Aula 09

Divisão e Conquista - MergeSort

Prof. Marco Aurélio Stefanes

`www.facom.ufms.br/~marco`

Divisão e Conquista

Método de Divisão e Conquista

- **Divisão:** Divida o problema em duas ou mais partes, criando subproblemas menores.
- **Conquista:** Os subproblemas são resolvidos recursivamente usando divisão e conquista. Caso os subproblemas sejam suficientemente pequenos resolva-os de forma direta.
- **Combina:** Tome cada uma das partes e junte-as todas de forma a resolver o problema original.

Mergesort

- Mergesort é um algoritmo de ordenação recursivo
- Ele recursivamente ordena as duas metades do vetor
- Usa a estratégia de divisão e conquista
- Mergesort é um algoritmo eficiente
- Tem tempo de execução $O(n \log n)$

Algoritmo Mergesort

- Caso o tamanho do vetor seja maior que 1
  1. divida o vetor no meio
  2. ordene a primeira metade recursivamente
  3. ordene a segunda metade recursivamente
  4. intercale as duas metades
- Senão devolva o elemento

Código do Mergesort

MergeSort (A, p, r)

```
if p < r then
    q = (p + q)/2
    Mergesort(A, p, q)
    Mergesort(A, q + 1, r)
    Merge(A, p, q, r)
```

Chamada Inicial: Mergesort$(A, 1, n)$

Intercalação

A intercalação de dois vetores ordenados pode ser feito em tempo linear

- Uma variável em cada vetor indica o próximo elemento a ser inserido a lista intercalada
- Enquanto ambas os vetores tiverem elementos
- Coloque o menor entre os dois elemento indicados no vetor intercalado e incremente índice respectivo
- Quando um dois vetores não tiver mais elementos, concatene o outro no final do vetor intercalado.

Análise da intercalação

Problema: Dados $A[p \mathinner{.\,.\,.} q]$ e $A[q{+}1 \mathinner{.\,.\,.} r]$ crescentes, rearranjar $A[p \mathinner{.\,.\,.} r]$ de modo que ele fique em ordem crescente.

Entrada:

	p				q				r
A	22	33	55	77	99	11	44	66	88

Saída:

	p				q				r
A	11	22	33	44	55	66	77	88	99

Intercalação

Merge (A, p, q, r)

for $i = p$ to q **do**
    $B[i] = A[i]$ $\triangleright$ $B[p \ldots r]$ vetor auxiliar
for $j = q + 1$ to r **do**
    $B[r + q + 1 - j] = A[j]$
$i = p$
$j = r$
for $k = p$ to r **do**
    if $B[i] \leq B[j]$ **then**
        $A[k] = B[i]$
        $i = i + 1$
    else
        $A[k] = B[j]$
        $j = j - 1$

Consumo de tempo

Se a execução de cada linha de código consome 1 unidade de tempo o consumo total é ($n := r - p + 1$):

linha		todas as execuções da linha
1	$=$	$q - p + 2 \ = \ n - r + q + 1$
2	$=$	$q - p + 1 \ = \ n - r + q$
3	$=$	$r - (q + 1) + 2 \ = \ n - q + p$
4	$=$	$r - (q + 1) + 1 \ = \ n - q + p - 1$
5	$=$	1
6	$=$	1
7	$=$	$r - p + 2 \ = \ n + 1$
8	$=$	$r - p + 1 \ = \ n$
9–13	$=$	$2\,(r - p + 1) \ = \ 2n$
total	$=$	$8n - 2(r - p + 1) + 5 = \ 6n + 5$

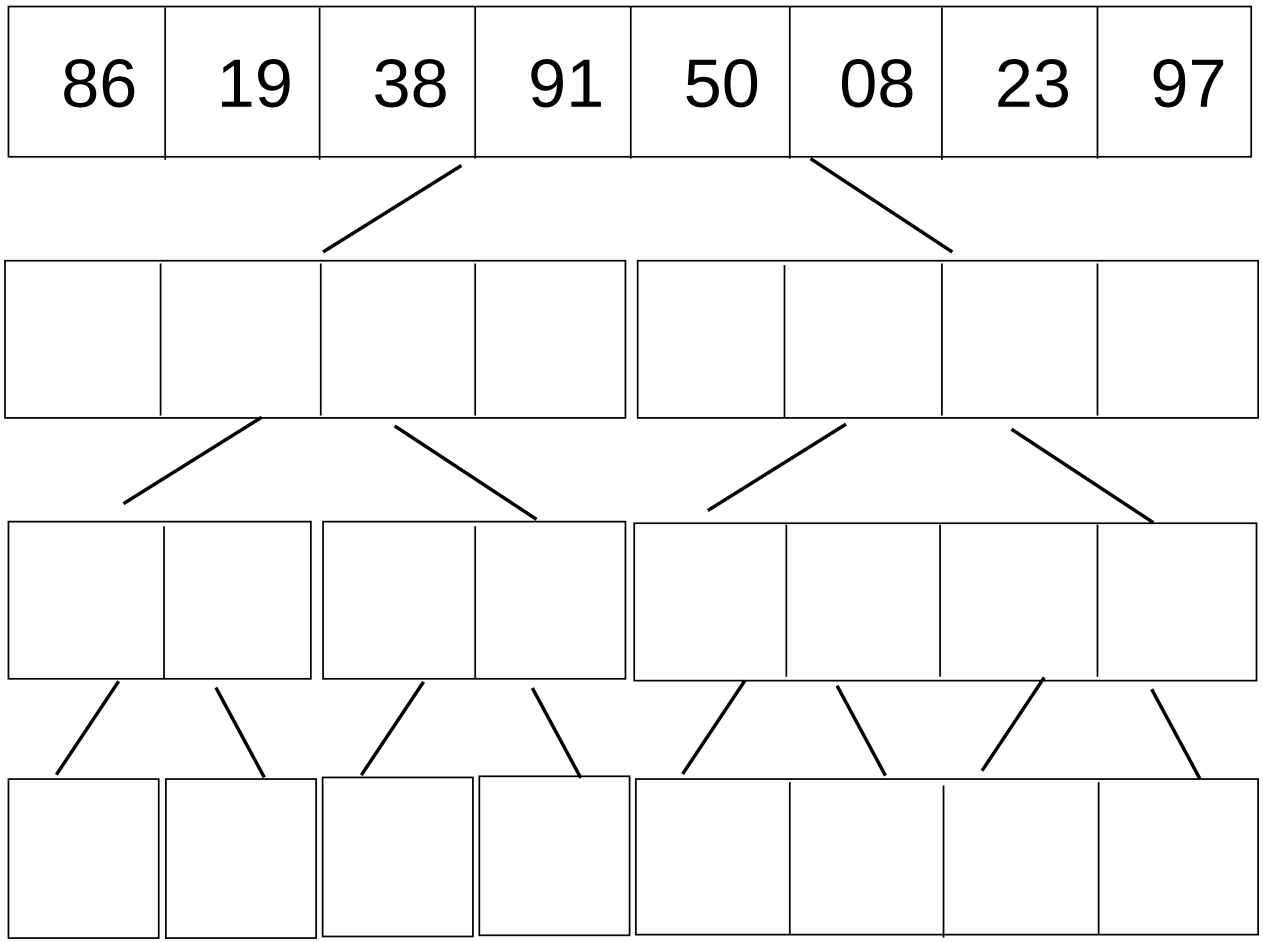
Consumo de tempo

Quanto tempo consome em função de $n := r - p + 1$?

