Algoritmos de Ordenação (Parte 3)

Prof. Jefferson T. Oliva

Algoritmos e Estrutura de Dados I (AE22CP)
Engenharia de Computação
Departamento Acadêmico de Informática (Dainf)
Universidade Tecnológica Federal do Paraná (UTFPR)
Campus Pato Branco

DAINF
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CÂMPUS PATO BRANCO

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Sumário

- *Shell sort*
- *Mergesort*

Sumário

Shell sort

- Extensão do *insertion sort*
- Contorna o principal problema do *insertion sort* possibilitando troca de registros que estão distantes um do outro
- Tem como objetivo aumentar o passo de movimento dos elementos ao invés das posições adjacentes
- Consiste em classificar sub-arranjos do original
- Esses sub-arranjos contêm todo h-ésimo elemento do arranjo original
- o valor de h é chamado de incremento

Shell sort

- Por exemplo, se h é 5, o sub-arranjo consiste dos elementos $x[0]$, $x[5]$, $x[10]$, etc
  - Sub-arranjo 1: $x[0]$, $x[5]$, $x[10]$
  - Sub-arranjo 2: $x[1]$, $x[6]$, $x[11]$
  - Sub-arranjo 3: $x[2]$, $x[7]$, $x[12]$
  - Sub-arranjo 4: $x[3]$, $x[8]$, $x[13]$

- Após a ordenação dos sub-arranjos:
  - Define-se um novo incremento menor que o anterior
  - Gera-se novos sub-arquivos
  - Aplica-se novamente o método da inserção

Shell sort

- O processo é realizado repetidamente até que h seja igual a 1
- O valor de h pode ser definido de várias formas
  - $h(s) = 3h(s-1) + 1$, para $s > 1$
  - $h(s) = 1$, para $s = 1$
- Nessa recorrência, s é o tamanho do conjunto

Shell sort

- Implementação

```
void shellsort(int v[], int n){
    int h = 1;
    int x, i, j;

    while (h < n)
       h = 3 * h + 1;

    h /= 3;

    while (h >= 1){
       for (i = h; i < n; i++){
          x = v[i];
          j = i;

          while ((j >= h) && (x < v[j - h])){
              v[j] = v[j - h];

              j -= h;
          }

          v[j] = x;
       }

       h /= 3;
    }
}
```

Shell sort

- Exemplo

x	h	i	v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]
-	-	-	25	57	48	37	12	92	86	33
12	4	4	**25**	57	48	37	**12**	92	86	33
92	4	5	12	**57**	48	37	25	**92**	86	33
86	4	6	12	57	**48**	37	25	92	**86**	33
33	4	7	12	57	48	**37**	25	92	86	**33**
33	4	8	12	57	48	33	25	92	86	37
57	1	1	**12**	**57**	48	33	25	92	86	37
48	1	2	**12**	**57**	**48**	33	25	92	86	37
33	1	3	**12**	**48**	**57**	**33**	25	92	86	37
25	1	4	**12**	**33**	**48**	**57**	**25**	92	86	37
92	1	5	**12**	**25**	**33**	**48**	**57**	**92**	86	37
86	1	6	**12**	**25**	**33**	**48**	**57**	**92**	**86**	37
37	1	7	12	25	33	48	57	86	92	**37**
37	1	8	12	25	33	37	48	57	86	92

- Quando $h = 1$, o comportamento é o mesmo em comparação com o *insertion sort*

Shell sort

- Um problema com o *shell sort* ainda não resolvido é a escolha dos incrementos que fornece os melhores resultados
- É desejável que ocorra o maior número possível de interações entre as diversas cadeias
- Ótima opção para arquivos de tamanho moderado
- Tempo de execução sensível à ordem dos dados
- Não estável

Shell sort

- A complexidade do algoritmo ainda não é conhecida
- Acredita-se que o custo de tempo é por volta de:
  - Pior caso: $O(n^2)$
  - Melhor caso: proporcional a $O(n^{1,25})$
- Links interessante:
  `https://www.youtube.com/watch?v=CmPA7zE8mx0`

Ordenação por Intercalação (*Mergesort*)

Ordenação por Intercalação (*Mergesort*)

- Divisão e conquista
- Se o vetor tiver o tamanho igual a 1, a função apenas retorna o elemento
- Caso o tamanho do vetor seja maior que 1
  1. **Divisão**: divida o vetor ao meio
  2. **Conquista 1**: ordene a primeira metade recursivamente
  3. **Conquista 2**: ordene a segunda metade recursivamente
  4. **Combinação**: intercale as duas metades

Ordenação por Intercalação (*Mergesort*)

- Implementação

```
void mergesort(int v[], int esq, int dir){
  int meio;
  if (esq < dir){
    meio = (esq + dir) / 2;

    mergesort(v, esq, meio);
    mergesort(v, meio + 1, dir);
    merge(v, esq, meio, dir);
  }
}
```

Ordenação por Intercalação (*Mergesort*)

- Implementação

```
void merge(int v[], int esq, int meio, int dir){
   int i, j, k;
   int n1 = meio - esq + 1;
   int n2 = dir - meio;
   int L[n1 + 1];
   int R[n2 + 1];

   for (i = 0; i < n1; i++)
     L[i] = v[esq + i];

   for (j = 0; j < n2; j++)
     R[j] = v[meio + j + 1];

   L[n1] = INT_MAX;
   R[n2] = INT_MAX;
   i = 0;
   j = 0;

   for (k = esq; k <= dir; k++)
     if (L[i] <= R[j]){
        v[k] = L[i];
        i++;
     }else{
        v[k] = R[j];
        j++;
     }
}
```

Ordenação por Intercalação (*Mergesort*)

