Analysis of Shellsort

Your project will be on the analysis of shellsort and is divided into two parts. The first part is concerned with some theoretical analysis of shellsort. The second section is concerned with performing an empirical analysis of shellsort. If you wish to analyze another algorithm in your project, check with me first.

Background

The following resources provide a useful background on shellsort.


Theoretical Analysis Section

In this analysis, we will show certain properties of the SHELLSORT algorithm below. Assume that $A$ consists of numbers.

\[
\text{SHELLSORT}(A) \\
\begin{align*}
1 & \quad n \leftarrow A.length \\
2 & \quad h \leftarrow 1 \\
3 & \quad \textbf{while} \ h < n \\
4 & \quad \quad h \leftarrow 2h + 1 \\
5 & \quad h \leftarrow \lfloor h/2 \rfloor \\
6 & \quad \textbf{while} \ h > 0 \\
7 & \quad \quad \textbf{for} \ j \leftarrow h + 1 \ \text{to} \ n \\
8 & \quad \quad \quad key \leftarrow A[j] \\
9 & \quad \quad \quad i \leftarrow j - h \\
10 & \quad \quad \quad \textbf{while} \ i > 0 \ \text{and} \ A[i] > key \\
11 & \quad \quad \quad \quad A[i + h] \leftarrow A[i] \\
12 & \quad \quad \quad \quad i \leftarrow i - h \\
13 & \quad \quad \quad A[i + h] \leftarrow key \\
14 & \quad \quad h \leftarrow \lfloor h/2 \rfloor
\end{align*}
\]

1. Illustrate the behavior of SHELLSORT on the array $A = \langle 9, 8, 7, 6, 5, 4, 3, 2, 1 \rangle$. Show the value of the array after each iteration of the for loop.
2. What are the loop invariants? Note that there are four loops.

Define \( h \)-ordered to mean that \( A[i] \leq A[i + h] \) for all \( i \) such that \( 1 \leq i \leq n - h \). Define \( h \)-sorting to indicate the process that makes \( A \) \( h \)-ordered by reordering \( A[i-h], A[i], A[i+h], \ldots \). Exercises 3–5 develop one of the key properties of SHELLSORT: The result of \( h \)-sorting an array that is \( k \)-ordered is an array that is both \( h \)-ordered and \( k \)-ordered.

3. Let \( B \) and \( C \) be arrays of size \( m \). Suppose that \( B[i] \leq C[i] \) for \( 1 \leq i \leq m \). Show that, after \( B \) and \( C \) are independently sorted (no numbers are exchanged between \( B \) and \( C \)), it is still true that \( B[i] \leq C[i] \) for \( 1 \leq i \leq m \). Hint: After \( B \) and \( C \) are sorted, determine how many numbers from \( C \) must be greater than or equal to \( B[i] \). Note that each number in \( B[i..m] \) must be less than or equal to a specific number from \( C \). You might find it useful to create a specific example to see how this works.

4. Let \( B \) and \( C \) be arrays of size \( m \). Suppose there exists a \( j > 0 \) such that \( B[i] \leq C[i+j] \) for \( 1 \leq i \leq m-j \). Show that, after \( B \) and \( C \) are sorted, it is still true that \( B[i] \leq C[i+j] \) for \( 1 \leq i \leq m-j \). Hint: Again determine how many numbers in \( C \) must be \( \geq \) to \( B[i] \) after \( B \) and \( C \) are sorted. Again, you might find it useful to create a specific example to see how this works.

5. Let \( k > h > 1 \). Suppose \( A \) is \( k \)-ordered. Show that \( A \) remains \( k \)-ordered after \( A \) is \( h \)-sorted. Hint: Use the result of Exercise 4 on the \( k \)-ordered array. Let \( \ldots, A[i-h], A[i], A[i+h], \ldots \) be the \( B \) array, and let \( \ldots, A[i+k-h], A[i+k], A[i+k+h], \ldots \) be the \( C \) array. And again, you might find it useful to create a specific example to see how this works; use \( k = 3 \) and \( h = 2 \) in your example.