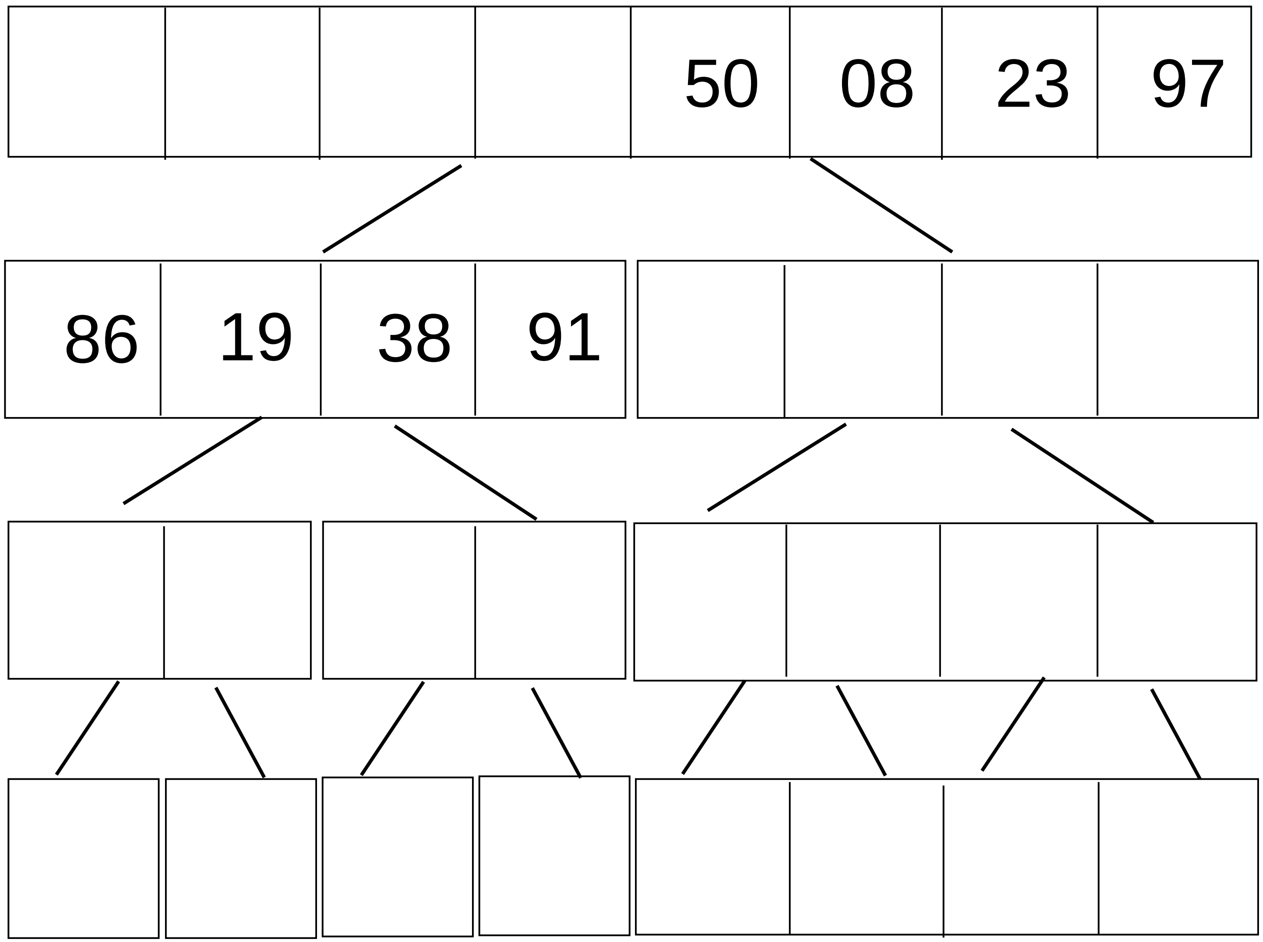
linha	consumo de todas as execuções da linha
1–4	$O(n)$
5–6	$O(1)$
7	$nO(1) = O(n)$
8	$nO(1) = O(n)$
9–13	$nO(1) = O(n)$
total	$O(4n+1)$ = $O(n)$

Conclusão:
O algoritmo consome $O(n)$ unidades de tempo.