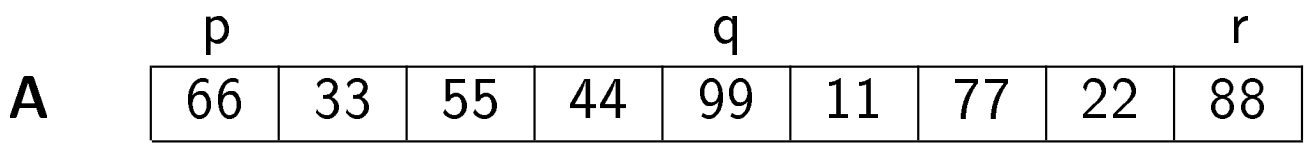
- Entrada

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

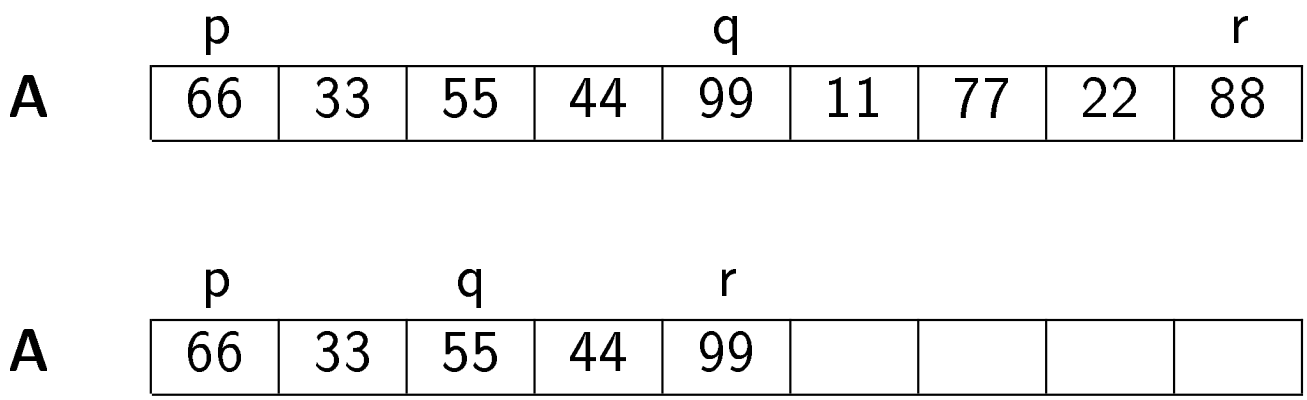
- Saída

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

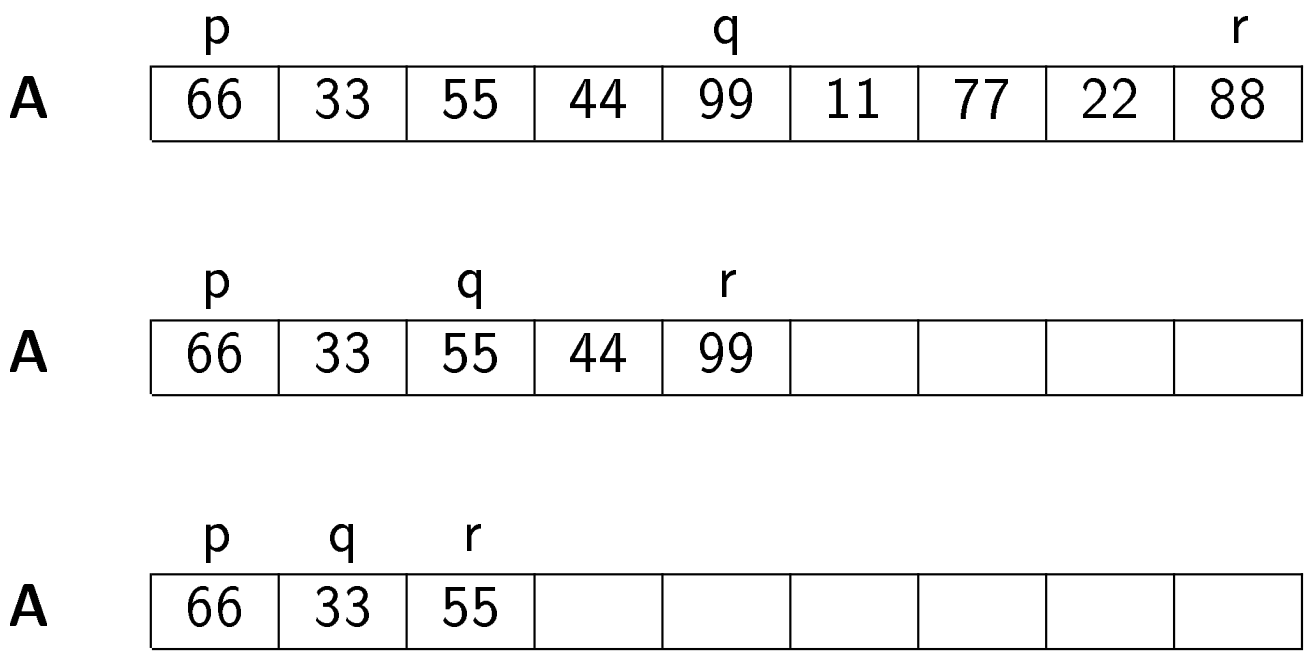
Ordenação por Intercalação (*Mergesort*)



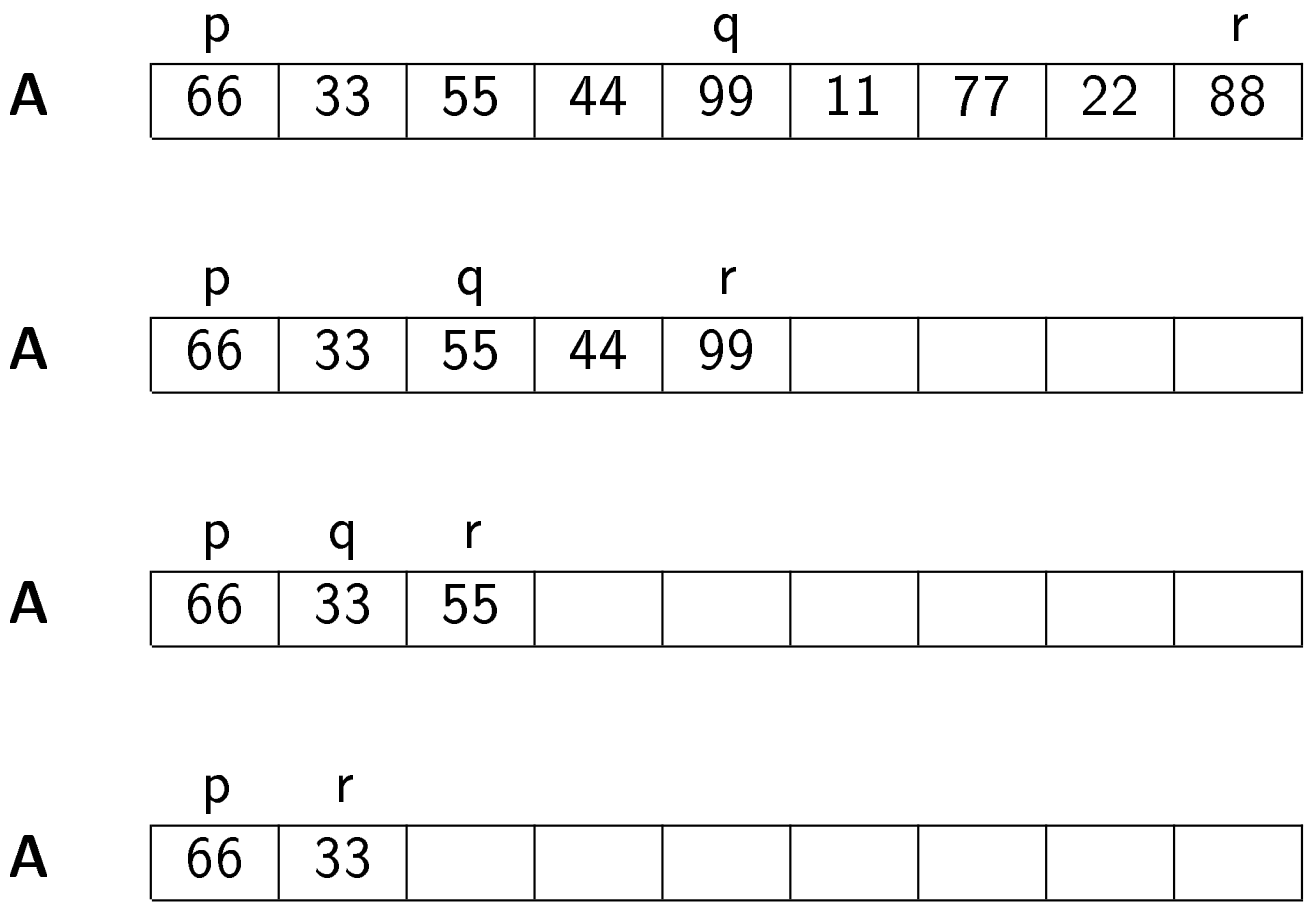
Ordenação por Intercalação (*Mergesort*)



