Note

An improved Shellsort algorithm

Renren Liu

Department of Computer Science, Xiangtan University, Xiangtan, Hunan, 411105, China

Received October 1996
Communicated by M. Nivat

Abstract

Shellsort algorithm is a refinement of the straight insertion sort. Each pass of this method sorts all items which are \( h \)-position apart by straight insertion sort, and the place of the item to be inserted is determined by comparing items which are already ordered from right to left.

In this paper, straight insertion of \( h \)-step length is improved to binary insertion of \( h \)-step length. So the number of comparison is reduced from \( O(n^2) \) to \( O(n \log_2 n) \).

Keywords: Algorithm; Sorting

1. Introduction

Sorting is done very frequently in data processing. The simplest sorting methods are selection sort, bubble sort and insertion sorting, these methods usually take \( O(n^2) \). Shellsort algorithm [1], sometimes called diminishing-increment sort, is a refinement of the straight insertion sorting. In each pass of this method, all items which are \( h \)-position apart are sorted by straight insertion sorting, hereafter called \( h \)-step sorting, where \( h \) decreases progressively. Finally, one-step sorting will be carried out for all elements. In the \( h \)-step sorting, for every element \( x \) beginning from position \( h + 1 \), it is, respectively, compared with its left elements which are \( h \)-position apart. If \( x < a[i] \) then \( a[i] \) would be moved right into \( h \) positions, etc. Finally, \( x \) will be inserted into its rightful place. Obviously, the elements which be compared with \( x \) in \( h \)-step sorting are in sorted order, called \( h \)-ordered. Thus, we may insert \( x \) into rightful place by binary search method so that elements are \( h \)-ordered after \( h \)-step sorting. So, the number of comparisons between elements will be \( O(n \log_2 n) \) on the average case.
2. Original Shellsort algorithm

Each \(h\)-step sorting of Shellsort can be described as follows:

Procedure \(h\)-sort\((n, h)\);
\[
\begin{align*}
&\text{Var } x, i, j: \text{ integer}; \\
&\quad h: \text{ boolean}; \\
&\text{Begin} \\
&\quad \text{for } i := h + 1 \text{ to } n \text{ do} \\
&\quad \quad \text{begin} \\
&\quad \quad \quad x := a[i]; \quad j := i - h; \quad b := \text{true}; \\
&\quad \quad \quad \text{while } (b = \text{true}) \text{ and } (a[j] > x) \text{ do} \\
&\quad \quad \quad \quad \text{begin} \\
&\quad \quad \quad \quad \quad a[j + h] := a[j]; \quad j := j - h; \\
&\quad \quad \quad \quad \quad \text{if } j \leq 0 \text{ then } b := \text{false} \\
&\quad \quad \quad \quad \text{end}; \\
&\quad \quad \quad a[j + h] := x; \\
&\quad \quad \text{end} \\
&\text{End; \{h-sort\}}
\end{align*}
\]

Whenever the procedure given above is carried out one time, the elements which be \(h\)-position apart are \(h\)-ordered. Take a descending chain \(h_1, h_2, \ldots, h_t\) such that \(h_t = 1\) and \(h_{t+1} < h_t\). After procedure \(h\)-sort\((n, h)\) is called \(t\) times, \(n\) elements will be in sorted order. Although now we do not know what is the best descending chain, some parts of conclusion have been obtained. For example, Knuth [2] claimed that if \(h_{i-1} = 3h_i + 1\), \(h_i = 1, \ t = \lceil \log_3 n \rceil - 1\), then the following sequence

\[\ldots, 121, 41, 13, 4, 1\]  \hspace{1cm} (1)

is reasonable. Hereafter we will take sequence (1) in the following algorithm.

