Heap-Sort Accelerates Dinky Database Dantai's Zip-Code-Sorting Operation

Suraj Kurapati
<skurapat@ucsc.edu>
CMPE-185, Winter 2005

June 8, 2006

0.1 Executive Summary

Problem Our most *respected* customers are unsatisfied with the slow performance of our database's zip-code-sorting operation. A variant of the bubble-sort algorithm, used by our zip-code-sorting operation to sort zip-codes, yields substandard performance when sorting a large number of zip-codes. Thereby damaging customer-relations, incurring revenue-loss, and tainting our unparalleled reputation of engineering egregious software.

Solution Replace our variant of the bubble-sort algorithm with the heap-sort algorithm—which can significantly improve the performance of our zip-code-sorting operation while preserving existing data-structures and their related procedures [7].

Difficulty Implementing the proposed algorithm, precisely detailed in Section 0.2 on the following page, requires no technical expertise in excess of that possessed by our entry-level programmers, and therefore can be realized[1] quickly enough to salvage our reputation and possibly strengthen customer-relations.

Hence, the process of applying the proposed solution can be summarized as: the (1) realization and (2) testing of the heap-sort algorithm, and (3) its incorporation with our zip-code-sorting operation; all of which can be performed by our entry-level programmers.

[1] Brought into existence.

0.2 Technical Description of Heap-Sort

> "In order to understand recursion, one must first understand recursion; after that, it's easy." –Andrew Koenig [3]

The reader's knowledge of arrays, binary trees, character-string comparison and exchange operations, and our zip-code-sorting operation's variant of bubble-sort algorithm—abbreviated as "BS" henceforth—is assumed.

0.2.1 Overview

Heap-Sort is a recursive divide-and-conquer sorting-algorithm that arranges a given input-sequence into non-decreasing[2] or non-increasing[3] order. Such algorithms simplify a given task, such as sorting a sequence of zip-codes, into smaller *sub-tasks*; each of which are easier to perform and require less time to finish than the original task—due to their reduced size. For example, a task of sorting a hundred zip-codes can be simplified into two sub-tasks which sort fifty zip-codes respectively.

> "Wait a minute...," cautions the reader. "Why don't we simplify those sub-tasks into *yet* smaller sub-tasks? Doing so would allow for faster completion each sub-task and thus increase the efficiency of the overall task."

Why, of course! Each sub-task can indeed be simplified once more, each of its resulting sub-tasks can be simplified yet again, and this *recursive* process can continue indefinitely. However, businesses, and customers alike, require that an algorithm compute its result within a finite period of time. In other words, algorithms that compute results in a shorter amount of time are preferred. Therefore, a recursive algorithm must only simplify sub-tasks until a termination-condition is reached; that is, until the task at hand is so simple that its result can be computed without further simplification.

Now, suppose the algorithm has finished computing the results of all maximally-simplified sub-tasks. The independent result of any one of these sub-tasks is an insufficient substitute for the original task's result. Therefore, a method of combination is necessary to transform these independent results into a collective-result sufficient for original task. Likewise, a method of division is necessary to help the algorithm decide how each task should be simplified. These methods are detailed, with respect to our zip-code-sorting operation, in Section 0.2.3 on the next page.

0.2.2 Data-Structures

The heap-sort algorithm shall reuse BS's existing data-structures and their related operations. Namely, an array of zip-codes—each of which is a variable-

[2] Each successive element is either the same as or larger than the preceding one.
[3] Each successive element is either the same as or smaller than the preceding one.

length character-string—and its associated zip-code comparison and exchange operations.

0.2.2.1 The Heap

A heap, shown in Figure 1 on the following page, is a kind of tree. Each node in the heap has an associated comparable and exchangeable value, and all sequences of nodes from the heap's root to each of its leaves are of either non-decreasing—the property of a *min-heap*—xor[4] non-increasing—the property of a *max-heap*—order [2]. Also, sequences of sibling nodes, such as $\langle d, e, f \rangle$ in Figure 1, are not guaranteed to be in any particular order [2].

Array-Representation A binary min-heap, whose nodes may have a maximum of two children, is often represented by an array. The following set of functions [2, 7] translate its hierarchical structure into a linear form, for storage in an array.

- The first array-position is always occupied by the root node.
- The remaining array-positions are occupied by non-root nodes—one node per position—whose children are accessible via:

$$child_1(x) = 2x \quad (1)$$
$$child_2(x) = 2x + 1 \quad (2)$$

  Here, x denotes the array-position occupied by a certain node. Equations 1 and 2 respectively define the array-positions of its first and second child.