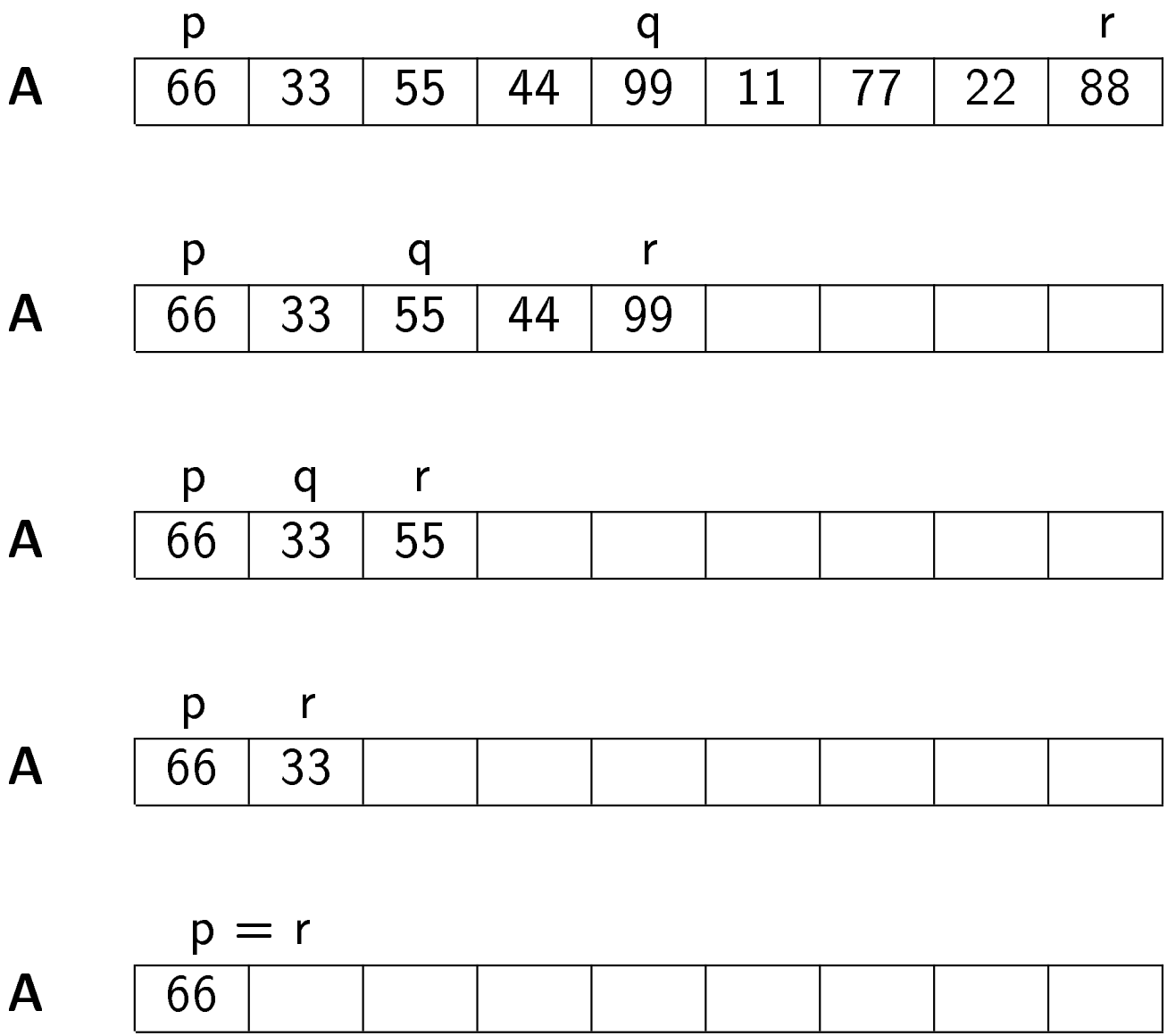
Ordenação por Intercalação (*Mergesort*)