The Shellsort algorithm can be described as follows:

Procedure Shellsort\((a: \text{ array } [1..n] \text{ of integer})\);
\[
\begin{align*}
&\text{Var } i, h: \text{ integer}; \\
&\text{Begin} \\
&\quad h := (3^{|\log_3 n|} - 1) \text{ div } 2; \\
&\quad \text{while } h > 0 \text{ do} \\
&\quad \quad \text{begin} \\
&\quad \quad \quad h\text{-sort}(n, h); \\
&\quad \quad \quad h := (h - 1) \text{ div } 3 \\
&\quad \quad \text{end} \\
&\text{End; \{shellsort\}}
\end{align*}
\]
3. The improved Shellsort algorithm

As mentioned in the introduction, in order to determine the insertion position of element $x$ in $h$-step sorting by binary research method, we first need to calculate the left and right positions of elements being checked by binary searching, denoted by $l$ and $r$, respectively. Obviously, $r = i - h$, where $i$ is the position of $x$, and $l$ must satisfies $1 \leq l \leq h$, also, $l = h$ if and only if $h \mid r$. So $l$ can be determined by formula as follows:

$$l = \begin{cases} h & h \mid r, \\ r - h \times (r \div h) & \text{otherwise.} \end{cases}$$

(2)

Second, we need to find the number of elements from $l$ to $r$ which are $h$-position apart, denoted by $s$. Clearly, $h \mid (r - l)$ whatever $l$ is in (2), and $(r - l) \div h$ show the number of intervals which have length of $h$. Thus, we have

$$s = (r - l) \div h + 1.$$

Finally, we need to determine the position of middle element among $s$ elements, denoted by $m$, we have

$$m = l + (s \div 2) \times h.$$

During the circulation of determining the position of element $x$, the values of $l$ and $r$ change alternately until $l > r$, at this time, $l$ is the position of $x$ to insert. Before inserting $x$, those elements $h$-position apart from address $l$ to $i - h$ will be moved $h$ units right one by one.

The improved Shellsort program is as follows:

Program Shellsort (input, output);

Const n = 100;  {the number of elements to be sorted}

Var a: array[-12..n] of integer;

$m, i, j, s, h$: integer;

procedure binsert (n, h: integer);

{binary inserting sort with $h$-step}

var x, i, j, l, r, s, m: integer;

begin

for $i := h + 1$ to $n$ do

begin

$x := a[i]$; $j := i - h$; $r := j$; $l := r - h \times (r \div h)$;

if $l = 0$ then $l := h$;

while $l \leq r$ do

begin

$s := (r - l) \div h + 1$;

$m := l + (s \div 2) \times h$;

if $x < a[m]$ then $r := m - h$ else $l := m + h$;

end

end

end

end
end;
while $j > 1$ do
begin
\[ a[j + h] := a[j]; \]
\[ j := j - h \]
end;
\[ a[Z] := x \]
end; \{ binsert \}

Begin
Randomize; \{ produce at random $n$ elements to be sorted \}
for $i := 1$ to $m$ do $h := h \times 3$;
for $i := 1$ to $h$ do $a[i] := 0$; \{ initial to $h_1$ expanding elements of $a$ \}
while $h > 0$ do
begin
\[ binsert(n, h); \]
\[ h := (h - 1) \text{div} 3 \]
end
End. \{ Shellsort \}

4. Analysis and comparison

We can compare improved Shellsort algorithm with original Shellsort on the number of comparison between $x$ and $a[j]$, $a[m]$, respectively.

4.1. The number of comparison of algorithm $h$-sort

4.1.1. The minimal number of comparison $C_{min}^h$

When $x$ is greater than the element left $h$-position apart, comparison stops, and we have

\[ C_{min}^h = \sum_{i=h+1}^{n} 1 = n - h. \]

4.1.2. The maximal number of comparison $C_{max}^h$

Suppose there are $S_i$ elements in the $i$th pass $h$-step sorting, we obtain from (2)

\[ S_i = (r - l) \text{div} h + 1 \]
\[ f_i = \begin{cases} \frac{(r - h)}{h + 1}, & h \mid r, \\ \frac{(r - h \times (r \div h)))}{h + 1} & \text{otherwise}, \end{cases} \]

\[ f_i = \begin{cases} \frac{(r - h)}{h + 1}, & h \mid r, \\ \frac{r \div h + 1}{h + 1} & \text{otherwise}. \end{cases} \tag{3} \]