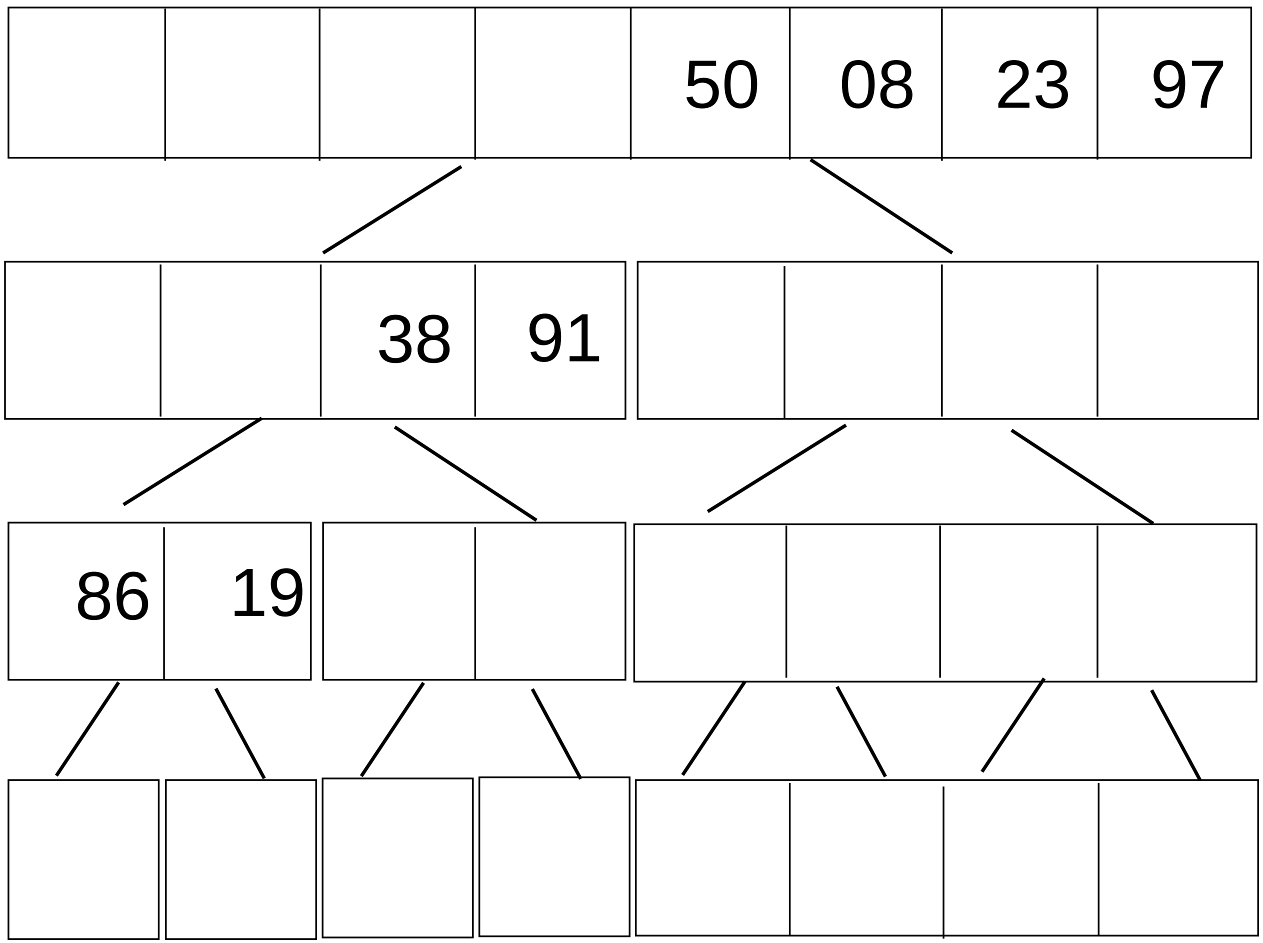
mergesort



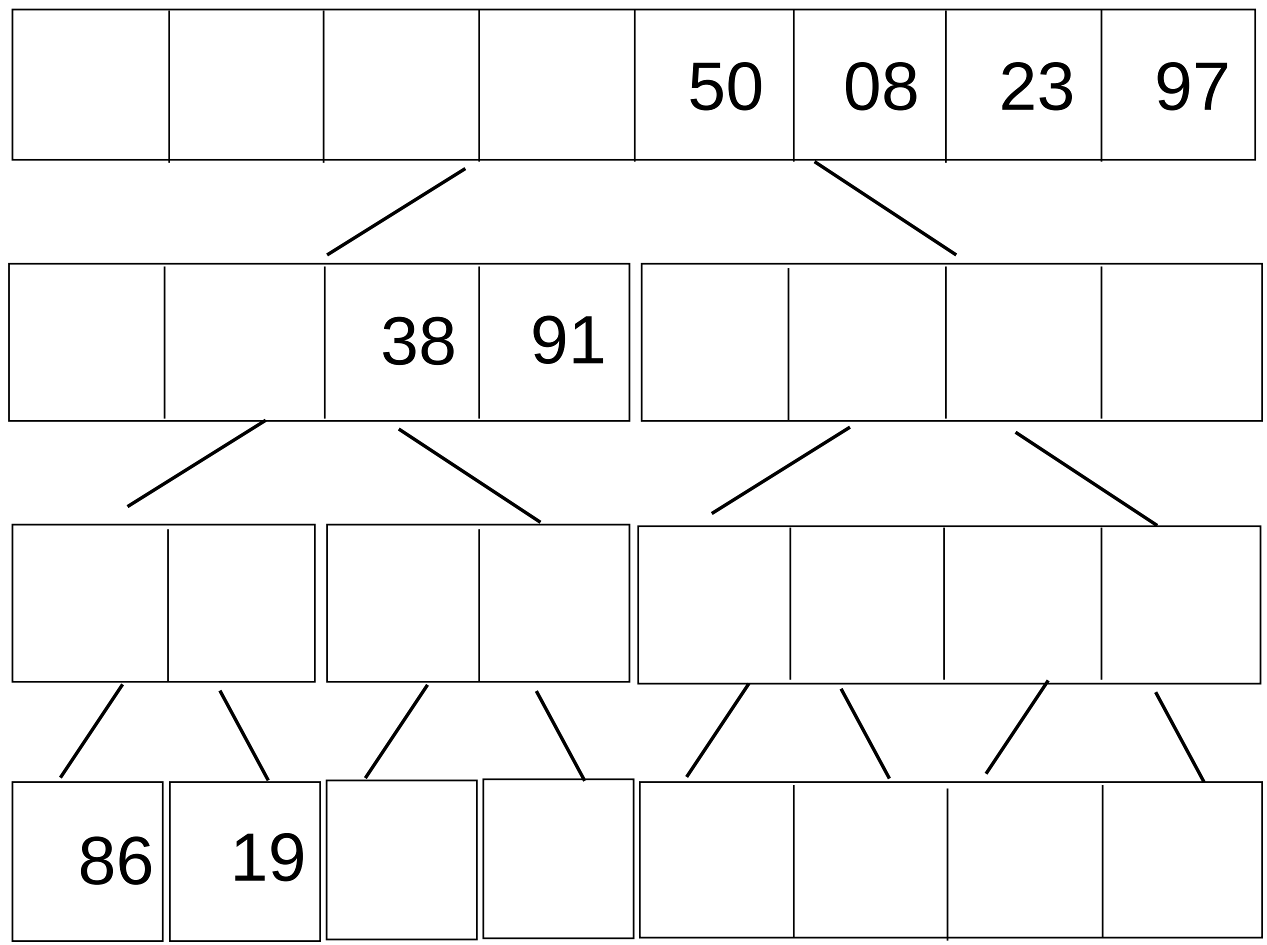
mergesort



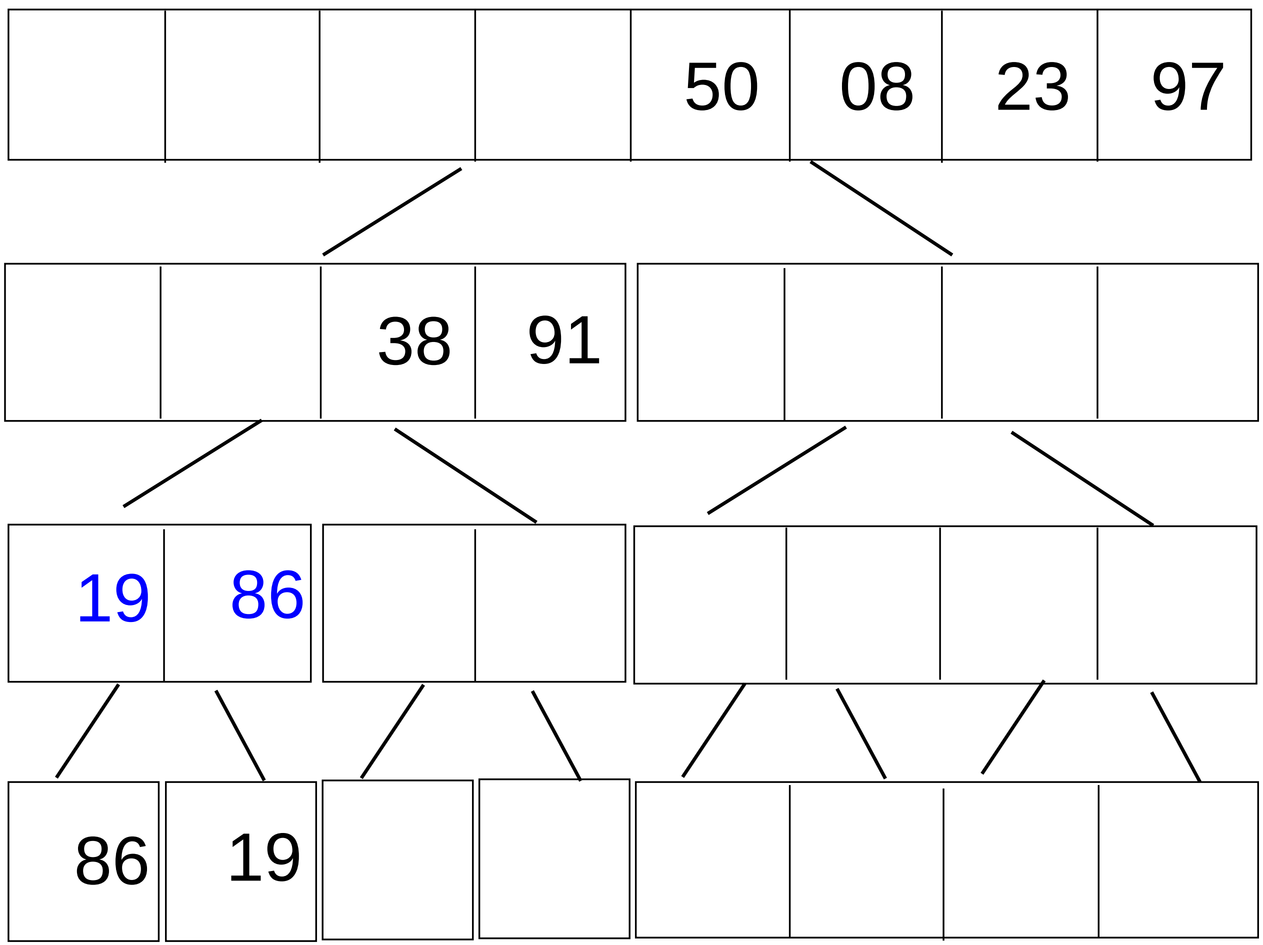
mergesort



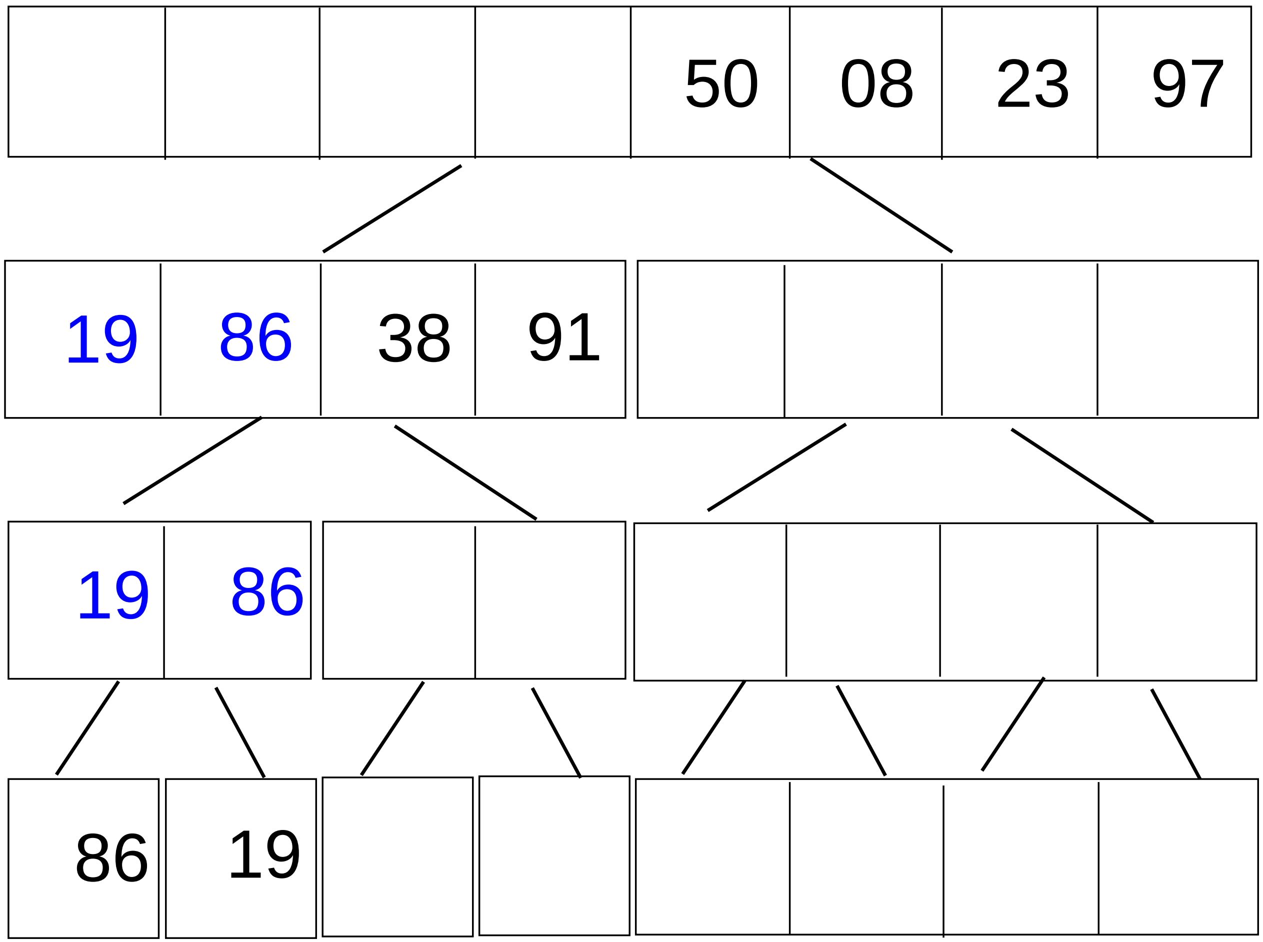
mergesort



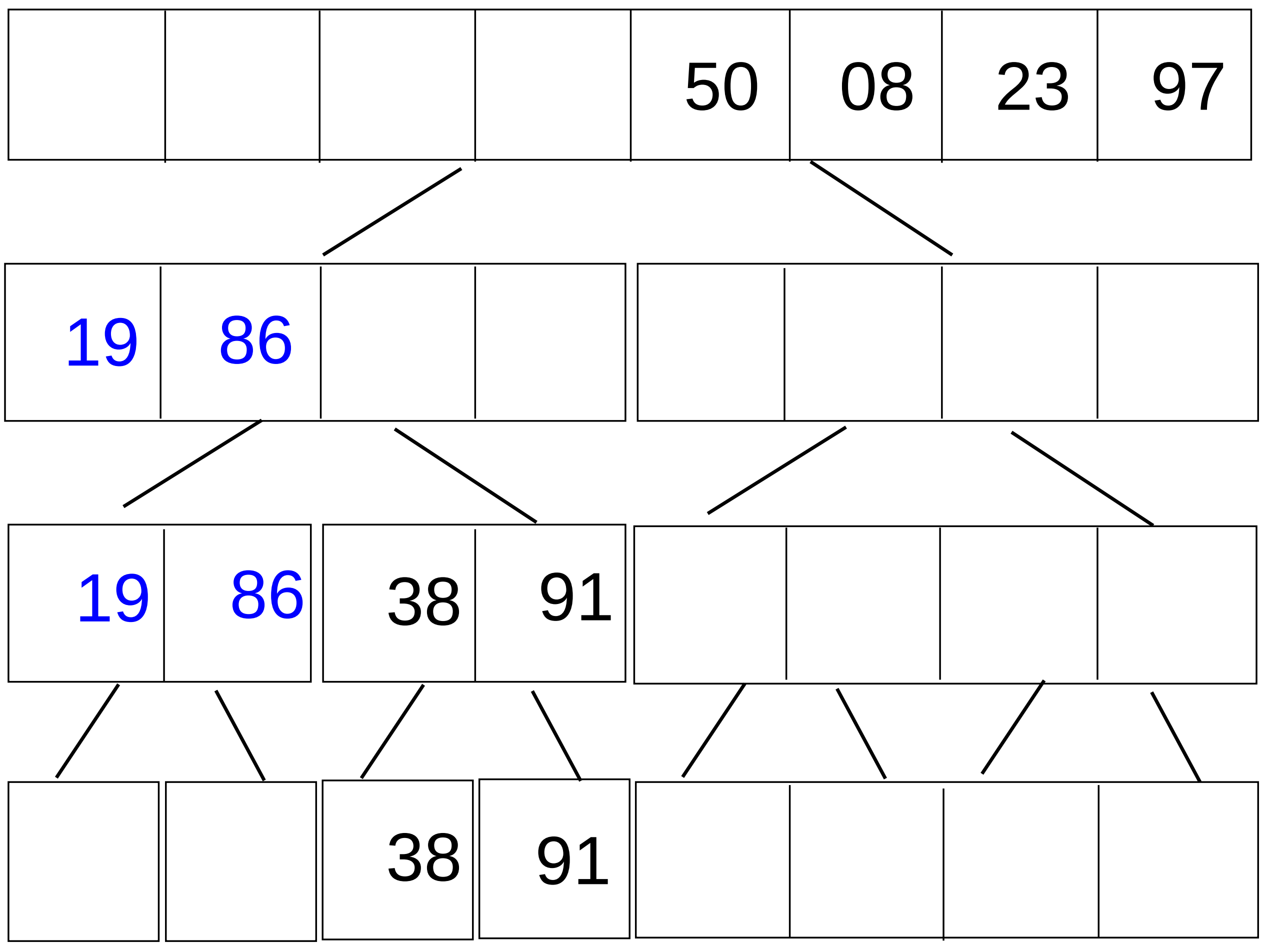
mergesort



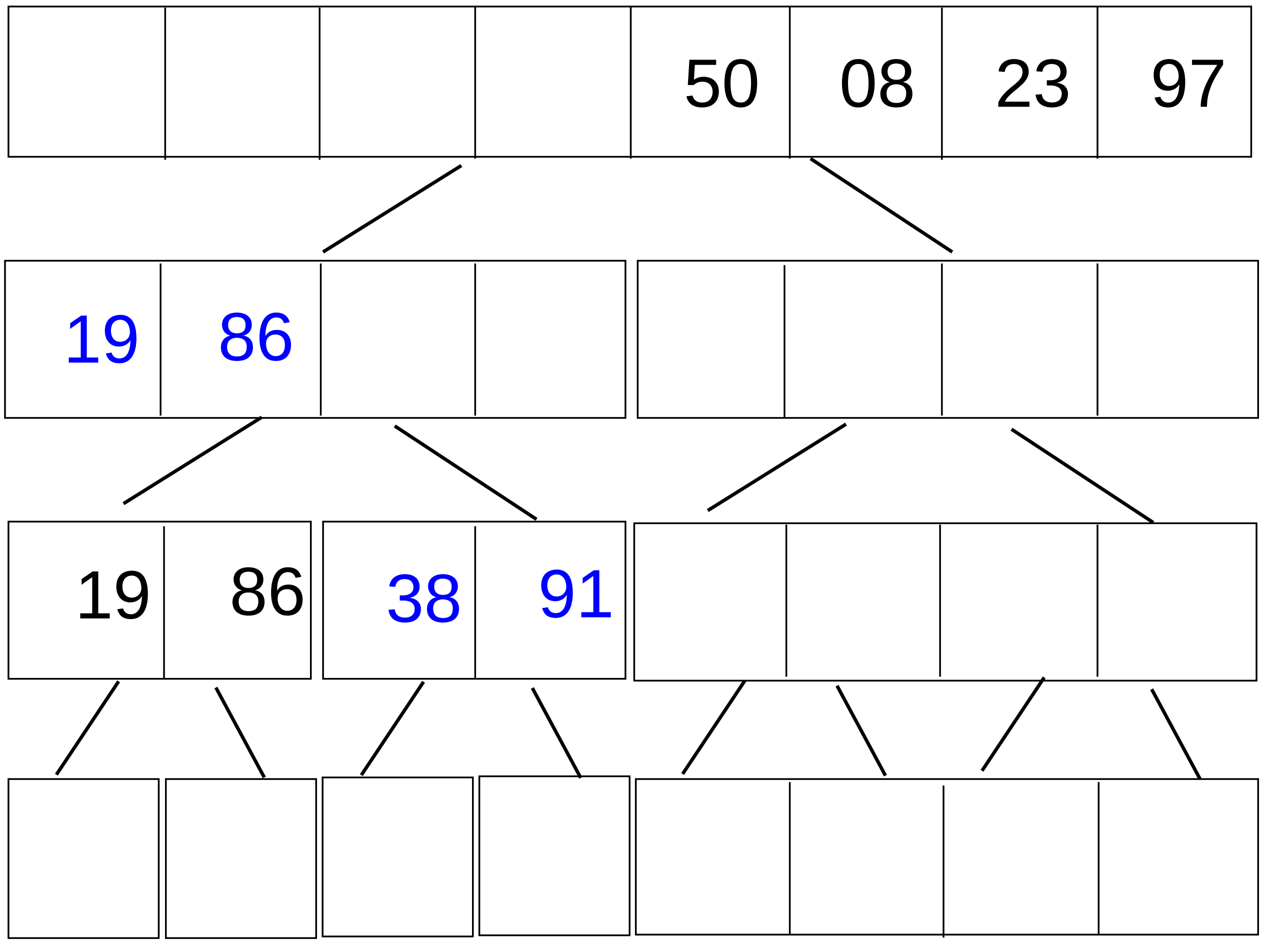
mergesort



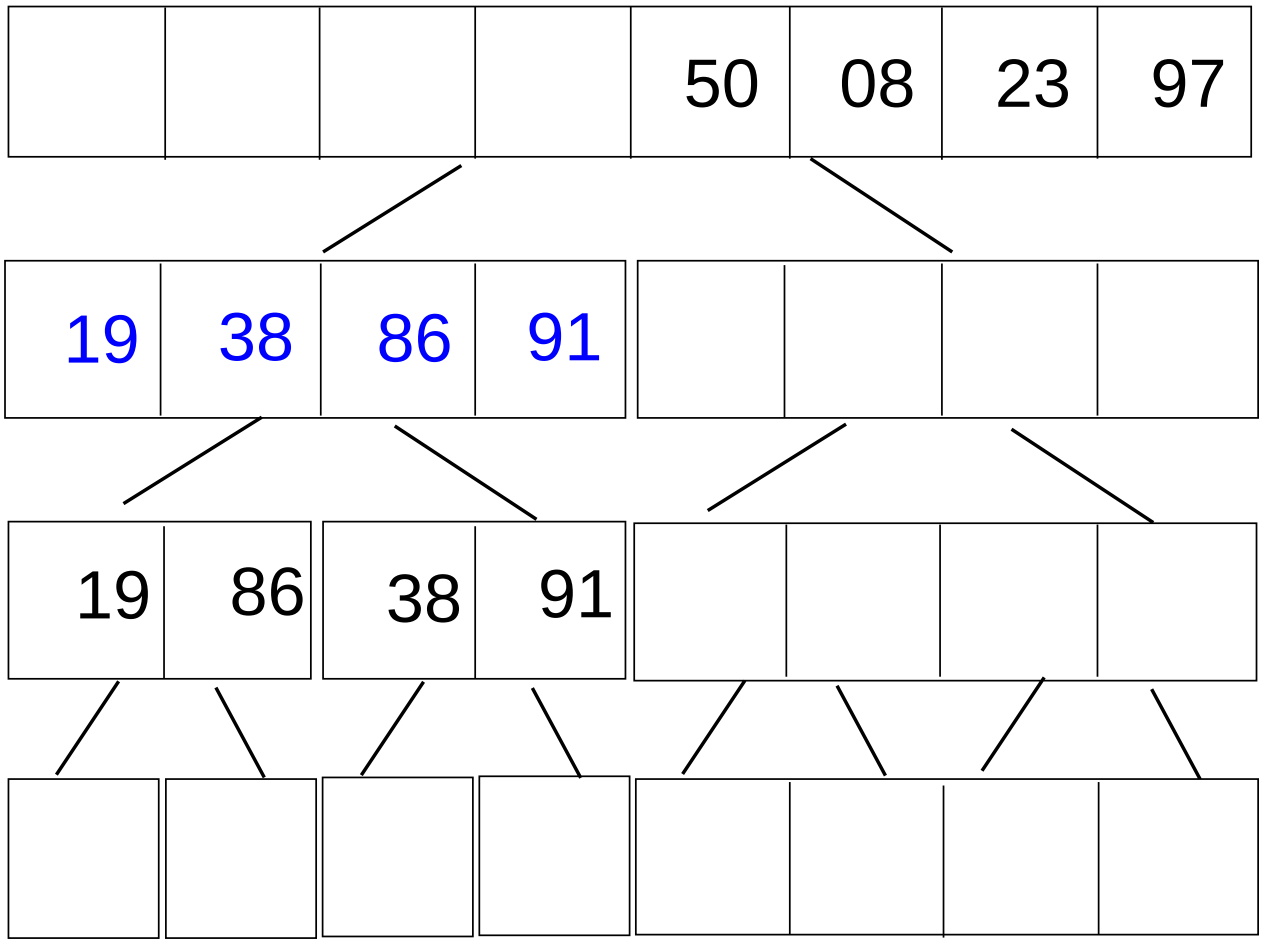
mergesort



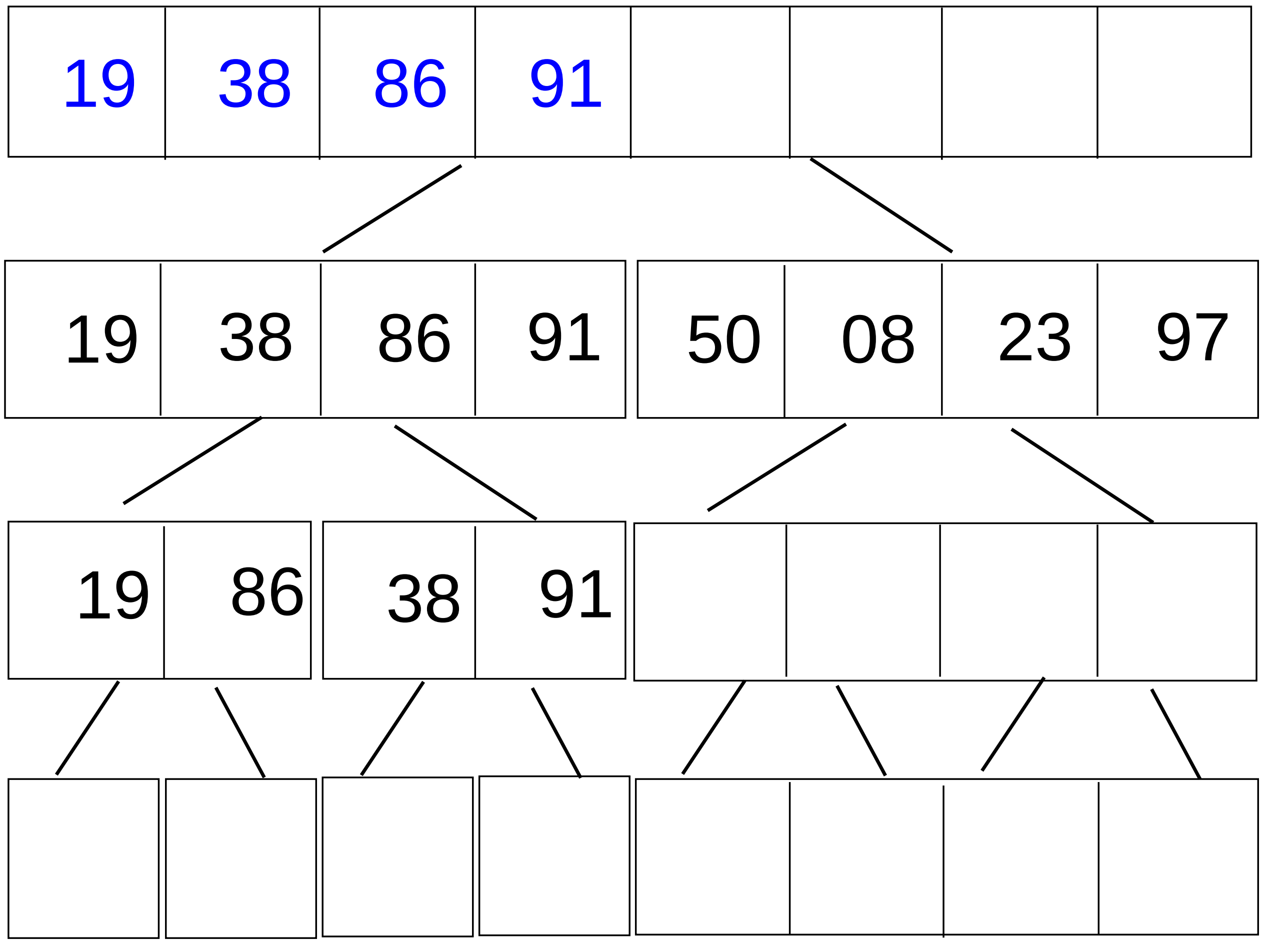
mergesort



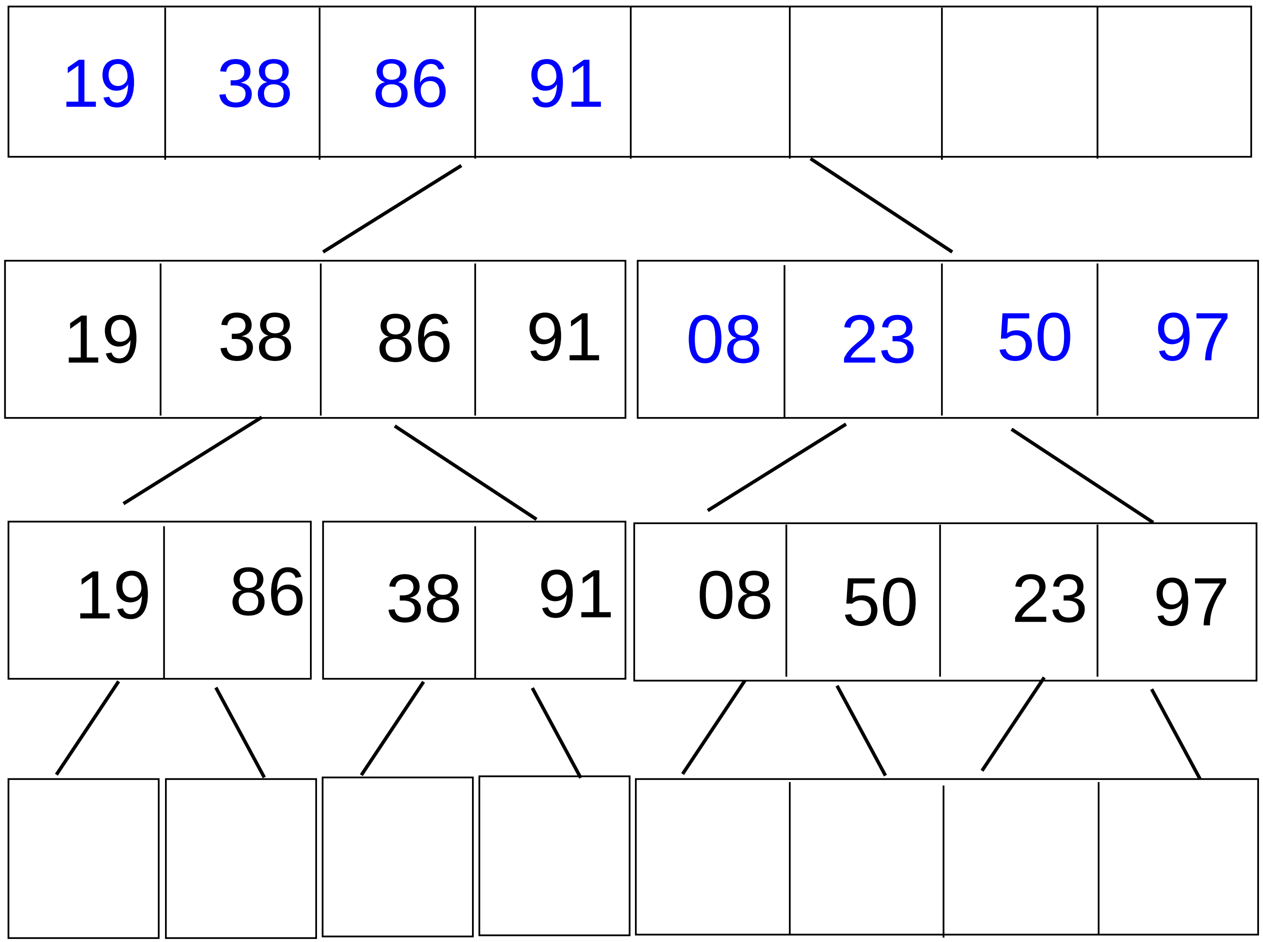