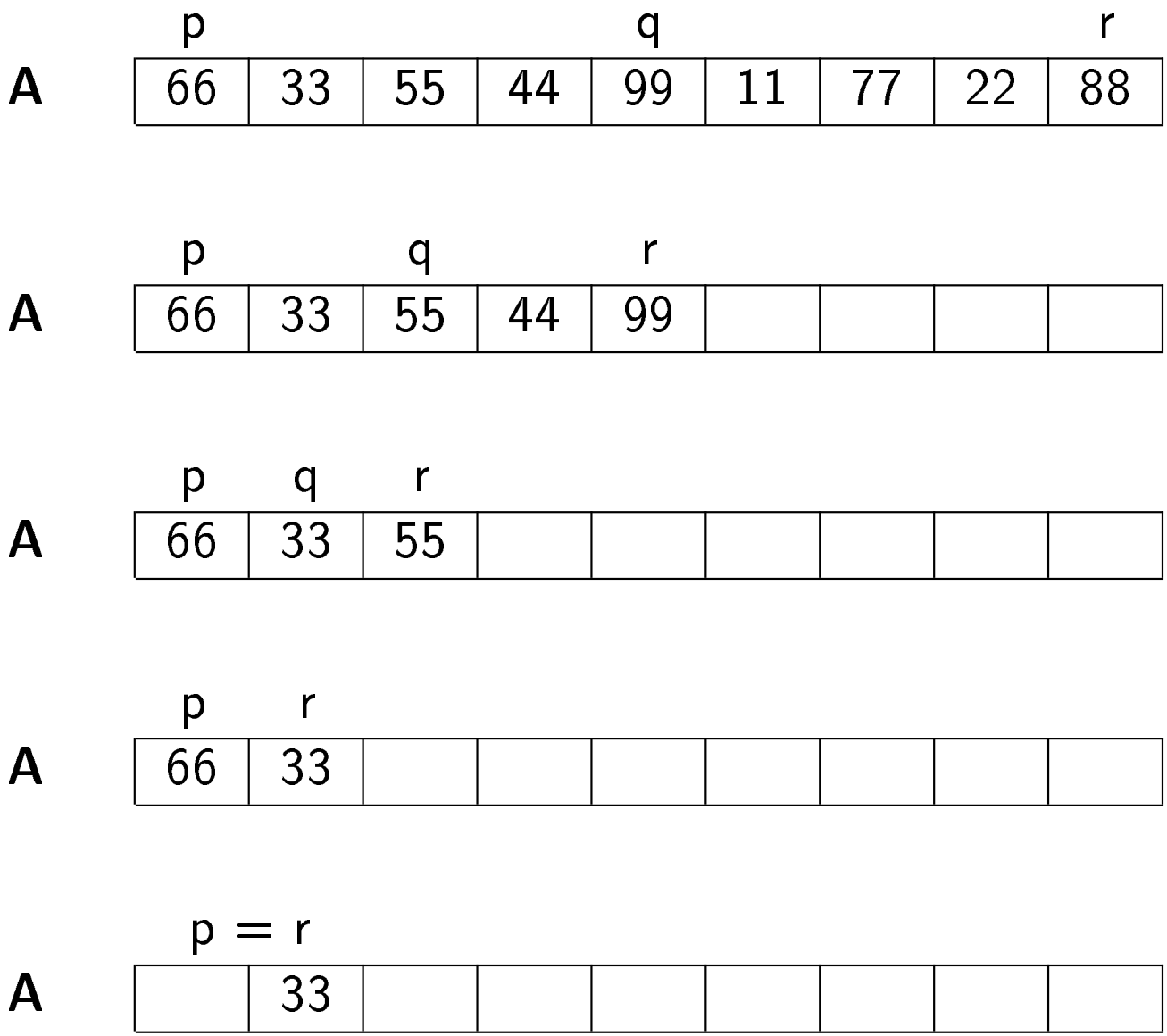
Ordenação por Intercalação (*Mergesort*)



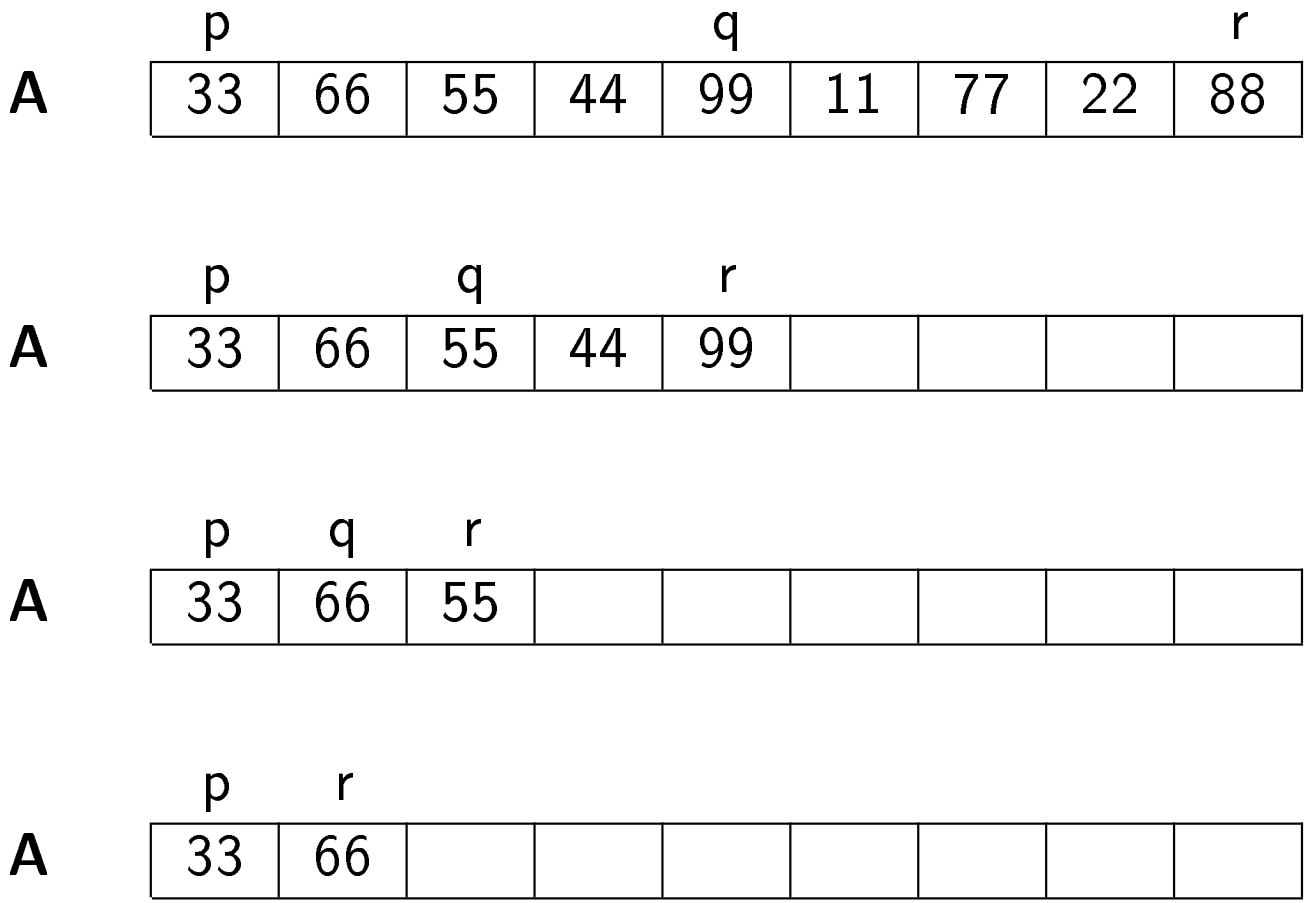
Ordenação por Intercalação (*Mergesort*)



