Sorting II

Intuition

Insertion Sort is slow for large values of N partly because elements can only move through the array **one position at a time**.

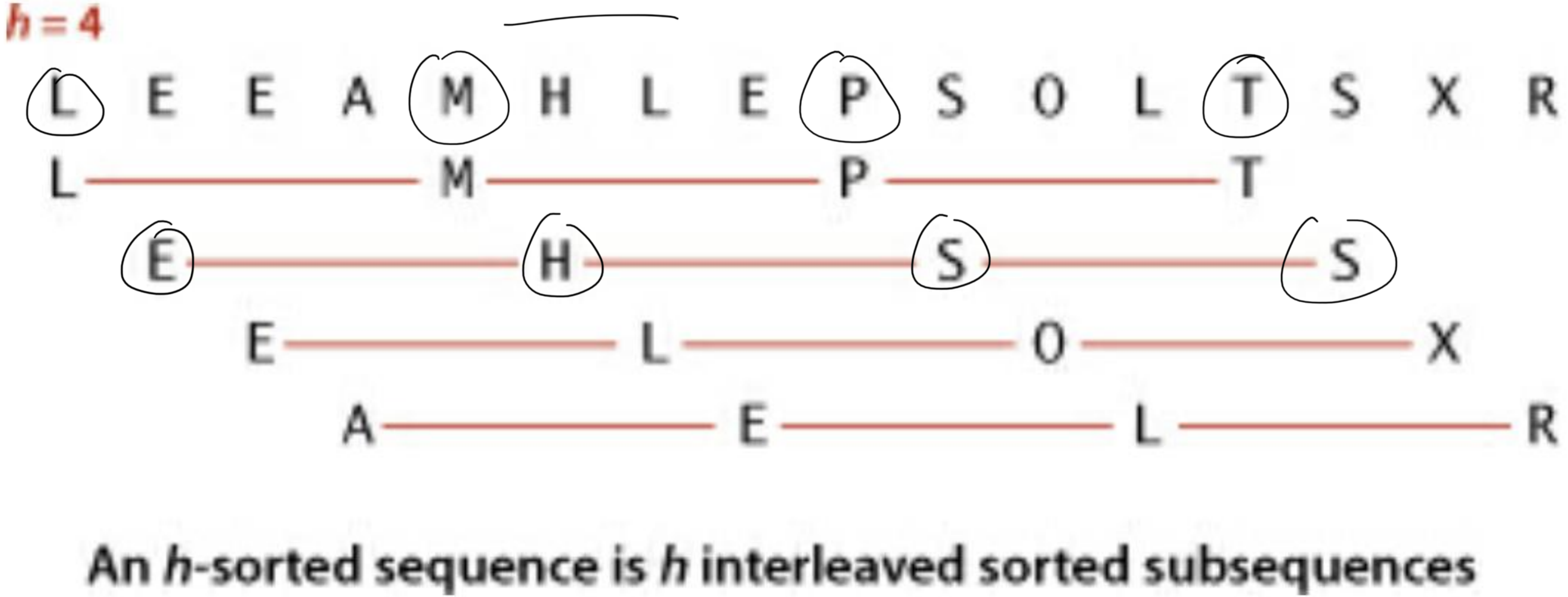
So...why not alter it so that elements can move through the array more quickly (e.g. **13, 4, etc. positions at a time**)?

Shellsort: Overview

- Use __Insertion Sort__ on every h^{th} value, creating sorted subsequences.
- Decrement the value of h according to a __h-sequence__, until $h = 1$ and you are just performing insertion sort on the array.
- When h is larger, the subsequence you are sorting is __Smaller__.
- As h gets smaller, the subsequence you are sorting gets __longer__, but it is also partially sorted.

