Shell Sort

Prof. Dr. Eleandro Maschio
Tecnologia em Sistemas para Internet
Câmpus Guarapuava
Universidade Tecnológica Federal do Paraná (UTFPR)

Shell Sort

- Criado por **Donald Shell** em 1959 e publicado pela Universidade de Cincinnati.
- O funcionamento é frequentemente descrito como uma analogia às **conchas** (*shells*) do mar empilhadas umas sobre as outras.
- É considerado um refinamento da **ordenação por inserção**.

Shell Sort

Lógica de Implementação

- O método é baseado na **ordenação de subsequências** por inserção.
- As subsequências são formadas por elementos afastados ***h*** **posições** um do outro.
- O valor de ***h***, a distância em posições, é **decrementado** a cada iteração.

Shell Sort

Lógica de implementação

(**1**) A **distância inicial** entre os elementos que serão ordenados é 3, ou seja, **metade do tamanho**.

f	d	a	c	b	e
0	1	2	3	4	5

TAM = 6

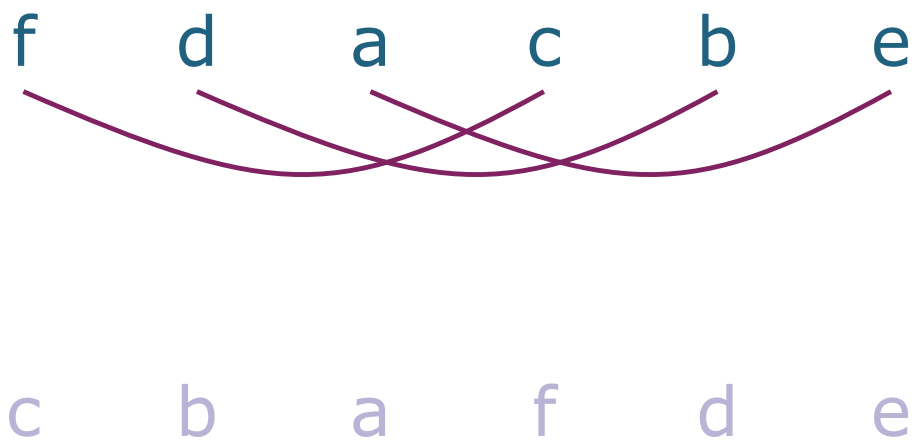
h = TAM/2 = 3

Shell Sort

Lógica de implementação

(**2**) Ordene por inserção todos os elementos que estão **3 posições afastados** um do outro.

h = 3

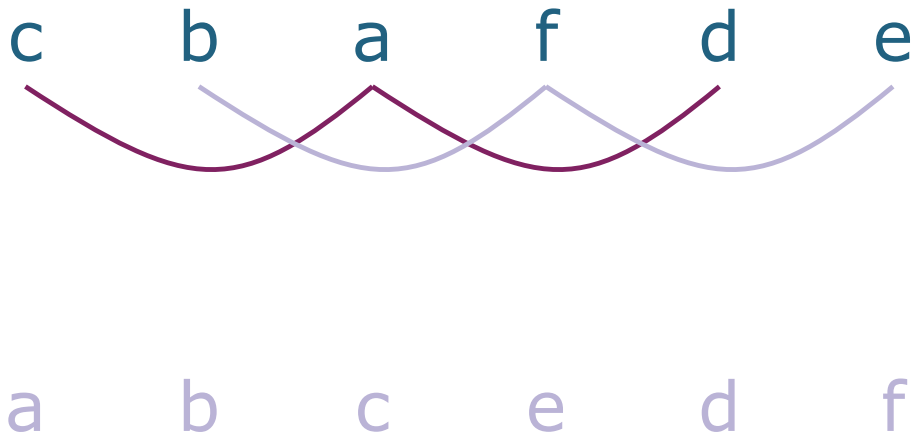


Shell Sort

Lógica de implementação

(**3**) Ordene por inserção todos os elementos que estão **2 posições afastados** um do outro.