For example, Figure 1 shows an alphabetical binary min-heap whose array-representation is $\langle a, b, c, d, e, f, \emptyset \rangle$. Since node c has only one child, the array-position of the second child is occupied by $\emptyset$—the empty node.

0.2.3 Technique

Our implementation of the heap-sort algorithm shall arrange an array of zip-codes in an alphabetically non-decreasing order through the use of a binary min-heap.

Method of Division Each node in the binary min-heap has an associated zip-code that is smaller than those of its children. Thus, sequences of zip-codes from the root of the binary min-heap to each of its leaves shall be in non-decreasing order.

[4] Either this or that, but not both.

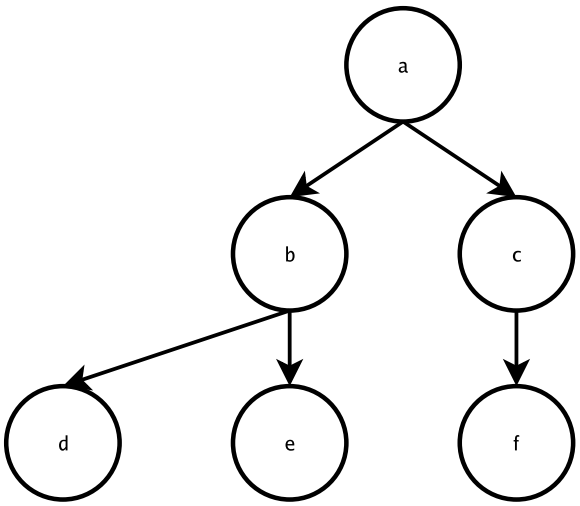

Figure 1: Illustrated above is an alphabetical binary min-heap. Notice how all sequences of nodes from its root a to each of its leaves $\langle d, e, f \rangle$ produces a non-decreasing sequence of alphabets.

Method of Combination Two nodes are compared to determine if one has an associated zip-code larger than, smaller than, or the same as the other. Depending upon this result, the two values are either exchanged xor ignored because they are already in the desired order. For example, if a parent node has an associated zip-code larger than that of its child, then their positions within the binary min-heap are exchanged. Therefore, the method of combination maintains the method of division.

0.2.3.1 Pseudo-Code

The following procedure, adapted from [1], is written in high-level pseudo-code to illustrate the conceptual steps performed by the heap-sort algorithm during sorting. [7] provides a pseudo-code listing that readily translates into a procedure for use in an actual programming language.

1. An array of zip-codes, which is to be sorted, is prepared for use as an input-sequence.
2. An empty binary min-heap and is prepared for use.
3. A zip-code is removed from the input-sequence and inserted into the binary min-heap via the method of combination.
   (a) If a node x is to be inserted as the child of a node y, then the zip-code associated with x must be larger than that of y. Otherwise, the two nodes x and y must be exchanged.

4. Step 3 is repeated until all zip-codes from the input-sequence have been inserted into the binary min-heap.

5. An empty output-sequence is prepared for use.

6. The root of the binary min-heap is detached and its associated zip-code is placed at the end of the output-sequence.

   (a) The right-most sibling node, located within the sequence of sibling nodes that are at the very bottom of the binary min-heap, is placed at the root of the binary min-heap. (See node f in Figure 1.)

   (b) The new root node is sifted down through the binary min-heap, using the rules defined in Step 3a. For example, in Figure 1, if node f was the new root, then it would be exchanged with node b.

7. Step 6 is repeated until all nodes have been removed from the binary min-heap and their associated zip-codes have been placed into the output-sequence.

8. The output-sequence now contains zip-codes arranged in an alphabetically non-decreasing order.

Note that the binary min-heap and output-sequence are purely theoretical constructions. Hence, one can implement this algorithm in such a way that these abstractions use the same data-storage space as the input-sequence—a technique called "in-place" sort [2, 7].

0.2.4 Justification

Heap-Sort is a well known, time-tested sorting-algorithm which has been thoroughly analyzed by professional computer-scientists and researchers in major universities during its heyday. The creation and verification of a formal proof of correctness for this algorithm are left as an exercise for the interested reader.

0.2.5 Analysis

0.2.5.1 Measuring Complexity

In order to compare sorting-algorithms, computer-scientists often employ the theoretical asymptotic-measure of running-time complexity, or "complexity-measurement" for short, which models the number of comparisons, exchanges, or similar operations performed by an algorithm [4]. These measurements accurately model the complexity of algorithms for sufficiently large[5] input-sequences in practice; thus enabling one to gauge the efficiency of algorithms without having to implement them [5].