mergesort



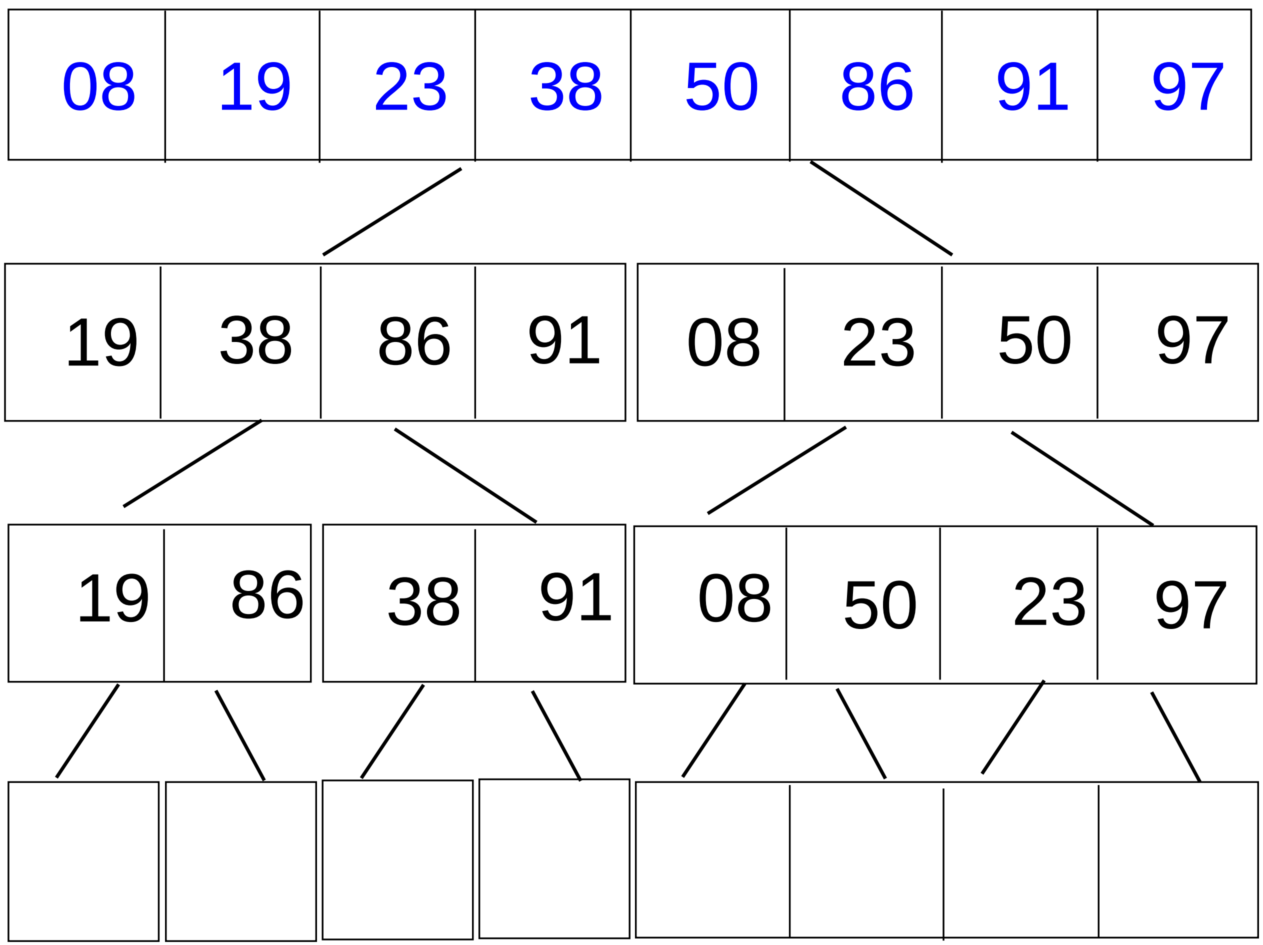
mergesort



mergesort



mergesort



Correção do Merge-Sort

Teorema O Mergesort está correto.

Prova: Indução em n (tamanho da entrada)

Base $n = 1$, $r - p = 1 \Rightarrow r = p + 1 \Rightarrow q = \lceil \frac{2p+1}{2} \rceil = p$. Então ambas as chamadas recursivas voltam sem alterar o vetor e o Procedimento Merge intercala um elemento. Ok!

