Dylan House
Prof. Mark Lehr
CIS-17C-48596
November 17, 2014

Time Complexity Analysis of Various Sorting Methods

Selection Sort

Two for loops.
The outer loop iterates n – 1 times.
For each i between 0 and n – 2, the inner loop iterates j = (n – 1) – i times.
The comparisons of data are done in the inner loop, so there are

$$\sum_{i=0}^{n-2} (\mathrm{n} - 1 - \mathrm{i}) = (n-1) + \cdots + 1 = \frac{n(n-1)}{2} = O(n^2)$$

comparisons.

Bubble Sort

The number of comparisons for best, average, and worst cases are the same and equal the total number of iterations of the inner for loop.

$$\sum_{i=0}^{n-2} (\mathrm{n} - 1 - \mathrm{i}) = (n-1) + \cdots + 1 = \frac{n(n-1)}{2} = O(n^2)$$

Quicksort

In the worst case, the algorithm operates on arrays of size n – 1, n – 2, … , 2. The partitions require n – 2 + n – 3 + … + 1 comparisons. Thus the run time is equal to $O(n^2)$.

The best case is when the bound divides an array into two subarrays of approximate length $\frac{n}{2}$. If the bounds are well chosen, the partitions produce four new subarrays $\frac{n}{4}$… eight new subarrays $\frac{n}{8}$, etc… Therefore the number of comparisons performed for all partitions is

$$n + 2\frac{n}{2} + 4\frac{n}{4} + 8\frac{n}{8} + \cdots + n\frac{n}{2} = n(\lg n + 1) = O(n \lg n)$$

Heap Sort

In the first phase of heapsort(), it uses moveDown(), which performs O(n) steps.
In the second phase, heapsort() exchanges n – 1 times the root elements in position I and also restores the heap n – 1 times, which in the worst case causes moveDown() to iterate lg i times. The total number of moves in all executions of moveDown() in the second phase of heapsort() is $\sum_{i=1}^{n-1} \lg i$, which is O(n lg n). In the worst case, heapsort() requires O(n) steps in the first phase, and n – 1 swaps in the second phase and O(n lg n) operations to restore the heap property, which gives O(n) + O(n lg n) + (n – 1) = O(n lg n) exchanges.
For the best case, when the array contains identical elements, n comparisons are made in the first phase and 2(n – 1) in the second. The total number of comparisons in the best case is O(n). But, if the array has distinct elements, then the number of comparisons equals
n lg n – O(n).

Mergesort

For an n-element array, the number of movements is computed by the following recurrence relation:

$$M(1) = 0$$

$$M(n) = 2M\left(\frac{n}{2}\right) + 2n$$

M(n) is computed like so:

$$\begin{aligned} M(n) &= 2\left(2M\left(\frac{n}{4}\right) + 2\left(\frac{n}{4}\right)\right) + 2n = 4M\left(\frac{n}{4}\right) + 4n \\ &= 4\left(2M\left(\frac{n}{8}\right) + 2\left(\frac{n}{4}\right)\right) + 4n = 8M\left(\frac{n}{8}\right) + 6n \\ &= 2^i M\left(\frac{n}{2^i}\right) + 2in \end{aligned}$$

Choosing i = lg n so that n = 2^i allows us to infer

$$M(n) \ = 2^i M\left(\frac{n}{2^i}\right) + 2in = nM(1) + 2n\lg n = 2n\lg n = O(n\lg n)$$

Works Cited

Drozdek, Adam. "Sorting." *Data Structures and Algorithms in C++*. Fourth ed. Cengage Learning, 2012. 495, 497, 508, 514, 518. Print.