(a) When \( h \mid r \), then \( r = i - h \geq h \), i.e., \( i \geq 2h \). Thus,

\[ S_i = (r - 1) \div h + 1 = (i - 2h) \div h + 1. \]

Hence, we have

\[ C_{\text{max}}^h = \frac{1}{h} \sum_{i=2h}^{n} (i - 2h) + \sum_{i=2h}^{n} 1 = \frac{(n - 2h)(n - 2h + 1)}{2h} + (n - 2h + 1) \]

\[ = \frac{n(n - 2h + 1)}{2h}. \]

Average comparing number \( C_{\text{ave}}^h \) is as follows:

\[ C_{\text{ave}}^h = \frac{1}{2} \left( C_{\text{min}}^h + C_{\text{max}}^h \right) = \frac{1}{2} \left[ (n - h) + \frac{n(n - 2h + 1)}{2h} \right] \]

\[ = \frac{1}{4h} (n^2 + n - 2h^2) = O(n^2). \]

(b) When \( h \mid r \) is not true, we have

\[ C_{\text{max}}^h = \sum_{i=h+1}^{n} S_i = \frac{1}{h} \sum_{i=h+1}^{n} (i - h) + \sum_{i=h+1}^{n} 1 = \frac{1}{h} \left[ 1 + 2 + \cdots + (n - h) \right] + (n - h) \]

\[ = \frac{(n - h)(n - h + 1)}{h} + (n - h). \]

Average comparing number is as follows:

\[ C_{\text{ave}}^h = \frac{1}{2} \left( C_{\text{min}}^h + C_{\text{max}}^h \right) \]

\[ = \frac{1}{2} \left[ \frac{(n - h)(n - h + 1)}{h} + 2(n - h) \right] \]

\[ = \frac{(n - h)(n - h + 3)}{2h} = O(n^2). \]

In a word, we obtain \( C_{\text{ave}}^h = O(n^2). \)

4.2. The number of comparison of improved algorithm binser

Clearly, when comparison is done one time, the number of elements to be compared decrease by a half. So, we have

\[ C_{\text{min}}^h = \sum_{i=h+1}^{n} \lceil \log_2 S_i \rceil, \tag{4} \]
\[ C_{\text{max}}^h = \sum_{i=1}^{n} ([\log_2 S_i] + 1), \]

\[ C_{\text{ave}}^h = \frac{1}{2} (C_{\text{min}}^h + C_{\text{max}}^h), \]

where \( S_i \) is defined by (3).

We can approximate to (4) and (5) by means of integration as follows:

\[
\int_{h+1}^{n} \log x \, dx = n(\log n - C) - (h + 1)(\log(h + 1) - C),
\]

where \( C = \log e = 1/\ln 2 = 1.44269 \ldots \).

References


\( n = 154 \)

\begin{align*}
807, & 798, 265, 274, 323, 123, 826, 650, 627, 606, 175, 760, 391, 448, \\
351, & 651, 873, 856, 656, 960, 638, 288, 346, 549, 667, 720, 404, 561, \\
550, & 293, 191, 864, 679, 938, 28, 137, 597, 849, 996, 75, 953, 913, \\
513, & 439, 707, 448, 468, 951, 530, 866, 986, 255, 553, 122, 85, 436, \\
898, & 49, 805, 830, 657, 947, 588, 103, 244, 29, 953, 848, 295, 595, \\
783, & 166, 524, 25, 746, 978, 227, 965, 661, 54, 341, 315, 857, 979, \\
232, & 865, 816, 555, 643, 705, 648, 370, 458, 758, 195, 512, 225, 52, \\
817, & 667, 584, 578, 146, 90, 50, 862, 359, 173, 795, 596, 781, 601, \\
378, & 942, 835, 336, 569, 364, 662, 893, 710, 115, 954, 671, 275, 705, \\
781, & 299, 544, 2, 476, 242, 587, 779, 869, 155, 944, 862, 415, 706, \\
710, & 202, 12, 872, 918, 499, 563, 920, 412, 236, 573, 578, 616, 225,
\end{align*}