Shellsort: *h*-sorted sequence

4 - sorted



Picture from [1]

Shellsort: Example

input	S	H	E	L	L	S	O	R	T	E	X	A	M	P	L	E
h=13 13-sort	P	H	E	L	L	S	O	R	T	E	X	A	M	S	L	E
h=4 4-sort	L	E	E	A	M	H	L	E	P	S	O	L	T	S	X	R
h=1 1-sort	A	E	E	E	H	L	L	L	M	O	P	R	S	S	T	X

Shellsort trace (array contents after each pass)

Picture from [1]

$h_k = 3h_{k-1} + 1$

$h_1 = 1$

$h_1 = 1$

$h_2 = 4$

$h_3 = 13$

$h_4 = 40$

$\vdots$

$h \geq N/3$

$N = 11$

$N/3 = 3.....$

$h = 4$

4 3 0 1 2 0 9 8 7 9 4

2 3 4 0 7 9

2 0 0 4 3 9 7 9 4

$h = 1$

2 0 0 1 4 3 4 8 7 9 9

0 2 0

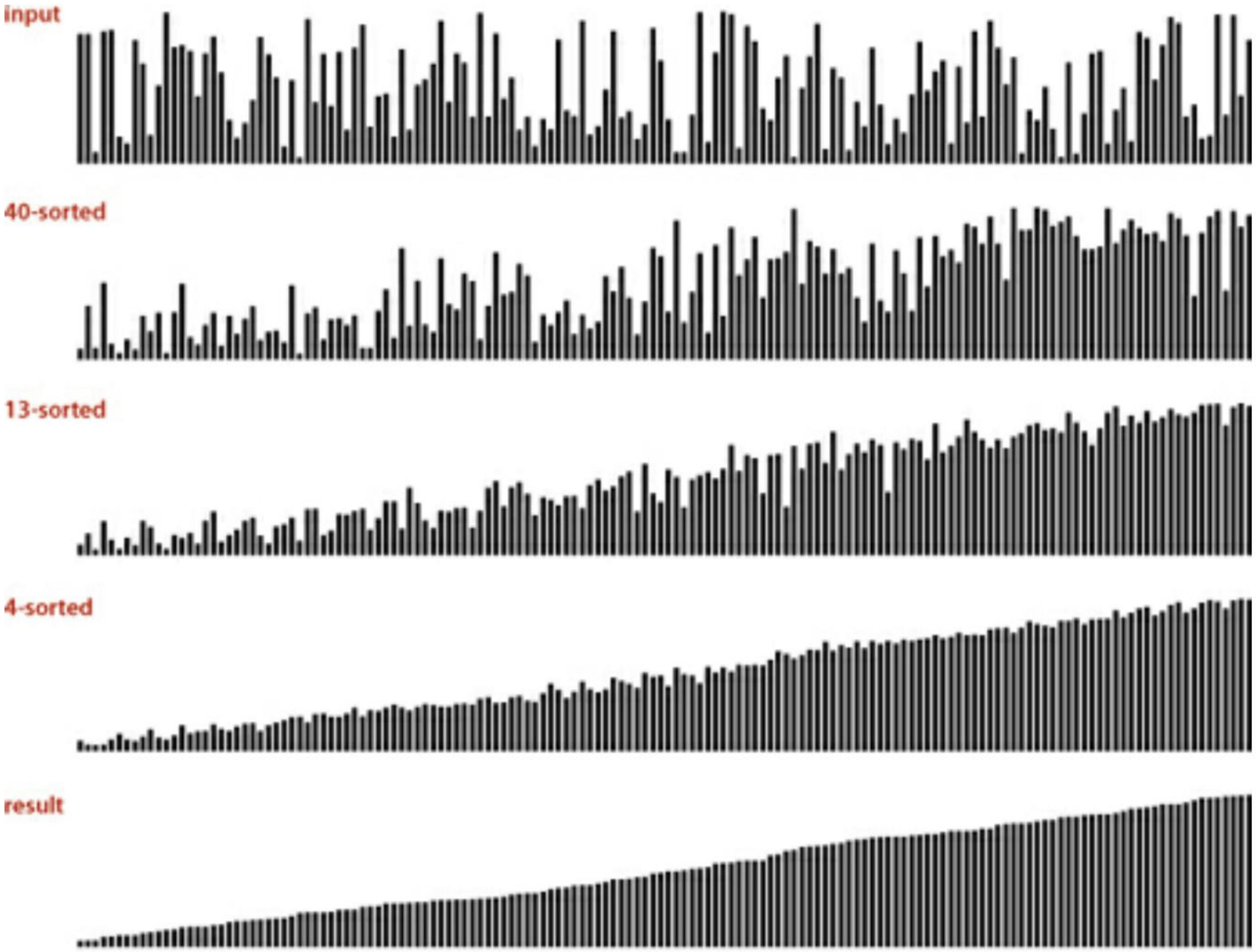
0 0 2 1

0 0 1 2 4

input	S	H	E	L	L	S	O	R	T	E	X	A	M	P	L	E
13-sort	P	H	E	L	L	S	O	R	T	E	X	A	M	S	L	E
	P	H	E	L	L	S	O	R	T	E	X	A	M	S	L	E
	P	H	E	L	L	S	O	R	T	E	X	A	M	S	L	E
4-sort	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E
	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E
	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E
	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E
	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E
	L	E	E	L	P	H	O	R	T	S	X	A	M	S	L	E
	L	E	E	L	P	H	O	R	T	S	X	A	M	S	L	E
	L	E	E	A	P	H	O	L	T	S	X	R	M	S	L	E
	L	E	E	A	M	H	O	L	P	S	X	R	T	S	L	E
	L	E	E	A	M	H	O	L	P	S	X	R	T	S	L	E
	L	E	E	A	M	H	L	L	P	S	O	R	T	S	X	E
	L	E	E	A	M	H	L	E	P	S	O	L	T	S	X	R
1-sort	E	L	E	A	M	H	L	E	P	S	O	L	T	S	X	R
	E	E	L	A	M	H	L	E	P	S	O	L	T	S	X	R
	A	E	E	L	M	H	L	E	P	S	O	L	T	S	X	R
	A	E	E	L	M	H	L	E	P	S	O	L	T	S	X	R
	A	E	E	H	L	M	L	E	P	S	O	L	T	S	X	R
	A	E	E	H	L	L	M	E	P	S	O	L	T	S	X	R