h = 2

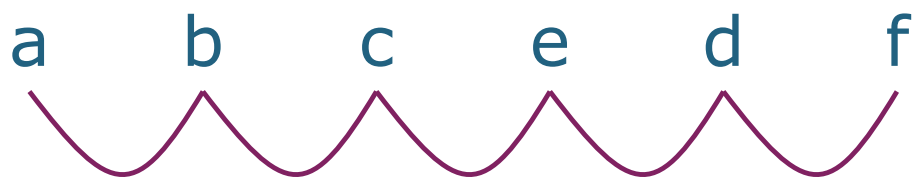


Shell Sort

Lógica de implementação

(**4**) Ordene por inserção todos os elementos que estão **1 posição afastados** um do outro.

h = 1



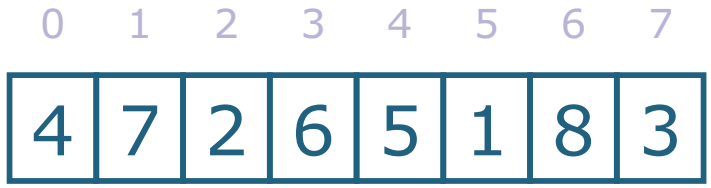
a b c d e f

Shell Sort

- **Não é trivial** perceber que este método conduz a bons resultados, ou mesmo que ordene a matriz.
- Entretanto, o *Shell Sort* executa bem ambas as tarefas.
- Cada passo da ordenação envolve relativamente **poucos elementos**, ou ainda elementos que já estão **razoavelmente em ordem**.
- Logo, o *Shell Sort* é eficiente e cada iteração aumenta a ordem dos dados.

Dinâmica de Funcionamento

Considere a matriz unidimensional.