\( h = 13 \)

\begin{align*}
29; & 54; 85; 173; 115; 49; 52; 275; 25; 137; 175; 103; 2; 50; \\
122; & 265; 195; 191; 123; 166; 468; 28; 346; 225; 146; 12; 75; 242; \\
341; & 274; 323; 155; 232; 524; 288; 530; 299; 202; 90; 364; 351; 359; \\
293; & 439; 225; 236; 573; 378; 555; 549; 227; 244; 29; 448; 662; 587; 436; 795; 656; \\
512; & 412; 448; 601; 415; 584; 578; 336; 255; 404; 561; 550; 315; 595; \\
596; & 671; 638; 578; 606; 588; 544; 391; 448; 662; 587; 436; 795; 656; \\
679; & 650; 627; 616; 597; 667; 569; 476; 798; 651; 513; 856; 707; 781; \\
817; & 657; 706; 643; 705; 648; 553; 862; 758; 563; 857; 783; 805; 830; \\
667; & 746; 710; 760; 720; 661; 918; 848; 710; 869; 864; 826; 862; 705; \\
781; & 835; 849; 965; 807; 953; 893; 779; 898; 954; 944; 865; 816; 942; \\
866; & 986; 996; 872; 953; 913; 873; 920; 979; 960; 938; 951; 947; 978;
\end{align*}
$h = 4$

| 2; 12; 52; 103; 25; 28; 75; 155; 29; 49; 85; 166; 115; 50; 90; 173; 146; 54; 122; 225; 195; 137; 123; 242; 225; 191; 175; 244; 232; 202; 227; 265; 255; 236; 288; 275; 299; 274; 293; 295; 315; 359; 323; 336; 341; 391; 346; 364; 351; 404; 448; 378; 370; 412; 448; 439; 415; 436; 499; 476; 468; 458; 513; 530; 512; 524; 561; 550; 554; 544; 549; 569; 563; 553; 578; 573; 588; 555; 584; 578; 601; 587; 595; 596; 616; 597; 643; 606; 648; 638; 650; 627; 656; 667; 651; 705; 657; 679; 661; 710; 662; 706; 667; 758; 671; 707; 705; 781; 760; 710; 746; 795; 826; 720; 779; 805; 830; 798; 781; 807; 835; 849; 783; 816; 848; 857; 862; 817; 856; 862; 865; 864; 872; 866; 869; 873; 920; 893; 913; 898; 942; 944; 960; 918; 951; 947; 965; 938; 953; 953; 978; 996; 954; 979; 986; |

$h = 1$

| 2; 12; 25; 28; 29; 49; 50; 52; 54; 75; 85; 90; 103; 115; 122; 123; 137; 146; 155; 166; 173; 175; 191; 195; 202; 225; 225; 227; 232; 236; 242; 244; 255; 265; 274; 275; 288; 293; 295; 299; 315; 323; 336; 341; 351; 359; 364; 370; 378; 391; 404; 412; 415; 436; 439; 448; 448; 458; 468; 476; 499; 512; 513; 524; 530; 544; 550; 553; 555; 561; 569; 573; 578; 578; 584; 584; 587; 588; 595; 596; 597; 601; 606; 616; 627; 638; 643; 648; 650; 651; 656; 657; 661; 662; 667; 667; 671; 679; 705; 705; 706; 707; 710; 710; 720; 746; 758; 760; 779; 781; 781; 783; 798; 805; 807; 816; 817; 826; 830; 835; 848; 849; 857; 858; 862; 862; 865; 864; 866; 869; 872; 873; 893; 898; 913; 918; 920; 938; 942; 944; 947; 951; 953; 953; 954; 960; 965; 978; 979; 986; 996; |