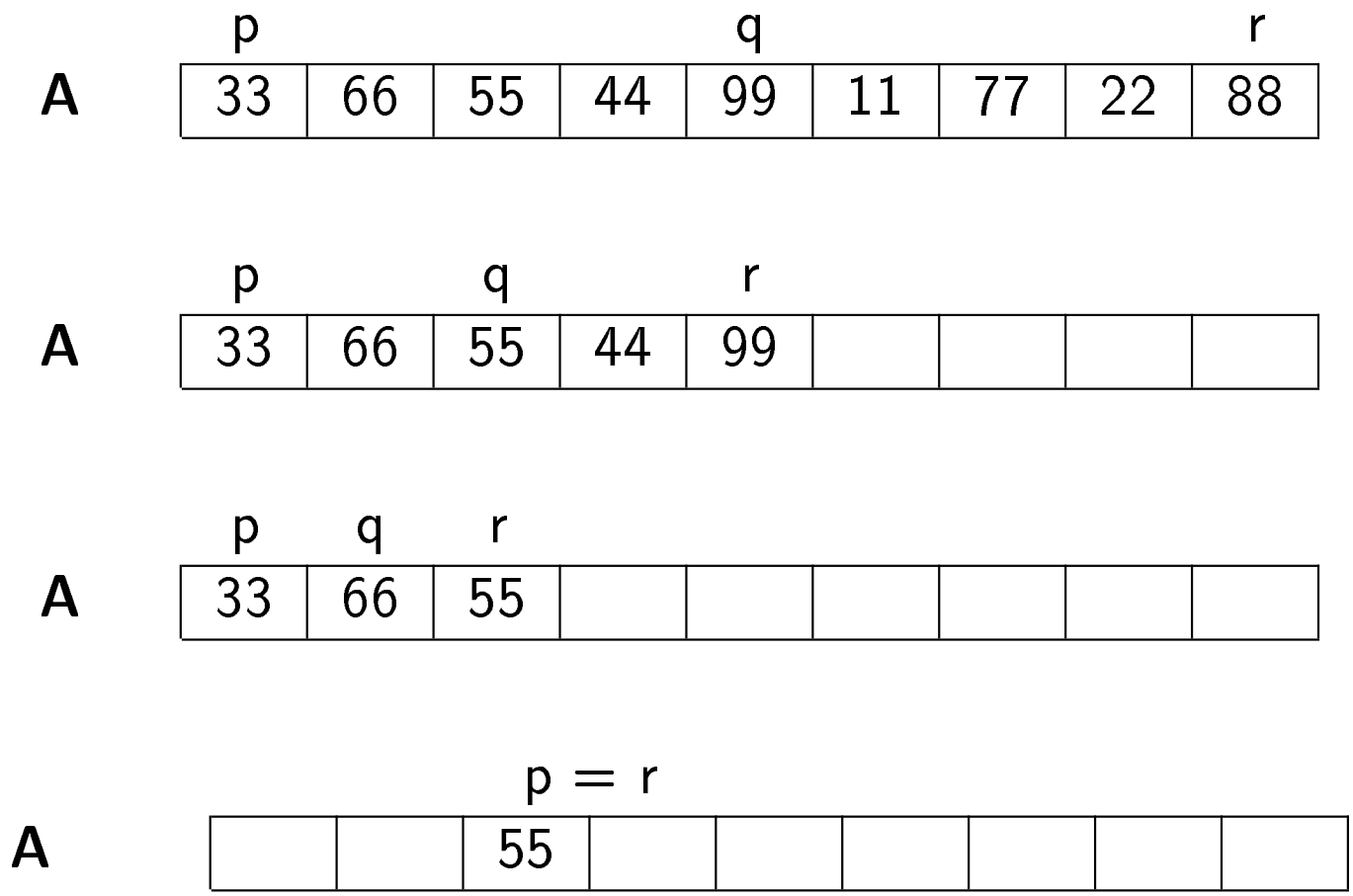
Ordenação por Intercalação (*Mergesort*)



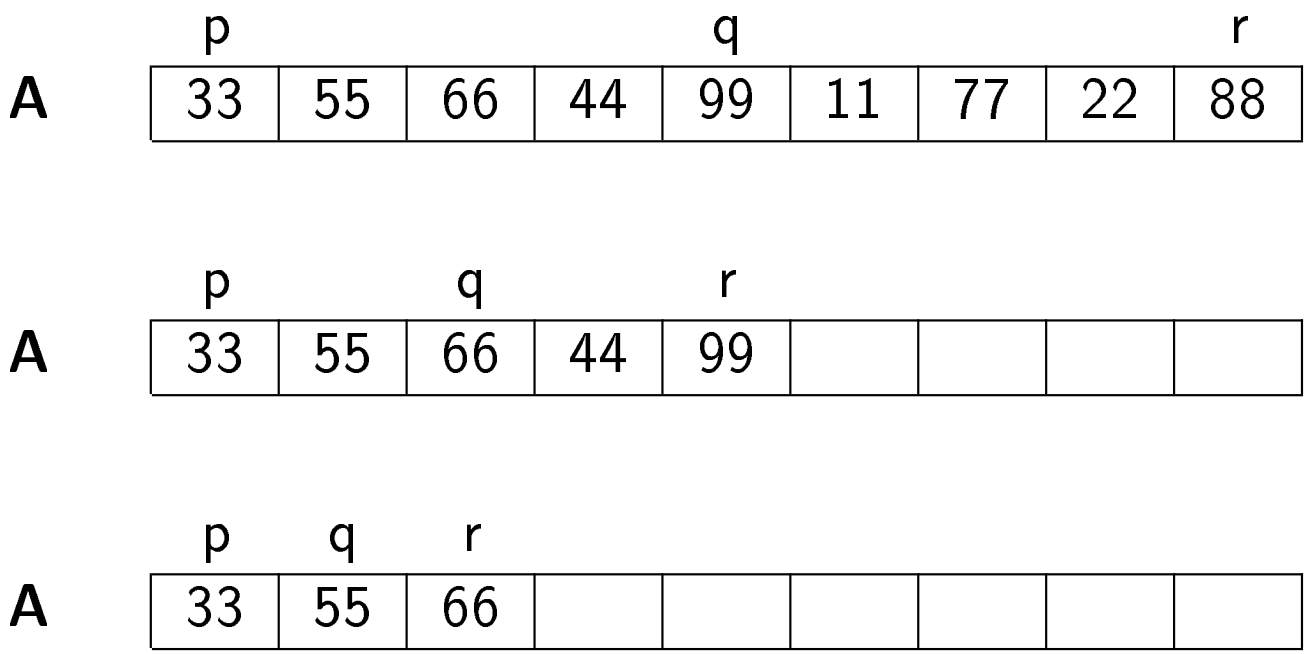
Ordenação por Intercalação (*Mergesort*)