What this implies is that when \( A \) is \( h \)-ordered, then \( A \) will remain \( h \)-ordered for the rest of the algorithm. This property is what makes shellsort more efficient than insertion sort.

**Empirical Analysis Section**

The above SHELLSORT algorithm uses a specific sequence of \( g \text{aps} \), values of \( h \) used by the for loop.

\[
1, 3, 7, 15, 31, \ldots
\]
The algorithm starts with the largest gap that is less than \( A.length \), and proceeds backward to \( h = 1 \).

There are many other gap sequences that could be used with different theoretical and empirical results. The background material mentions several of them. You should try at least four gap sequences.

- 1, 3, 7, 15, 31, \ldots as in the above pseudocode. This gap sequence is defined by \( h_i = 2^i - 1 \), but gap sequences do not have to have a simple definition. It is okay to directly initialize an array with the gap sequence.

- A gap sequence that appears to be among the best by empirical tests as indicated in the background material.

- A gap sequence that appears to be among the best by theoretical analysis as indicated in the background material. If you use Pratt’s method, instead of 2 and 3, use a higher pair of relatively prime numbers.

- Your very own completely new gap sequence. You are unlikely to find a gap sequence that is better than everything else, but it should at least be competitive with the 1, 3, 7, 15, 31, \ldots gap sequence. You can add more of your own gap sequences if you wish.

You should explain why the three additional gap sequences meet these criteria.

For data, you should run shellsort on varying sizes of \( A.length \), where the values in \( A \) are generated randomly. Generating random doubles is recommended. It should be unlikely that you have duplicate values in an array.

Your maximum \( A.length \) should be at least \( n_{max} = 1,000,000 \). You should at least test for array sizes of \( n_{max}/10 \), \( 2 \times n_{max}/10 \), \( 3 \times n_{max}/10 \), \ldots, \( 9 \times n_{max}/10 \), \( n_{max} \) and on array sizes of 10, 100, 1000, \ldots, \( n_{max} \). For each array size and gap sequence, you should run your program at least 10 times. A higher \( n_{max} \), more array sizes, and more runs will improve the quality of your graphs.

For each run, you should determine the number of comparisons. In the pseudocode above, this would be the number of times \( A[i] > key \) is performed. Note that if \( i \leq 0 \) (or \( i < 0 \) if the algorithm is modified for zero-indexed arrays), then \( A[i] > key \) is not performed and so the number of comparisons should not be incremented in this case. You will be plotting the average number of comparisons for a given array size and gap sequence.

**Performance Analysis**

6. Display and describe a graph with the \( x \)-axis as the size of the array and the \( y \)-axis as the average number of comparisons. Both the \( x \)-axis and \( y \)-axis should be linearly scaled. This graph should have four different lines for the four different gap sequences that you chose.
7. Display and describe a graph with the $x$-axis as the size of the array and the $y$-axis as the average number of comparisons. Both the $x$-axis and $y$-axis should be logarithmically scaled. This graph should have four different lines for the four different gap sequences that you chose.

8. Display and describe a graph with the $x$-axis as the size of the array and the $y$-axis as the average number of comparisons divided by $\lg(n!)$. Note that $\lg(n!)$ is the theoretical lower bound for the number of comparisons by a comparison sorting algorithm. Both the $x$-axis and $y$-axis should be linearly scaled. This graph should have four different lines for the four different gap sequences that you chose.

9. Display and describe a graph with the $x$-axis as the size of the array and the $y$-axis as the average number of comparisons divided by $\lg(n!)$. Note that $\lg(n!)$ is the theoretical lower bound for the number of comparisons by a comparison sorting algorithm. Both the $x$-axis and $y$-axis should be logarithmically scaled. This graph should have four different lines for the four different gap sequences that you chose.

Submission
In addition to your report, also submit your source code as a zip file. A README file should be included if there are any special commands that are needed to compile or run your source code. The report should be in PDF format.