	A	E	E	E	H	L	L	M	P	S	O	L	T	S	X	R
	A	E	E	E	H	L	L	M	P	S	O	L	T	S	X	R
	A	E	E	E	H	L	L	M	P	S	O	L	T	S	X	R
	A	E	E	E	H	L	L	M	O	P	S	L	T	S	X	R
	A	E	E	E	H	L	L	L	M	O	P	S	T	S	X	R
	A	E	E	E	H	L	L	L	M	O	P	S	T	S	X	R
	A	E	E	E	H	L	L	L	M	O	P	S	S	T	X	R
	A	E	E	E	H	L	L	L	M	O	P	S	S	T	X	R
	A	E	E	E	H	L	L	L	M	O	P	R	S	S	T	X
result	A	E	E	E	H	L	L	L	M	O	P	R	S	S	T	X

Detailed trace of shellsort (insertions)

Picture from [1]



Visual trace of shellsort

Picture from [1]

$\rightarrow h_k + 2h_{k-1}$ $\rightarrow h_k = 2h_{k-1} + 2 \leftarrow$

Analysis

$h_k = 2h_{k-1} + 3$

- The *h*-sequence matters, as it affects HOW sorted the subsequences are and if they overlap a lot, you may be doing extra, redundant work.
- Performance depends on:
  - the # of h-values
  - the relationship b/t the h-values
- This makes analysis of the performance difficult!
- So far, no one has found a __provably best__ *h*-sequence.
- The implementation presented here: worst-case number of compares is __$O(N^{3/2})$__, which is still better than __$O(N^2)$__.
- Reminder: The *h*-sequence used in these slides is determined by $h_k = 3h_{k-1}+1$ and $h_0=0$, and the first *h* is chosen to be the first one where $h >= N/3$.

Shellsort Summary

- Small alteration to insertion sort beats quadratic time!
- Generally acceptable running time for moderately large arrays
- Requires small amount of code
- Requires no extra memory

References

[1] *Algorithms, Fourth Edition;* Robert Sedgewick and Kevin Wayne (and associated slides)