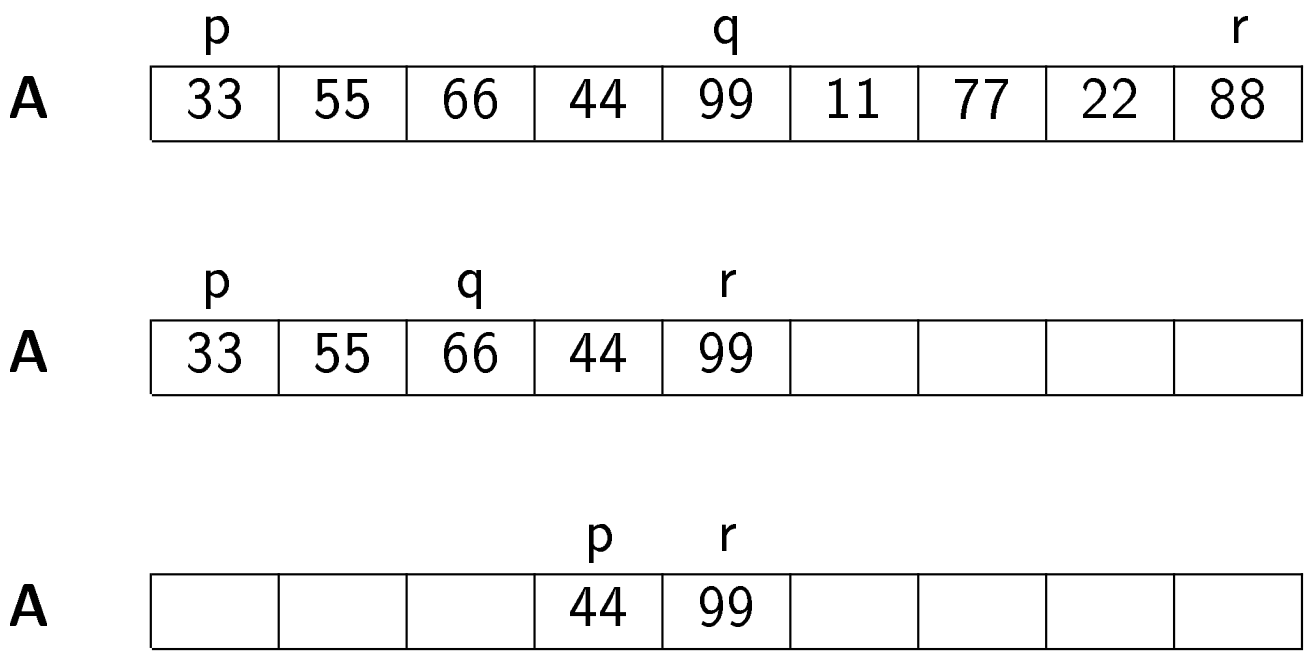
Ordenação por Intercalação (*Mergesort*)



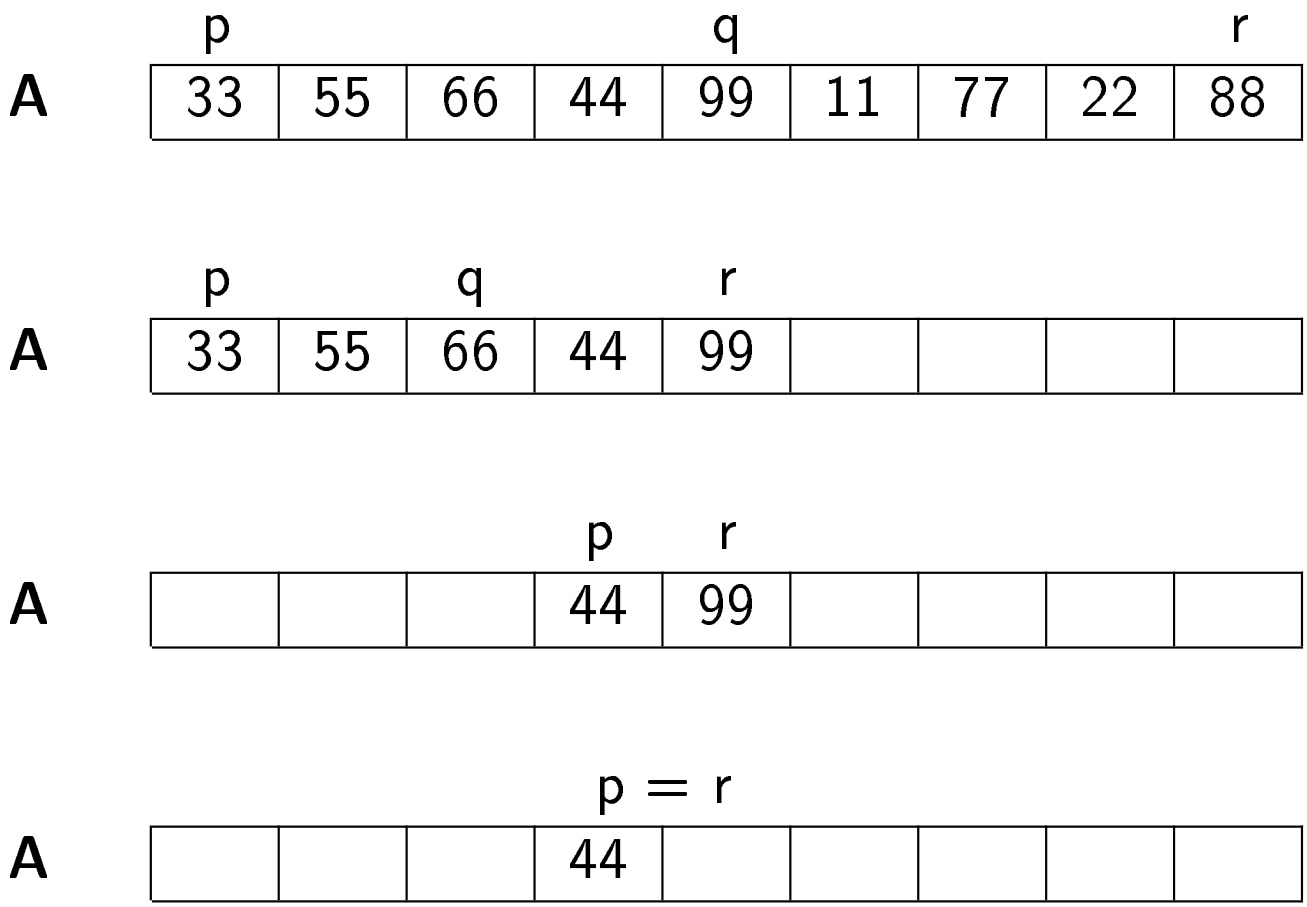
Ordenação por Intercalação (*Mergesort*)



