

Algoritmul ShellSort

- ◆ ShellSort este un algoritm de sortare performant, bazat pe sortarea prin insertie (InsertSort), fiind supranumit si "insertie cu diminuarea incrementului"
- ◆ Algoritmul lucreaza pe tablouri de lungime N , fiind de clasa $O(N^2)$, clasa ce caracterizeaza, in mod normal, algoritmii de sortare mai putin performanti
- ◆ Cu toate acestea, algoritmul este vizibil mai rapid decat algoritmii obisnuiti din clasa $O(N^2)$: InsertSort, BubbleSort, ShakerSort, SelSort, etc., fiind de circa 2 ori mai rapid decat InsertSort, cel mai apropiat competitor din clasa $O(N^2)$
- ◆ ShellSort nu este un algoritm de sortare "in situ", adica necesita structuri de date suplimentare, in plus fata de spatiul de memorie al tabloului ce trebuie sortat – spatiul suplimentar este revendicat de un tablou de incrementi necesar rularii algoritmului

Algoritmul ShellSort

◆ Presupunem ca dorim sa sortam urmatorul tablou:

Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A:	17	8	3	21	14	24	2	12	30	9	4	19	6	18	23	15	7	13	1

◆ Algoritmul Shellsort propune alegerea in prealabil a unui tablou H, numit "tablou de incrementi":

◆ Tabloul de incrementi trebuie sa indeplineasca conditiile:

- H are M elemente (0 ... M-1) cu $M > 1$
- $H[M-1] = 1$ (ultimul element trebuie sa fie 1)
- $H[i] > H[i+1]$ pentru $i = 0 \dots M-2$ (H trebuie sa fie un tablou strict descrescator)

◆ De exemplu, $H = [7, 4, 3, 2, 1]$

Algoritmul ShellSort

- ◆ Algoritmul va face M treceri asupra tabloului A , unde M este lungimea tabloului de incrementi
- ◆ Dupa fiecare pas i , elementele aflate in tabloul A la distanta $H[i]$ unul de altul vor fi sortate
- ◆ Sortarea acestor elemente se va face folosind algoritmul InsertSort
- ◆ InsertSort este cel mai performant dintre algoritmii de sortare neperformanti

Algoritmul ShellSort

Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A:	17	8	3	21	14	24	2	12	30	9	4	19	6	18	23	15	7	13	1


- ◆ La pasul 0, vom sorta folosind InsertSort elementele aflate la distanta $H[0]=7$ unul de celalalt
- ◆ Acestea sunt:
 - 17, 12, 23
 - 8, 30, 15
 - 3, 9, 7
 - s. a. m. d.
- ◆ Ele ocupa celule colorate la fel in reprezentarea de mai sus

Algoritmul ShellSort

Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A:	17	8	3	21	14	24	2	12	30	9	4	19	6	18	23	15	7	13	1

- ◆ Pentru usurinta intelegerii, vom copia elementele tabloului intr-o matrice avand 7 coloane si vom sorta pe coloane (sortarea coloanelor foloseste tehnica InsertSort):

17	8	3	21	14	24	2
12	30	9	4	19	6	18
23	15	7	13	1		



12	8	3	4	1	6	2
17	15	7	13	14	24	18
23	30	9	21	19		

- ◆ Mai apoi, vom reface tabloul initial din liniile matricii, observand ca daca luam elementele sale din 7 in 7, obtinem siruri sortate

Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A:	12	8	3	4	1	6	2	17	15	7	13	14	24	18	23	30	9	21	19

Algoritmul ShellSort

Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A:	12	8	3	4	1	6	2	17	15	7	13	14	24	18	23	30	9	21	19


- ◆ La pasul 1, vom sorta folosind InsertSort elementele aflate la distanta $H[1]=4$ unul de celalalt
- ◆ Acestea sunt:
 - 12, 1, 15, 24, 9
 - 8, 6, 7, 18, 21
 - 3, 2, 13, 23, 19
 - 4, 17, 14, 30
- ◆ Ele ocupa celule colorate la fel in reprezentarea de mai sus

Algoritmul ShellSort

Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A:	12	8	3	4	1	6	2	17	15	7	13	14	24	18	23	30	9	21	19

- ◆ Vom copia elementele tabloului intr-o matrice avand 4 coloane si vom sorta pe coloane (sortarea coloanelor foloseste tehnica InsertSort):

12	8	3	4
1	6	2	17
15	7	13	14
24	18	23	30
9	21	19	

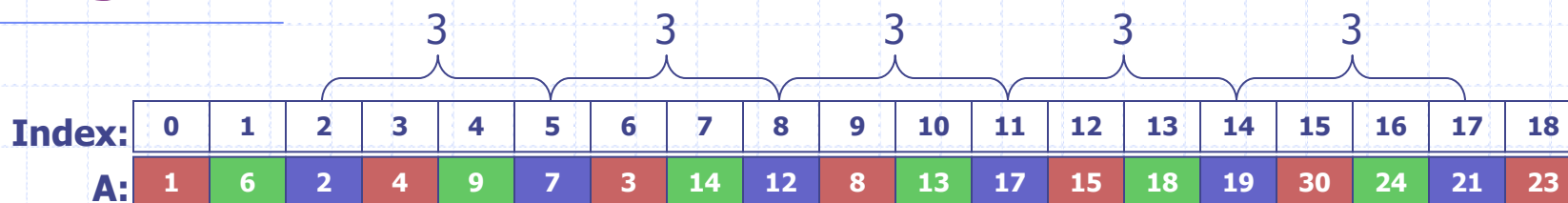


1	6	2	4
9	7	3	14
12	8	13	17
15	18	19	30
24	21	23	

- ◆ Apoi vom reface tabloul:

Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A:	1	6	2	4	9	7	3	14	12	8	13	17	15	18	19	30	24	21	23

Algoritmul ShellSort



Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A:	1	6	2	4	9	7	3	14	12	8	13	17	15	18	19	30	24	21	23


- ◆ La pasul 2, vom sorta folosind InsertSort elementele aflate la distanta $H[2]=3$ unul de celalalt
- ◆ Acestea sunt:
 - 1, 4, 3, 8, 15, 30, 23
 - 6, 9, 14, 13, 18, 24
 - 2, 7, 12, 17, 19, 21
- ◆ Ele ocupa celule colorate la fel in reprezentarea de mai sus

Algoritmul ShellSort

Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A:	1	6	2	4	9	7	3	14	12	8	13	17	15	18	19	30	24	21	23

- ◆ Vom copia elementele tabloului intr-o matrice avand 3 coloane si vom sorta pe coloane (sortarea coloanelor foloseste tehnica InsertSort):

1	6	2
4	9	7
3	14	12
8	13	17
15	18	19
30	24	21
23		



1	6	2
3	9	7
4	13	12
8	14	17
15	18	19
23	24	21
30		

- ◆ Apoi vom reface tabloul:

Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A:	1	6	2	3	9	7	4	13	12	8	14	17	15	18	19	23	24	21	30

Algoritmul ShellSort

Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A:	1	6	2	3	9	7	4	13	12	8	14	17	15	18	19	23	24	21	30

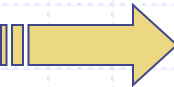
- ◆ La pasul 3, vom sorta folosind InsertSort elementele aflate la distanta $H[3]=2$ unul de celalalt
- ◆ Acestea sunt:
 - 1, 2, 9, 4, 12, 14, 15, 19, 24, 30
 - 6, 3, 7, 13, 8, 17, 18, 23, 21
- ◆ Ele ocupa celule colorate la fel in reprezentarea de mai sus

Algoritmul ShellSort

Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A:	1	6	2	3	9	7	4	13	12	8	14	17	15	18	19	23	24	21	30

- ◆ Vom copia elementele tabloului intr-o matrice avand 2 coloane si vom sorta pe coloane (sortarea coloanelor foloseste tehnica InsertSort):

1	6
2	3
9	7
4	13
12	8
14	17
15	18
19	23
24	21
30	



1	3
2	6
4	7
9	8
12	13
14	17
15	18
19	21
24	23
30	

Algoritmul ShellSort

Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A:	1	3	2	6	4	7	9	8	12	13	14	17	15	18	19	21	24	23	30

- ◆ La pasul 4, vom sorta folosind InsertSort elementele aflate la distanta $H[4]=1$ unul de celalalt
- ◆ Cu alte cuvinte, vom aplica un InsertSort obisnuit pe intreg tabloul A
- ◆ Faptul ca ultimul element al lui H este 1 garanteaza ca tabloul A sfarseste prin a fi sortat
- ◆ Se observa ca pasii anteriori au adus tabloul A la o forma aproape ordonata, deci ultimul InsertSort va reusi sa sorteze tabloul foarte rapid, chiar daca este o metoda putin performanta, fiind de clasa $O(N^2)$

Algoritmul ShellSort

Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A:	1	2	3	4	6	7	8	9	12	13	14	15	17	18	19	21	23	24	30

- ◆ Mai sus este prezentat rezultatul sortarii
- ◆ Performanta algoritmului ShellSort depinde decisiv de alegerea tabloului de incrementi H
- ◆ De exemplu:
 - pentru tabloul de incrementi 1 3 7 15 31 63 127 255 ... (luat invers, evident), s-a demonstrat ca algoritmul este de clasa $O(N^{3/2})$
 - pentru tabloul de incrementi 1 8 23 77 281 1073 4193 ... avand termenul general de forma $4^{j+1} + 3 \cdot 2^j + 1$, s-a demonstrat ca algoritmul este de clasa $O(N^{4/3})$
 - s-a demonstrat ca exista tablouri de incrementi pentru care algoritmul este de clasa $O(N^{1+1/k})$, folosind $O(\log_2 N)$ incrementi
 - alte rezultate sunt disponibile in literatura

Algoritmul ShellSort

◆ Codul C al algoritmului este dat mai jos:

```
void shellsort(int a[], int l, int r) {  
    int i, j, h, v;  
    int increments[16] = {1391376, 463792, 198768, 86961, 33936, 13776, 4592,  
                           1968, 861, 336, 112, 48, 21, 7, 3, 1};  
  
    for (k = 0; k < 16; k++)  
        for (h = increments[k], i = l+h; i <= r; i++) {  
            v = a[i]; j = i;  
            while (j > h && a[j-h] > v) {  
                a[j] = a[j-h]; j = j-h;  
            }  
            a[j] = v;  
        }  
}
```

Algoritmul ShellSort

- ◆ Nu se recomanda alegerea puterilor lui 2 pe post de incrementi, deoarece aceasta ar insemna ca elementele de la indici pari nu vor fi sortate cu elementele de la indici impari decat in cadrul ultimei treceri
- ◆ Algoritmul ShellSort este cel mai rapid algoritm de clasa $O(N^2)$
- ◆ Totusi, nu se poate compara cu algoritmii de sortare super-performanti (QuickSort sau HeapSort)