Hipótese de Indução Supor que o algoritmo está correto para toda entrada menor que n.

Passo Indutivo Na Linha 2 q assume o indice do meio do vetor. Na Linha 3 e 4, o Algoritmo é chamado para as duas metades do vetor. Como pela Hipótese de Indução ambos resolvem suas respectivas metades, na Linha 5 o Merge intercala duas listas ordenadas e portanto segue o resultado.

(Provar que o Procedimento Merge está correto Exerc.)

Análise de Divisão e Conquista

- Quando um algoritmo contém chamadas recursivas, o cálculo de seu tempo de execução pode usar recorrências.
- Para o método de Divisão e Conquista
- Seja $T(n)$ o tempo do algoritmo. Suponha que dividimos em a subproblemas de tamanho n/b cada e seja $D(n)$ o tempo para dividir os subproblemas e $C(n)$ o tempo para combiná-los. Então

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq c \\ aT(n/b) + D(n) + C(n) & \text{se } n > c \end{cases}$$

Análise do Merge-Sort

- **Dividir** Tempo Constante $\Theta(1)$
- **Conquista** Dois problemas de $n/2$ cada: $2T(n/2)$
- **Combina** Tempo do Merge $\Theta(n)$

$$T(n) = \begin{cases} \Theta(1) & \text{se} n = 1 \\ 2T(n/2) + \Theta(n) & \text{se } n > 1 \end{cases}$$

Como Resolver?

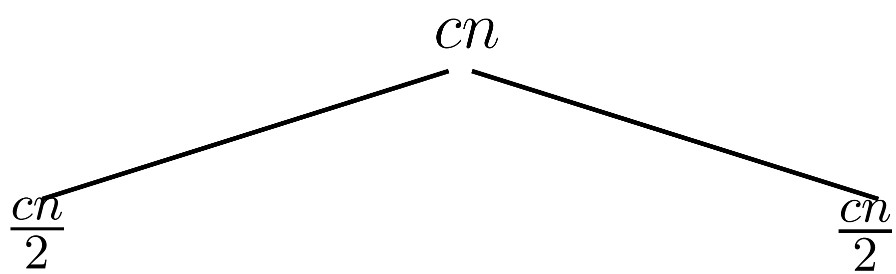
Análise do Merge-Sort

- **Dividir** Tempo Constante $\Theta(1)$
- **Conquista** Dois problemas de $n/2$ cada: $2T(n/2)$
- **Combina** Tempo do Merge $\Theta(n)$

