

Shell sort algorithm

Section: 6c4

Student names	Academic numbers	Email addresses
Mjd alamri	443007585	443007585@pnu.edu.sa

Abstract

The essay introduces the Shell Sort algorithm, which is useful for sorting large arrays by allowing exchanges of far items. It sorts far-away elements first and then reduces the gap between them. Shell Sort has advantages such as improving time complexity by exchanging far elements, but it can be slow and complex. The time complexity varies depending on the list's order. The essay includes an example of Shell Sort implementation in Java and references to further resources.

Introduction

Shell sort is an algorithm similar to insertion sort but in this algorithm, it will allow the exchange of far items, and this makes it better for large arrays.[1]

It is an in-place comparison-based sorting algorithm.

This algorithm first sorts the elements that are far away from each other, then it subsequently reduces the gap between them. This gap is called interval.

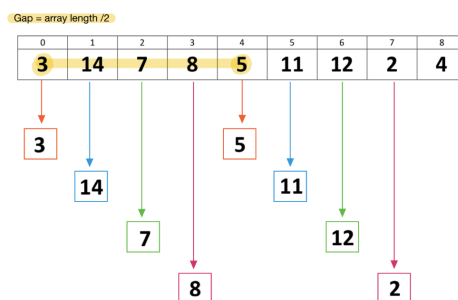
Advantages of shell sort:

- Shell Sort can improve its time complexity because it exchanges between far elements and that will result in a smaller number of moves.
- is only efficient for a finite number of elements in an array. [2]

Disadvantages:

- It is complex in structure and a bit more difficult to understand.
- it is a slow option. [3]

This is a picture showing how shell sort reorders the array.



An interesting fact to share, Donald Shell published the first version of this sort in 1959; That's why it is called shell sort. [4]

An example of the shell sort
with input: {3,14,7,8,5,11,12,2,4}

0	1	2	3	4	5	6	7	8
3	14	7	8	5	11	12	2	4

Gap = $n/2 = 9/2 = 4.5 = 4$

0	1	2	3	4	5	6	7	8
3	14	7	8	5	11	12	2	4

No swap

0	1	2	3	4	5	6	7	8
3	14	7	8	5	11	12	2	4

14 < 11 swap

0	1	2	3	4	5	6	7	8
3	11	7	8	5	14	12	2	4

0	1	2	3	4	5	6	7	8
3	11	7	8	5	14	12	2	4

0	1	2	3	4	5	6	7	8
3	11	7	2	4	14	12	8	5

Gap = $4/2 = 2$

0	1	2	3	4	5	6	7	8
3	11	7	2	4	14	12	8	5

0	1	2	3	4	5	6	7	8
3	2	7	11	4	14	12	8	5

0	1	2	3	4	5	6	7	8
3	2	7	11	4	14	12	8	5

0	1	2	3	4	5	6	7	8
3	2	4	11	7	14	12	8	5

0	1	2	3	4	5	6	7	8
3	2	4	11	7	14	12	8	5

0	1	2	3	4	5	6	7	8
3	2	4	11	7	14	12	8	5

0	1	2	3	4	5	6	7	8
3	2	4	11	7	14	12	8	5

0	1	2	3	4	5	6	7	8
3	2	4	11	7	8	12	14	5

0	1	2	3	4	5	6	7	8
3	2	4	11	7	8	12	14	5

0	1	2	3	4	5	6	7	8
3	2	4	11	7	8	5	14	12

0	1	2	3	4	5	6	7	8
3	2	4	11	7	8	5	14	12

Gap= 2/2=1

0	1	2	3	4	5	6	7	8
3	2	4	11	7	8	5	14	12

0	1	2	3	4	5	6	7	8
2	3	4	11	7	8	5	14	12

0	1	2	3	4	5	6	7	8
2	3	4	11	7	8	5	14	12

0	1	2	3	4	5	6	7	8
2	3	4	11	7	8	5	14	12

0	1	2	3	4	5	6	7	8
2	3	4	11	7	8	5	14	12

0	1	2	3	4	5	6	7	8
2	3	4	7	11	8	5	14	12

0	1	2	3	4	5	6	7	8
2	3	4	7	8	11	5	14	12

0	1	2	3	4	5	6	7	8
2	3	4	7	8	5	11	14	12

0	1	2	3	4	5	6	7	8
2	3	4	7	8	5	11	14	12

0	1	2	3	4	5	6	7	8
2	3	4	7	8	5	11	12	14

Sort again until all sorted.

0	1	2	3	4	5	6	7	8
2	3	4	5	7	8	11	12	14

Analysis

Pseudo code:

```
gap = size / 2
do until gap <= 0
  swapflag = true
  do until swapflag is false
    swapflag = false
    for s = 0 to size - gap
      if num[s] > num[s + gap]
        swap num[s] with num[s + gap]
        swapflag = true
      end-if
    end-for
  end-do
  gap = gap / 2
end-do
```

[5]

Space complexity is $O(1)$ and it's not a stable algorithm.

Time complexity after tracing the pseudo code:

code	time
int n = arr.length;	$O(1)$
int gap;	$O(1)$
for (gap = n / 2; gap > 0; gap /= 2) {	$O(\log n)$
for (int i = gap; i < n; i++) {	$O(n)$
int temp = arr[i];	$O(1)$
int j = i; while(j >= gap && arr[j-gap] > temp){	$O(n/\text{gap})$
arr[j] = arr[j - gap];	$O(1)$
j -= gap; }	$O(1)$
arr[j] = temp; }	$O(1)$

Time complexity:

$$\begin{aligned}
 &O(1) + O(1) + O(\log n) + (O(n) + (O(1) + O(1) + O(n/\text{gap})) + O(1) + O(1)) + O(1) \\
 &O(1) + O(1) + O(\log n) + (O(n) + O(n) + O(n * n/\text{gap}) + O(n * n/\text{gap}) + O(n)) \\
 &O(1) + O(1) + O(n \log n) + O(n \log n) + O(n \log n * n/\text{gap}) + O(n \log n * n/\text{gap}) + O(n \log n) \\
 &O(n \log n * n/\text{gap}) \\
 &= O(n \log n^2)
 \end{aligned}$$

Time complexity is different for each case:

Best case:

We have the best case when we have a sorted list the doesn't need to be sorted.