The tightly bounded complexity-measurement, defined by the following big-theta notation [5], is most useful in our analysis because it provides both a

[5] Approaching ∞.

lower- and upper-bound measure of complexity. For example, the function f is asymptotically bounded both from above, by βg; and from below, by αg.

$$\Theta(g) \equiv \{f \mid (\exists \alpha, \beta, \gamma > 0 \wedge \forall x \geq \gamma \mid 0 \leq \alpha g \leq f \leq \beta g)\}$$

To further illustrate this point, consider a sorting-algorithm, which requires $f(x) = x^2 + 5x + 10$ number of instructions to sort an input-sequence. Using the aforementioned definition, we determine that this function has a complexity-measurement of $\Theta\left(x^2\right)$ when $\alpha = 1$, $\beta = 2$, and $\gamma = 7$.

0.2.5.2 Special Conditions

Conditions known as *best-*, *worst-*, and *average-cases* aid in the comparison of complexity-measurements from different algorithms.

Best-Case refers to sorting an input-sequence that is already in the wanted order. For example, a best-case input-sequence for our zip-code-sorting operation would be $\langle 1, 2, 2, 3 \rangle$.

Worst-Case refers to sorting an input-sequence that is in the exact *opposite* of the wanted order. For example, a worst-case input-sequence for our zip-code-sorting operation would be $\langle 3, 2, 2, 1 \rangle$.

Average-Case refers to sorting an input-sequence whose contents are in no particular order. Thus, an input-sequence classified as best- xor worst-case can also be classified as an average-case.

The probability of a best- xor worst-case condition, modeled by $P(x) = \frac{1}{x!}$ where $x \geq 2$, is shown in Figure 2 on the next page. Notice how the likelihood of an input-sequence being classified as a best- xor worst-case diminishes as the size of the input-sequence grows sufficiently large. Thus, complexity-measurements observed under these cases can be, more or less, disregarded for input-sequences containing at least five elements—as inferred from Figure 2.

0.2.5.3 Algorithm Comparison

As table 1 shows, the heap-sort algorithm has lower complexity-measurements than those of the bubble-sort algorithm, and thus BS, in all but the best-case. However, the best-case will rarely occur (See Figure 2) as the length of an input-sequence grows large. Hence, the heap-sort algorithm is the better choice and its employment will yield faster performance of our zip-code-sorting operation.

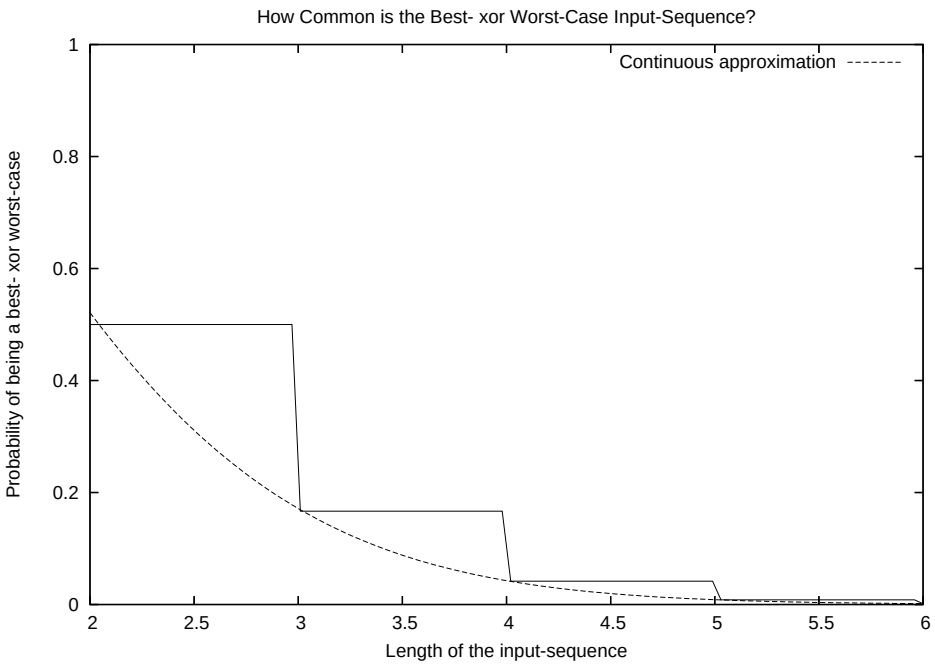

Figure 2: The probability of a best- xor worst-case shrinks as the input-sequence grows in length. The continuous approximation [6] illustrates this trend within the non-integer intervals of $\mathbb{R} \cap \overline{\mathbb{Z}}$.