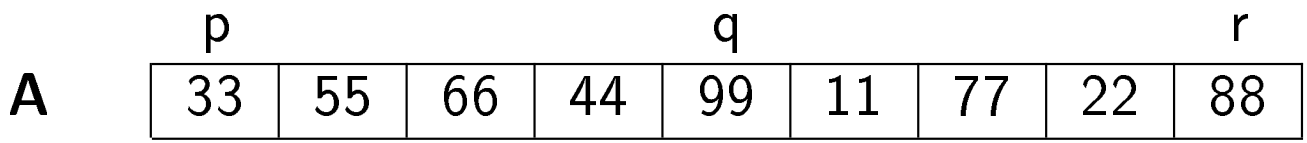
Ordenação por Intercalação (*Mergesort*)

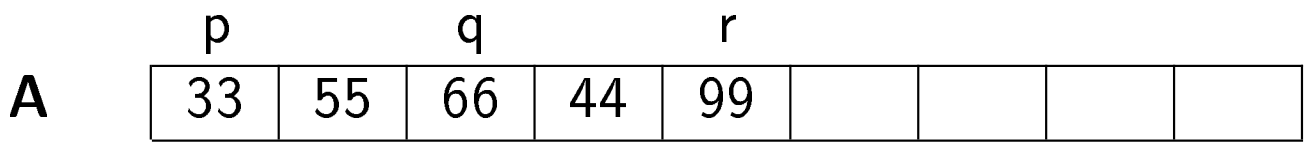


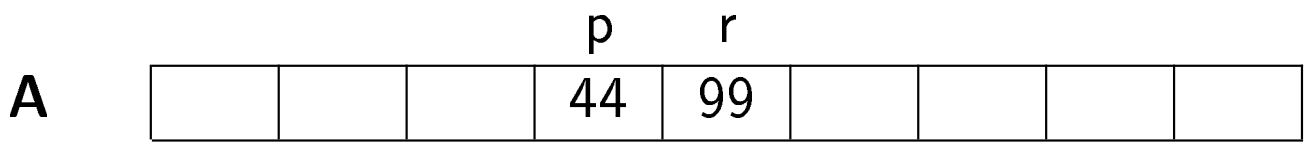
Ordenação por Intercalação (*Mergesort*)

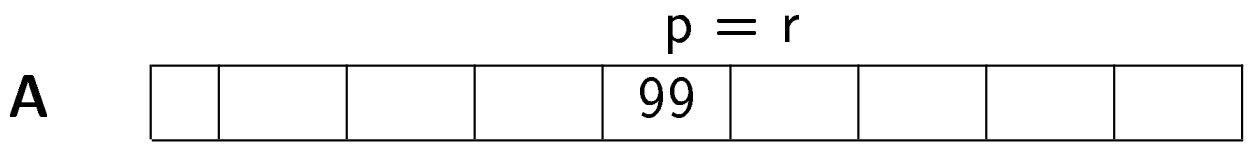


Ordenação por Intercalação (*Mergesort*)