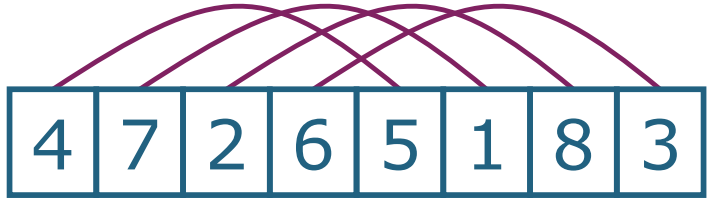


TAM = 8

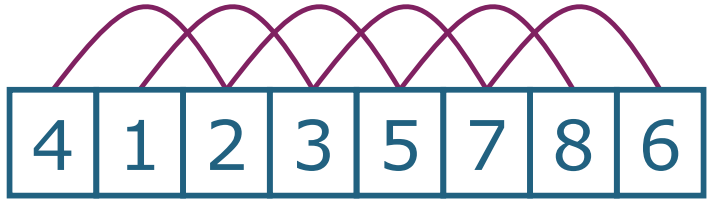
h = TAM/2 = 4

Dinâmica de Funcionamento

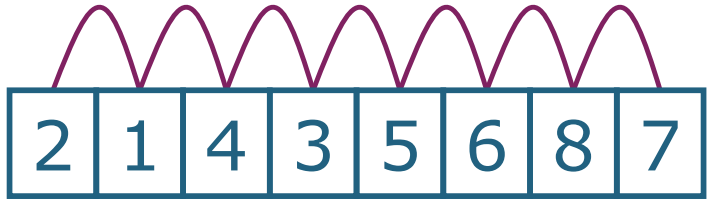
h = 4



h = 2

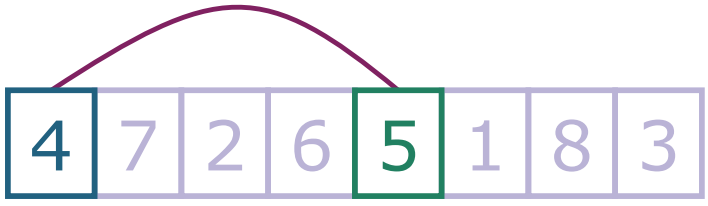


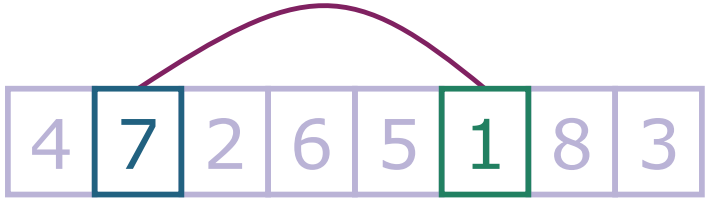
h = 1

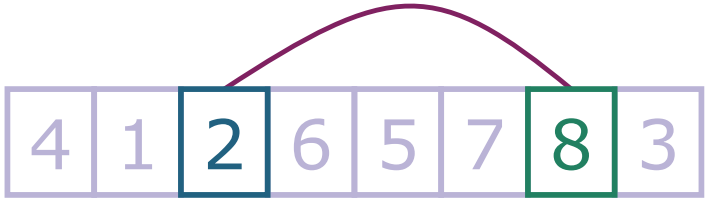


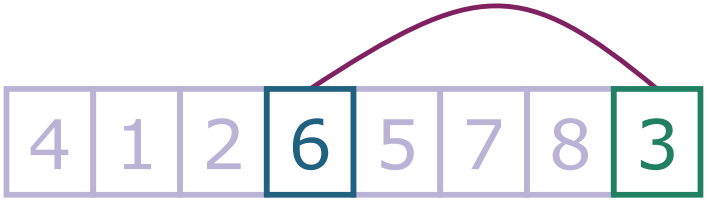
Dinâmica de Funcionamento

h = 4

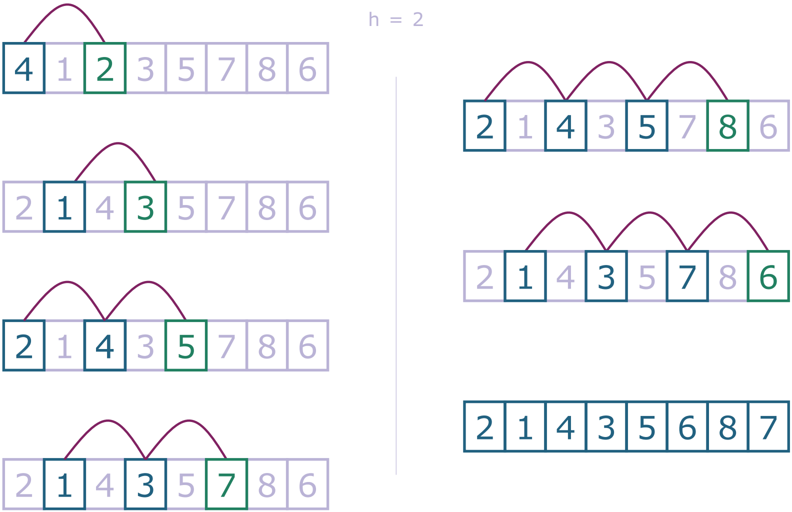






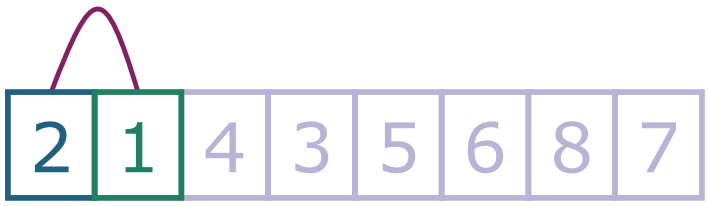


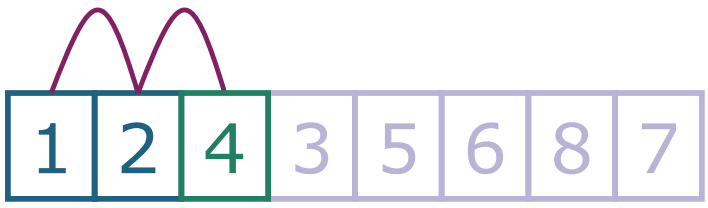
Dinâmica de Funcionamento

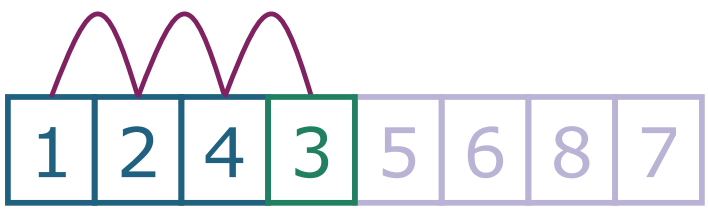


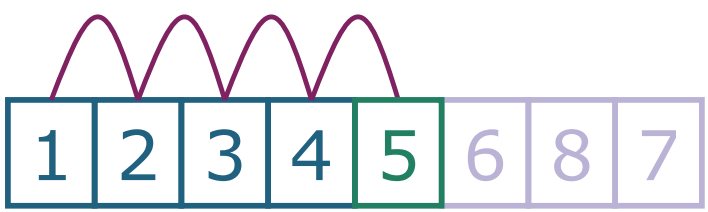
Dinâmica de Funcionamento

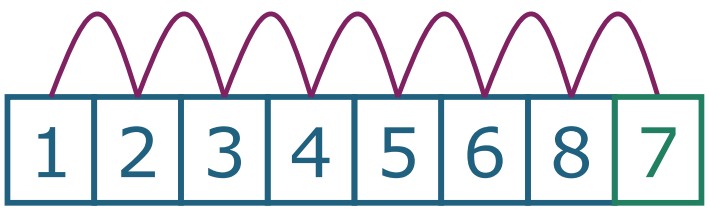
h = 1











Dinâmica de Funcionamento e Desempenho

(**1**) Considere que o *Shell Sort* é um método de ordenação por inserção.

(**2**) A complexidade computacional tanto do *Shell Sort* quanto do *Insertion Sort* são quadráticas.

(**3**) A última iteração do *Shell Sort*, ou seja, quando $h = 1$, corresponde à própria implementação completa do *Insertion Sort*.

(**4**) **Pergunta:** Como então o *Shell Sort* consegue ser mais rápido do que o *Insertion Sort*?

Implementação

- A **sequência** exata para os decrementos da distância entre os elementos pode variar conforme a implementação. A única regra é que a última distância **deve ser 1**.

- Uma possível sequência seria: 9, 5, 3, 2, 1.

- Em critérios de implementação, poderia ser armazenada em uma matriz unidimensional de distâncias:

```
int h[5] = {9, 5, 3, 2, 1};
```

- Embora bastante didática, sequências de distâncias que sejam **potências de 2** (por razões matemáticas complexas) reduzem a eficiência do *Shell Sort*. Ainda que assim o método funcione bem.

Implementação

- Em resumo, há **complexas** teorias matemáticas acerca da sequência de decremento desta distância.