Algorithm	Best-Case	Worst-Case	Average-Case
Bubble-Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$
Heap-Sort	$\Theta(n * \log(n))$	$\Theta(n * \log(n))$	$\Theta(n * \log(n))$

Table 1: Various complexity-measurements of bubble- and heap-sort—provided by [4, 8]—are shown above. Notice that heap-sort exhibits the same complexity in all cases.

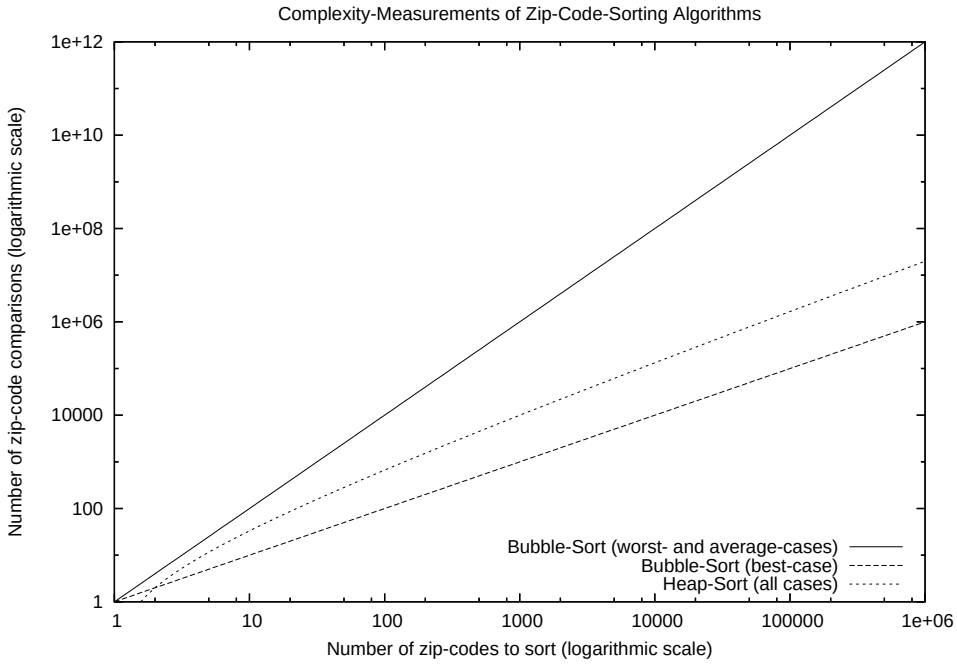

Figure 3: The complexity-measurements shown above model the number of comparisons needed by each algorithm to sort an entire input-sequence. Also, note the use of logarithmic scales; the worst- and average-case complexities of bubble-sort dwarf those of heap-sort.

Bibliography

[1] W. Ang, "Heap Sort Animation," [Online document], 1998 Sep 28, [cited 25 Jan 2005], Available HTTP: `http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/heapsort.html`

[2] R. Johnsonbaugh and M. Schaefer, *Algorithms*, Upper Saddle River, N.J.: Pearson Education, 2004.

[3] Portland Pattern Repository, "Moment Of Understanding," [Online document], 28 Jul 2003, [cited 21 Jan 2005], Available HTTP: `http://c2.com/cgi/wiki?MomentOfUnderstanding`

[4] SparkNotes LLC, "BubbleSort: The BubbleSort Algorithm," [Online document], 11 Jan 2005, [cited 21 Jan 2005], Available HTTP: `http://www.sparknotes.com/cs/sorting/bubble/section1.html`

[5] P. Tantalo, "Asymptotic Growth of Functions," [Online document], 8 Jul 2004, [cited 22 Jan 2005], Available HTTP: `http://www.soe.ucsc.edu/classes/cmps101/Summer04/asymptotic.pdf`

[6] E. W. Weisstein, "Stirling's Approximation," [Online document], 1999, [cited 23 Jan 2005], Available HTTP: `http://mathworld.wolfram.com/StirlingsApproximation.html`

[7] Wikimedia Foundation, Inc. "Heapsort," [Online document], 2005 Jan 21, [cited 2005 Jan 22], Available HTTP: `http://en.wikipedia.org/wiki/Heapsort`

[8] O. d. Vel, "CP2001 Lecture Notes - Table ADT and Hashing," [Online document], 1998, [cited 22 Jan 2005], Available HTTP: `http://www.it.jcu.edu.au/Subjects/cp2001/1998/LectureNotes/Sorting/node2.html`