Deakin University

Data Structures and Algorithms

OnTrack Submission

Algorithm complexity (credit)

Submitted By:
Peter Stacey
pstacey
2020/08/13 20:42

Tutor:
Maksym Slavnenko

Outcome	**Weight**
Complexity	♦♦♦♦♦
Implement Solutions	♦◊◊◊◊
Document solutions	♦◊◊◊◊

This involves additional investigation into the time complexity of algorithms and extends the work in task 2.1 by including comparisons between algorithms to identify trends in the way they scale against each other. It's an in depth application of knowledge for ULO1 and not associated with either ULO2 or ULO3.

August 13, 2020

Task 2.2

Student Name: Peter Stacey

Student ID: 219011171

Question 1

Method	Performance of My Implementation	Performance of Microsoft Implementation	Comment
Insert	*O(n)* search	*O(n)* search	The Microsoft implementation contains some additional logic to match null objects, which adds 1 extra operation, however overall the performance of both implementations are *O(n)*
Clear	O(1)	O(1) or O(n) depending on implementation of Array.Clear, which I couldn't find in the source code, as Array.cs calls an Interface.	The Microsoft implementation uses Array.Clear() and sets the size of the Array to 0, in order to allow the garbage collector to reclaim the references. My implementation simply declares a new T[] and sets the Count to 0. Depending on the implementation of Array.Clear by Microsoft, their approach is either O(1) or O(n), while my approach is definitely O(1). In this case, my algorithm performs no worse than Microsoft's, but may perform better.
Contains	*O(n)* linear search	*O(n)* linear search	The implementations are essentially the same.
Remove	*O(n)* (2n operations)	*O(n)* (2n operations)	Implementations are essentially the same.
RemoveAt	*O(n)*	*O(n)*	The Microsoft implementation uses Array.Copy(), passing the existing array into the method as an argument. Ultimately, this

			calls CopyTo with performs at O(n). My implementation shuffles the values within the existing array and has the same performance as the MS implementation.
ToString	*O(n)*		list.cs does not contain an implementation of ToString, which is type dependent, so the method is contained in other files. This one is difficult to comment on as a result of possible different performance based on the type declared for the Vector.

Question 2

Function: *f(n) = nlog n*

Well known Algorithm: Merge Sort

Show that the function satisfies:

f is in $O(n^2)$:

$\lim_{n\to\infty} g(n) / f(n) = n^2 / n \log n = \infty \Rightarrow n^2 = \omega(n \log n)$

$\therefore f(n\log n)$ is in $O(n^2)$

f is in $\Omega(n)$:

$\lim_{n\to\infty} f(n) / g(n) = n\log n / n = \log n = \infty \Rightarrow n\log n = \omega(n)$

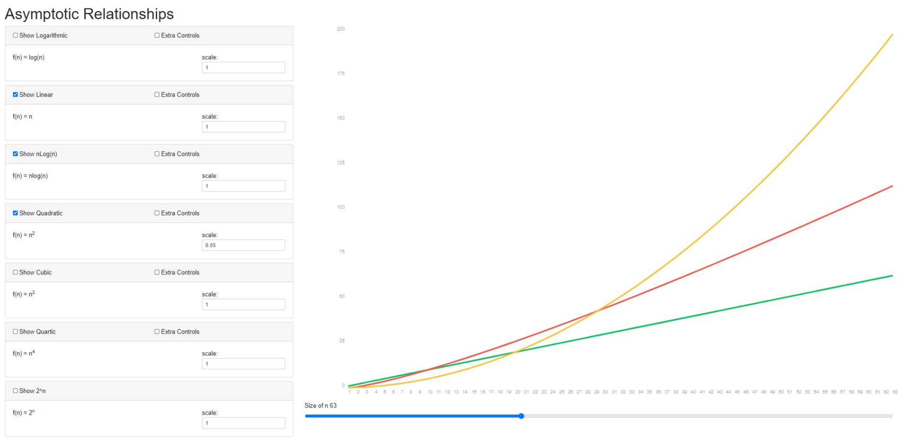
$\therefore f(n\log n)$ is in $\Omega(n^2)$

F is neither in $\Theta(n)$ nor in $\Theta(n^2)$, but can be represented with Θ:

From the above, *nlog n* can be dscribed as growing faster than *f(n) and slower than* $g(n^2)$, so it is neither $\Theta(n)$ nor $\Theta(n^2)$.

However, f(nlog n) has an tightest upper bound of *O(nlog n)* and a tightest lower bound of *Ω(nlog n)* and is therefore also, *Θ(nlog n).*

Graphically, this can also be demonstrated:



Asymptotic Relationships

☐ Show Logarithmic ☐ Extra Controls
f(n) = log(n) scale: 1

☑ Show Linear ☐ Extra Controls
f(n) = n scale: 1

☑ Show nLog(n) ☐ Extra Controls
f(n) = nlog(n) scale: 1

☑ Show Quadratic ☐ Extra Controls
$f(n) = n^2$ scale: 0.05

☐ Show Cubic ☐ Extra Controls
$f(n) = n^3$ scale: 1

☐ Show Quartic ☐ Extra Controls
$f(n) = n^4$ scale: 1

☐ Show 2^n
$f(n) = 2^n$ scale: 1

Size of n 63

Figure 1: $O(n^2)$: Yellow, $\Omega(n)$: Green and $\Theta(n\log n)$: Red)

Here, the $O(n^2)$ has been scaled by 0.05x in order to more clearly see how $f(n\log n)$ is bounded by both n^2 and n, but it also $\Theta(n\log n)$.

Question 3

Functions	Asymptotic Relationships
$f(n) = n^{½}$ and $g(n) = \log n$	$\lim_{->\infty} f(n)/g(n) \quad = n^{1/2} / \log n$ $= \infty$ $f = \omega(g), \Omega(g)$
$f(n) = 1500$ and $g(n) = 2$	$\lim_{->\infty} f(n)/g(n) \quad = 1500 / 2$ $= c$ $= 1$ $f \sim g, f = O(g), \Omega(g), \Theta(g)$
$f(n) = 800 \cdot 2^n$ and $g(n) = 3^n$	$\lim_{->\infty} f(n)/g(n) \quad = 800 \cdot 2^n / 3^n$ $= 800 \cdot e^{n\ln 2} / e^{n\ln 3}$ $= 0$ $f = o(g), O(g)$
$f(n) = 4^{n+13}$ and $g(n) = 2^{2n+2}$	$\lim_{->\infty} f(n)/g(n) \quad = 4^{n+13} / 2^{2n+2}$ $= 0$ $f = o(g), O(g)$
$f(n) = 9n \cdot \log n$ and $g(n) = n \cdot \log 9n$	$\lim_{->\infty} f(n)/g(n) \quad = 9n \log n / n \log 9n$ $= 9 \log n / \log 9n$ $= (\log n + \log 9) / 9 \log n$ $= \log n / 9 \log n$ $= 1/9$ $f \sim g, f = \Theta(1), O(1), \Omega(1)$

f(n) = n! and *g(n) = (n+1)!*	$\lim_{->\infty} f(n)/g(n)$ = n! / (n+1)! = 1/(n + 1) = 0 *f = o(g), O(g)*