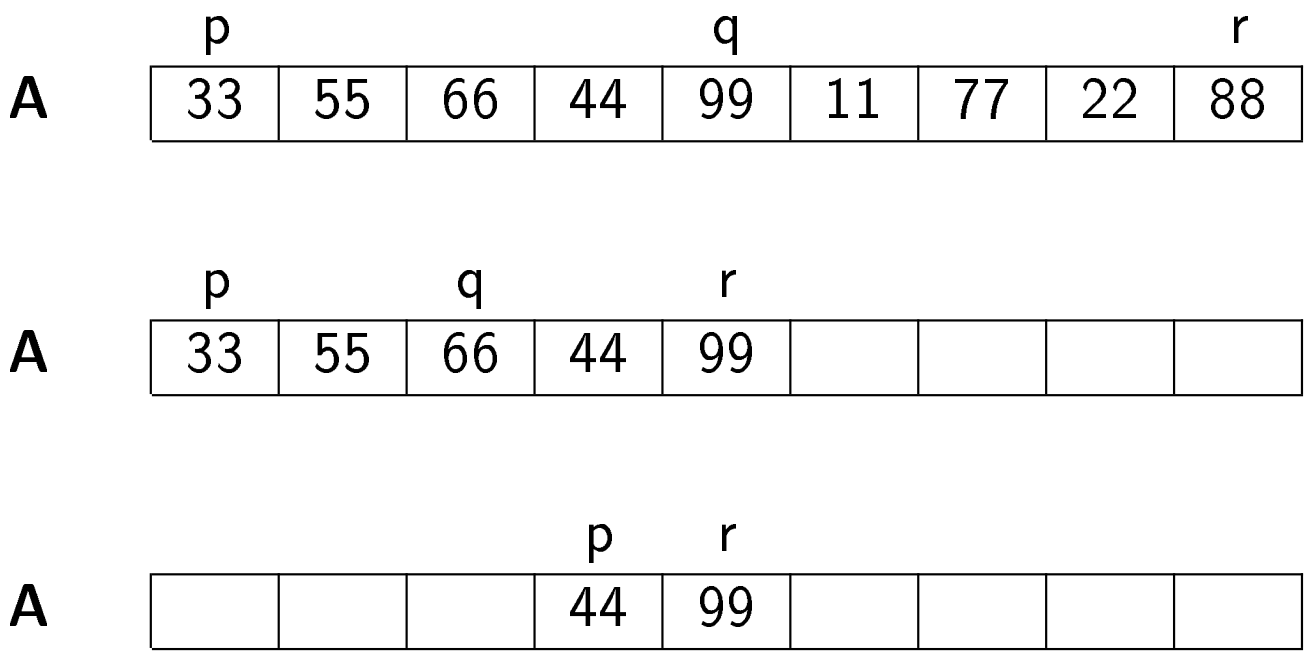




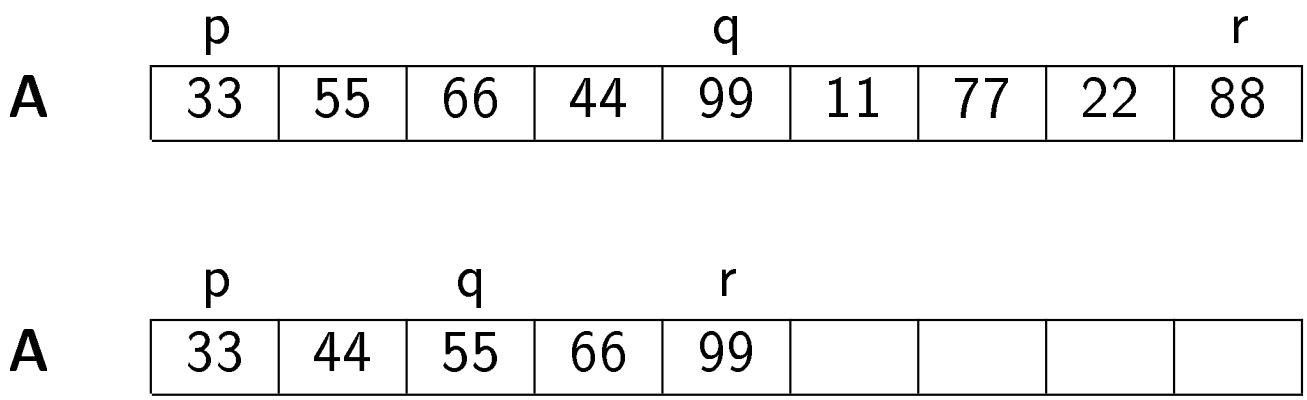




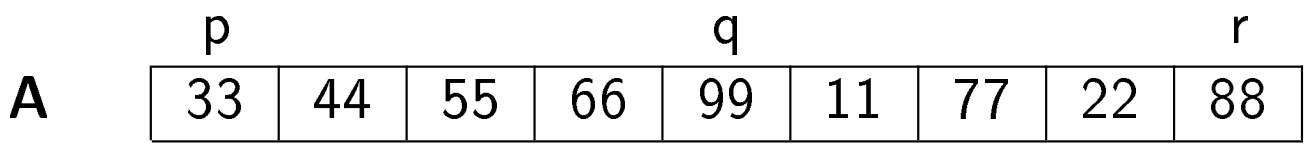
Ordenação por Intercalação (*Mergesort*)



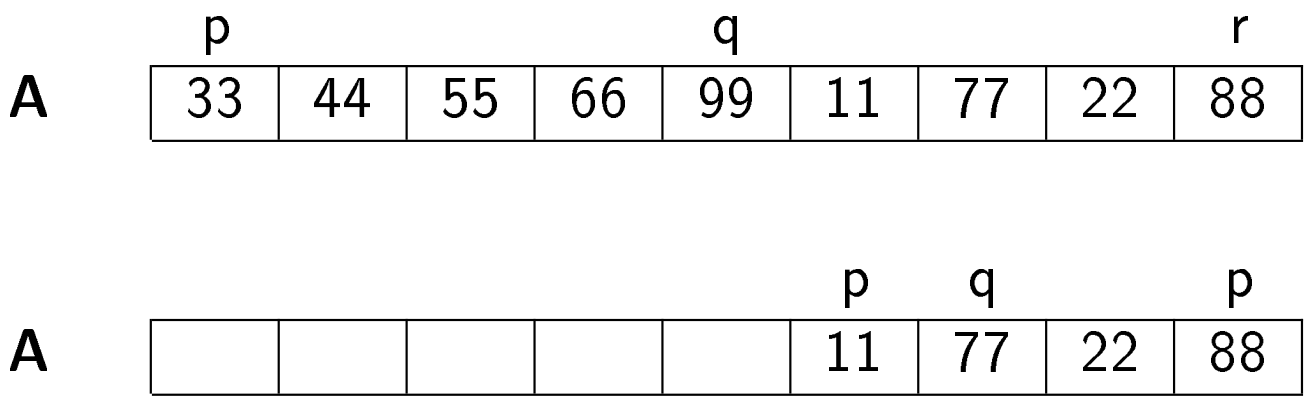
Ordenação por Intercalação (*Mergesort*)



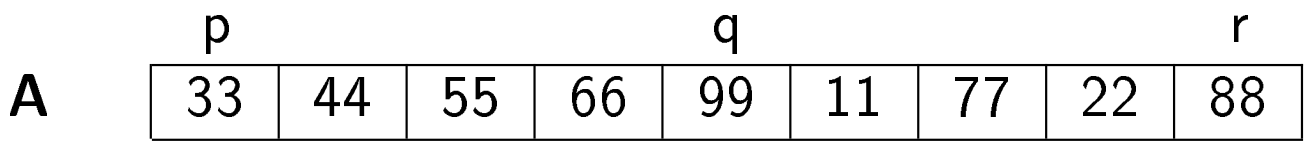
Ordenação por Intercalação (*Mergesort*)

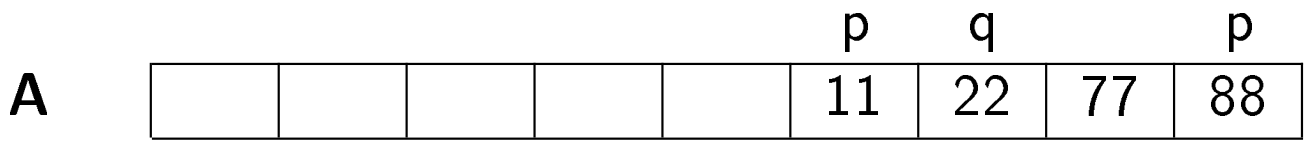


Ordenação por Intercalação (*Mergesort*)



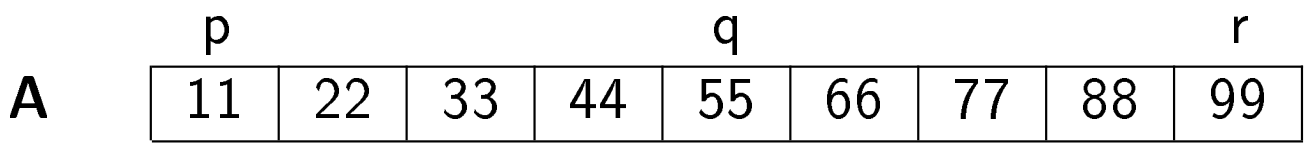
Ordenação por Intercalação (*Mergesort*)





...

Ordenação por Intercalação (*Mergesort*)



Ordenação por Intercalação (*Mergesort*)

- A complexidade do *mergesort* é $O(n \log n)$ no pior caso

Referências I


Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.
Introduction to Algorithms.
Third edition, The MIT Press, 2009.


Horowitz, E., Sahni, S. Rajasekaran, S.
Computer Algorithms.
Computer Science Press, 1998.


Rosa, J. L. G.
Métodos de Ordenação. SCE-181 – Introdução à Ciência da Computação II.
Slides. Ciência de Computação. ICMC/USP, 2018.


Ziviani, N.
Projeto de Algoritmos - com implementações em Java e C++.
Thomson, 2007.