operações primitivas de *merge-sort_aux* quando $r-l+1=n$, podemos expressar $t(n)$ como segue:

$$t(n) = \begin{cases} \Theta(1) & \text{se } n = 1, \\ t\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + t\left(\left\lceil \frac{n}{2} \right\rceil\right) + \Theta(n) & \text{caso contrário.} \end{cases}$$

Na recorrência acima, $\Theta(1)$ representa o número de operações primitivas executadas quando *mergesort_aux* não chama a si próprio.

Mergesort

Logo, se $t(n)$ é o número de operações primitivas de *merge-sort_aux* quando $r - l + 1 = n$, podemos expressar $t(n)$ como segue:

$$t(n) = \begin{cases} \Theta(1) & \text{se } n = 1, \\ t\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + t\left(\left\lceil \frac{n}{2} \right\rceil\right) + \Theta(n) & \text{caso contrário.} \end{cases}$$

Na recorrência acima, $\Theta(1)$ representa o número de operações primitivas executadas quando *mergesort_aux* não chama a si próprio.

Já $\Theta(n)$ representa o número de operações primitivas de *intercala*$()$.

Mergesort

Mergesort

Podemos mostrar que a recorrência anterior tem uma solução, $t(n)$, tal que

$$t(n) = \Theta(n \lg n)\,.$$

Mergesort

Podemos mostrar que a recorrência anterior tem uma solução, $t(n)$, tal que

$$t(n) = \Theta(n \lg n)\,.$$

Isto implica que a complexidade de tempo de *mergesort* é $\Theta(n \lg n)$ para um vetor de entrada, A, com n elementos. Note que este é o caso para **toda** e **qualquer** entrada de tamanho n.

Mergesort

Podemos mostrar que a recorrência anterior tem uma solução, $t(n)$, tal que

$$t(n) = \Theta(n \lg n)\,.$$

Isto implica que a complexidade de tempo de *mergesort* é $\Theta(n \lg n)$ para um vetor de entrada, A, com n elementos. Note que este é o caso para **toda** e **qualquer** entrada de tamanho n.

Como veremos na próxima aula, o fato acima implica que *mergesort* é um algoritmo ótimo para o problema da ordenação, quando consideramos o modelo de comparações entre chaves.

Mergesort

Mergesort

Se o *mergesort* é ótimo, por que o *quicksort* é preferido na prática?

Mergesort

Se o *mergesort* é ótimo, por que o *quicksort* é preferido na prática?

O fato do *mergesort* ser preterido se deve a dois fatores.

Se o *mergesort* é ótimo, por que o *quicksort* é preferido na prática?

O fato do *mergesort* ser preterido se deve a dois fatores.

O primeiro deles é que, na grande maioria das entradas, ambos os algoritmos são $\Theta(n \lg n)$. No entanto, a constante "escondida" na notação Θ da complexidade de tempo de *quicksort* é bem menor do que aquela do *mergesort* para a maioria das entradas.

Mergesort

Mergesort

O segundo fator diz respeito ao gasto de memória com os vetores auxiliares, L e R, no procedimento *intercala*. O tamanho desses vetores chega a ser proporcional a n (na primeira chamada).

Mergesort

O segundo fator diz respeito ao gasto de memória com os vetores auxiliares, L e R, no procedimento *intercala*. O tamanho desses vetores chega a ser proporcional a n (na primeira chamada).

O problema é que não temos como intercalar as duas subseqüências sem o uso de vetores auxiliares (tente imaginar o porquê).

O segundo fator diz respeito ao gasto de memória com os vetores auxiliares, L e R, no procedimento *intercala*. O tamanho desses vetores chega a ser proporcional a n (na primeira chamada).

O problema é que não temos como intercalar as duas subseqüências sem o uso de vetores auxiliares (tente imaginar o porquê).

Por outro lado, *quicksort* ordena os elementos em A usando apenas o próprio A. Dizemos que o *quicksort* realiza ordenação *in-loco*.

Referências

Referências

- Paulo A. Azeredo
  Métodos de Classificação de Dados, Editora Campus, 1996.

Existem inúmeros applets na Internet que ilustram o funcionamento dos algoritmos de ordenação mais conhecidos. Eu recomendo o seguinte:

http://www.cs.oswego.edu/~mohammad/classes/csc241/samples/sort/Sort2-E.html