Then the time complexity will be $O(n \cdot \log n)$.

Average case:

We have the average case when the list is not sorted, but it is not sorted reversely, the

Complexity time will be $O(n \cdot \log(n^2))$.

Worst case:

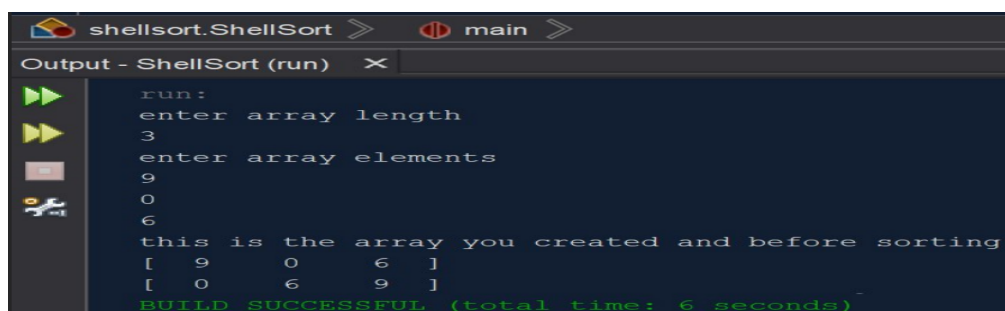
We say that the algorithm is in the worst case when the list is sorted reversely, ex: we need it sorted in ascending order but it is in descending order, time complexity will be $O(n^2)$. [6]

Implementation

The shell sort is implemented using java language :

```
3 import java.util.Scanner;
4
5 public class ShellSort {
6
7     public static void main(String[] args) {
8         Scanner in = new Scanner(System.in);
9
10        int[] arr;
11
12        //creating array from user
13        System.out.println("enter array length");
14        int n = in.nextInt();
15        arr = new int[n];
16        System.out.println("enter array elements");
17        for (int i = 0; i < arr.length; i++) {
18            arr[i] = in.nextInt();
19        }
20        //printing array before sorting
21        System.out.println("this is the array you created and before sorting");
22        System.out.print("[");
23        for (int i = 0; i < arr.length; i++) {
24            System.out.print(" " + arr[i] + " ");
25        }
26        System.out.print("]\n");
27        //calling the array
28        SHellSort(arr);
29
30
31
32        public static void SHellSort(int[] arr) { //shellsort method
33            int n = arr.length;
34            int gap;
35            for (gap = n / 2; gap > 0; gap /= 2) {
36                for (int i = gap; i < n; i++) {
37                    int temp = arr[i];
38                    int j = i;
39                    while (j >= gap && arr[j - gap] > temp) {
40                        arr[j] = arr[j - gap];
41                        j -= gap;
42                    }
43                    arr[j] = temp;
44                }
45            } //printing array
46            System.out.print("[" );
47            for (int i = 0; i < arr.length; i++) {
48                System.out.print(" " + arr[i] + " ");
49            }
50            System.out.print("]\n");
51        }
52    }
53
54 }
55
```

The output:



```
run:
enter array length
3
enter array elements
9
0
6
this is the array you created and before sorting
[ 9 0 6 ]
[ 0 6 9 ]
BUILD SUCCESSFUL (total time: 6 seconds)
```

References

- [1]. GeeksforGeeks , <https://www.geeksforgeeks.org/shellsort/>
- [2].[3]. 2braces, <https://www.2braces.com/data-structures/shell-sort>
- [4]. Wikipedia, <https://en.wikipedia.org/wiki/Shellsort>
- [5].chegg,<https://www.chegg.com/homework-help/questions-and-answers/shell-sort-variation-bubble-sort-instead-comparing-adjacent-values-shell-sort-adapts-conce-q65738120>
- [6].javatpoint, <https://www.javatpoint.com/shell-sort>

Appendix

Source code:

```
package shellsort;

import java.util.Scanner;

public class ShellSort {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int[] arr;

        //creating array from user
        System.out.println("enter array length");
        int n = in.nextInt();
        arr = new int[n];
        System.out.println("enter array elements");
        for (int i = 0; i < arr.length; i++) {
            arr[i] = in.nextInt();
        }
        //printing array before sorting
        System.out.println("this is the array you created and before sorting");
        System.out.print("[");
        for (int i = 0; i < arr.length; i++) {
            System.out.print(" " + arr[i] + " ");
        }
        System.out.print("]\n");
        //calling the array
        SHellSort(arr);
    }
}
```



```

}

public static void SHellSort(int[] arr) { //shellsort method
    int n = arr.length;
    int gap;
    for (gap = n / 2; gap > 0; gap /= 2) {
        for (int i = gap; i < n; i++) {
            int temp = arr[i];
            int j = i;
            while (j >= gap && arr[j - gap] > temp) {
                arr[j] = arr[j - gap];
                j -= gap;
            }
            arr[j] = temp;
        }
    }

    //printing array
    System.out.print("[");
    for (int i = 0; i < arr.length; i++) {
        System.out.print(" " + arr[i] + " ");
    }
    System.out.print("\n");

}

}

```