Project Deliverable 4

Alex Abrahamson - Mitchell Baker - Brock Ellefson - Yue Hou - Seth Severa

27 October 2017

Introsort problem description

Introspective sort is a sorting algorithm that combines the quick average time of Quicksort with the stable worst case performance of Heapsort developed by Dr. David Musser in 1997. Sorting algorithms often need to find a compromise between fast execution in the common case and a reliable performance when it comes to problematic inputs.

Quicksort is perhaps the fastest sorting algorithm in practice, but fails to hold up when it chooses pivot values far away from the midpoint of its partition. When dividing subarrays on a value greater or less than the majority of the other elements, Quicksort's time complexity moves from $O(n\log n)$ to $O(n^2)$. Different methods of combating this issue have been implemented, including random pivot values, the median-of-three method, and even linearly searching the partition for the optimal median, but none provide us with a complete resistance to bad inputs.

The problem Introspective Sort solves is that very balance between a fast common case algorithm, and a fast worst case algorithm. By combining the speed of Quicksort with the reliability of Heapsort, Introsort guards itself from median-of-three killers but maintains a faster runtime than other $O(n\log n)$ worst case algorithms by themselves.

When running Introsort, the algorithm functions as Quicksort until a certain number of partitions has been reached, in which case it transitions into the Heapsort algorithm. We determine the size threshold for revertign to heapsort as $2[log_2 n]$. We set this limit so as to preserve the logarithmic running time we desire, without cutting off Quicksort prematurely. Even though Heapsort is on average slower than Quicksort, on the specific instances where our input array would tend Quicksort towards quadratic time, Introsort uses the combination to improve speed up to a factor of 5.

Identification of the +1 element

The +1 element for the project will be an analysis of the time complexity of Introsort, and how the time complexity was found. For the analysis, the complexity of Quicksort combined with the complexity Heapsort will be used. David Musser mentions in his paper on sorting and selection algorithms the idea of using iterators rather than parring array indexes by reference. This apparently speeds up the algorithm, we are going to implement the use of iterators and then compare this new implementation to the older version of Introsort and its competitors.

Algorithm 1 INTROSORT

```
1:  Procedure Sort( A)
2:  maxdepth = 2⌊log‖A‖⌋
3:  Introsort( A, maxdepth)
4:
5:  Procedure Introsort( A, maxdepth)
6:  in: A, maxdepth
7:  n ← |A|
8:
9:  if n ≤ 1 then
10:     return
11: else if maxdepth = 0 then
12:     HEAPSORT(A)
13: else
14:     p ← PARTITION(A)
15:     INTROSORT(A[0:p], maxdepth - 1)
16:     INTROSORT(A[p + 1:n], maxdepth −1)
17: end if
```

Loop invariant

In the recursive procedure of INTROSORT, the loop invariant L is

$L = \{$ A is the concatenation of the return values of INTROSORT(A[0:p], maxdepth $- 1$) and INTROSORT(A[0:p], maxdepth $- 1$, up until the maxdepth value reaches 0. $\}$

Following this, in the HEAPSORT procedure the loop invariant L becomes

$L = \{$ At the start of each iteration of the loop, the subarray $A[1, ..., i]$ is a max heap containing the i^{th} smallest elements of A, and the subarray $A[i + 1, ..., n]$ contains the $n - i$ largest elements of $A[1, ..., n]$ sorted. $\}$ (CLRS, 160).

References

[1] Musser, David. (1997). Introspective Sorting and Selection Algorithms. Software Practice and Experience. 27. . 10.1002/(SICI)1097-024X(199708)27:8<983::AID-SPE117>3.0.CO;2-#.

[2] Cormen, Thomas H., et al. *Introduction to Algorithms*. 3rd ed., MIT Press, 2009.