- Pesquise pelo termo **Gap Sequence.**

  `http://en.wikipedia.org/wiki/Shellsort`

Desempenho

- Por iniciar com elementos distantes, o *Shell Sort* pode alocar elementos desordenados na posição correta mais rápido do que a comparação entre vizinhos próximos.
- Conforme aula posterior, em que serão comparados os métodos e respectivas complexidades, o *Shell Sort* é o mais eficiente algoritmo de ordenação dentre os de complexidade quadrática.

Shell Sort

```c
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define TAM 20

int main()
{
    srand(time(NULL));

    printf("Shell Sort\n");
    printf("==========\n\n\n");

    int dados[TAM], i = 0;
    printf("Sequencia Gerada\n");
    printf("----------------\n");
    for (i = 0; i < TAM; i++)
    {
        dados[i] = rand() % 100;
        printf("%2d ", dados[i]);
    }
```

Shell Sort

Implementação Sugerida com While

```
int h = TAM / 2;
int atual, j;
while (h > 0)
{
   // Aplicação do Insertion Sort na subsequência.
   for (i = h; i < TAM; i++)
   {
      atual = dados[i];
      j = i;
      while (j >= h && dados[j - h] > atual)
      {
          dados[j] = dados[j - h];
          j = j - h;
      }
      dados[j] = atual;
   }
   h = h / 2;
}
```

Shell Sort

Implementação Sugerida com For

```
int h = TAM / 2;
int atual, j;
while (h > 0)
{
   // Aplicação do Insertion Sort na subsequência.
   for (i = h; i < TAM; i++)
   {
      atual = dados[i];
      for (j = i; j >= h && dados[j-h] > atual; j = j - h)
      {
         dados[j] = dados[j-h];
      }
      dados[j] = atual;
   }
   h = h / 2;
}
```

Shell Sort

```
    printf("\n\n");
    printf("Sequencia Ordenada\n");
    printf("------------------\n");
    for (i = 0; i < TAM; i++)
    {
        printf("%2d ", dados[i]);
    }

    return(0);
}
```