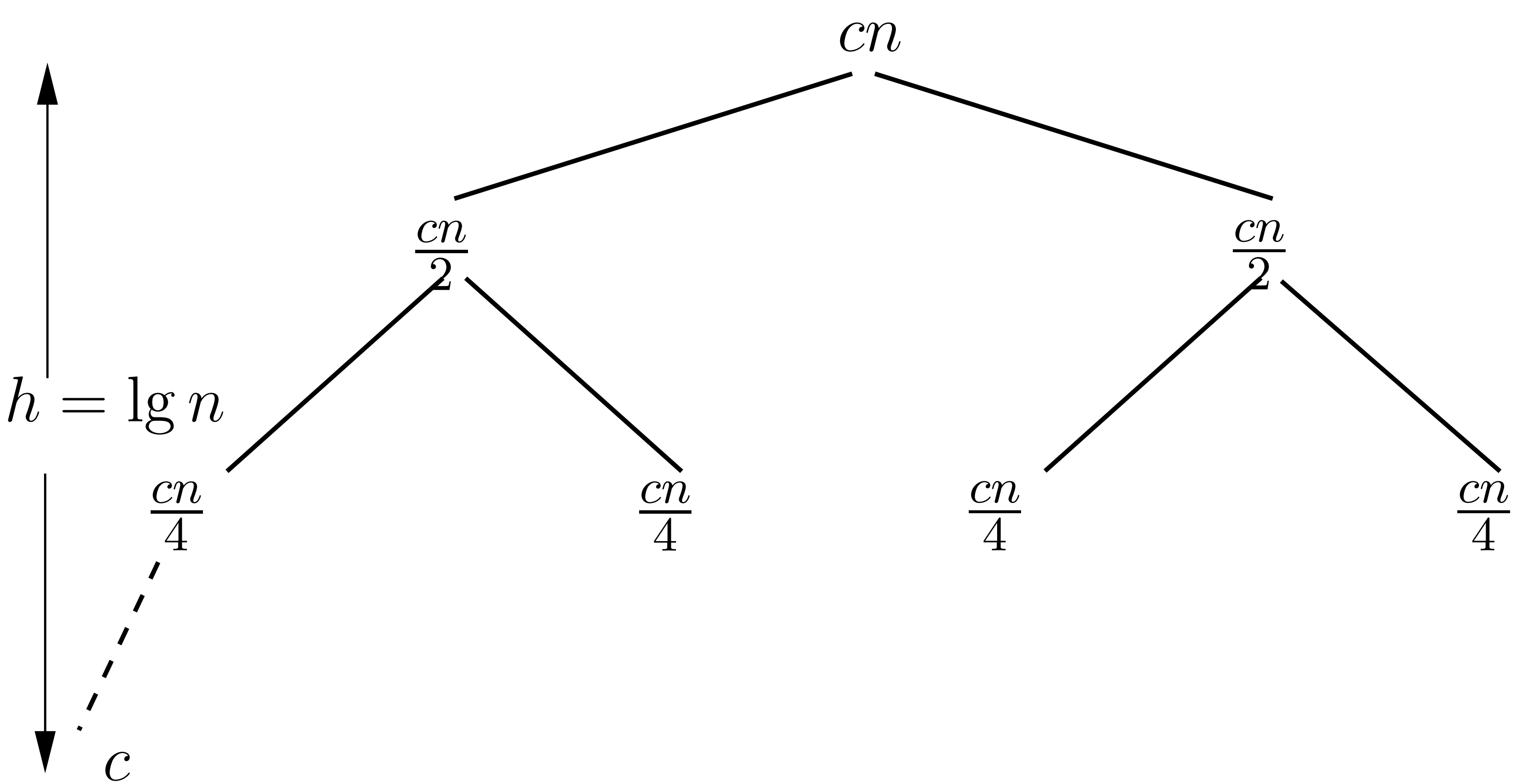
$$T(n) = \begin{cases} \Theta(1) & \text{se} n = 1 \\ 2T(n/2) + \Theta(n) & \text{se } n > 1 \end{cases}$$

Como Resolver? Árvore de Recursão

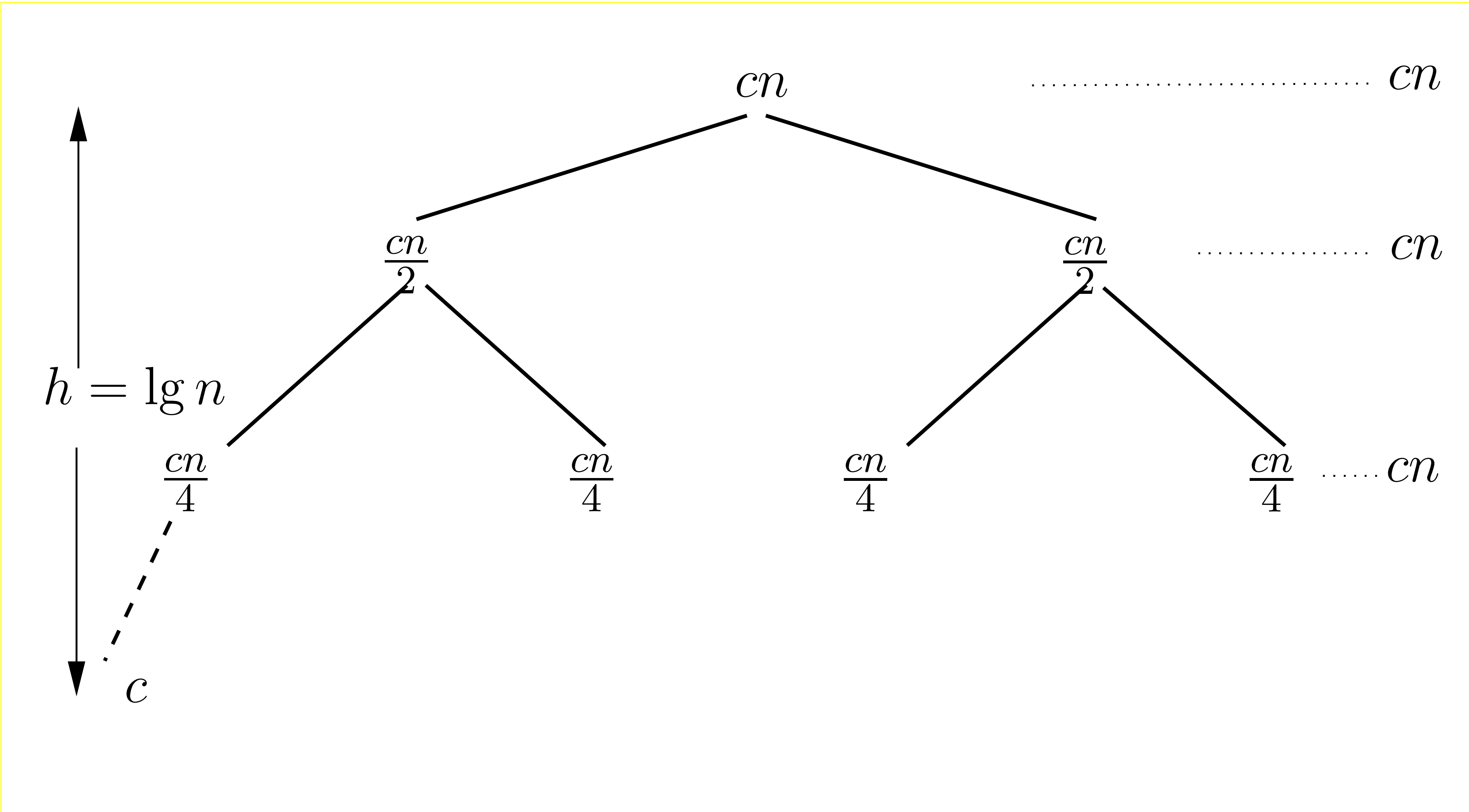
Árvore de Recursão



Árvore de Recursão



Árvore de Recursão



Árvore de Recursão