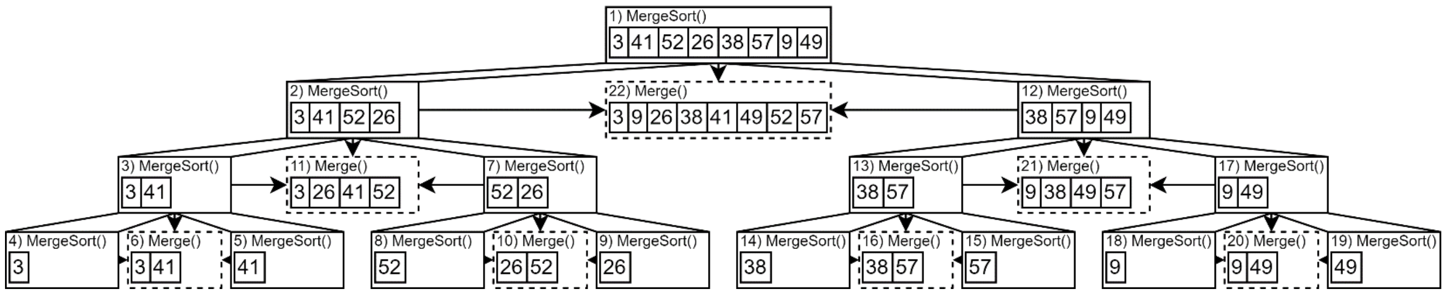
Jon Rippe
CSCE A311
Assignment 2

#1 Merge Sort Operation



#2 Linear Search

1. How many elements of the input sequence need to be checked on the average, assuming that the element being searched for is equally likely to be any element in the array?

   Average # of elements checked in an array with *n* elements would be $\frac{n+1}{2}$ (or about half) if there is an equal chance of the value being in any array index: $\frac{\sum_{i=1}^{n} i}{n} = \frac{n+1}{2} = \frac{1}{2}n + \frac{1}{2}$
2. How about in the worst case?

   Every element would have to be checked (*n*)
3. What are the average-case and worst-case running times of linear search in Theta notation?

   $\Theta(n)$ for both: *n* is the driving factor ($\frac{1}{2}n$ for average and n for worst).

#3 Binary Search Algorithm

Pseudocode

The following algorithm performs a binary search on array *A* and returns the index value *i* where the search value *v* is found or returns -1 if the search value *v* is not found.

```
Search($A$) for $v$

binSearch($A, p, q, v$)
    $i = A\left[p + \left\lfloor \frac{q-p}{2} \right\rfloor\right]$
    **if** $v = A[i]$
        **return** $i$
    **else if** $p = q$
        **return** -1
    **else if** $v < A[i]$
        **return** binSearch($A, p, i - 1, v$)
    **else**
        **return** binSearch($A, i + 1, q, v$)
```

Recurrence Relation & Iterative Substitution

$$T(n) = T\left(\frac{n}{2}\right) + 1$$
$$T\left(\frac{n}{2}\right) = T\left(\frac{n}{4}\right) + 1$$
$$T\left(\frac{n}{4}\right) = T\left(\frac{n}{8}\right) + 1$$

$$T(n) = T\left(\frac{n}{8}\right) + 1 + 1 + 1$$
$$T(n) = 1 + 1 + 1 + T\left(\frac{n}{8}\right)$$

Create Recurrence Relation & Solve for Θ

$$T(n) = i + T\left(\frac{n}{2^i}\right)$$
$$T(n) = i + T(1)$$
$$T(n) = \log_2 n + \Theta(1)$$

$$\frac{n}{2^i} = 1$$
$$n = 2^i$$
$$i = \log_2 n$$

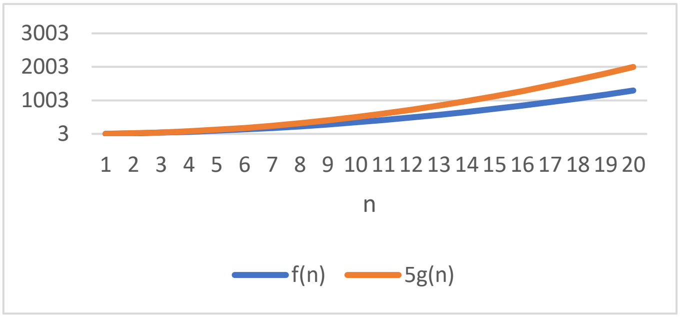
Worst Case Scenario (element not in array)

$T(n) = \Theta(\lg n)$ for the worst case scenario.
Based on the recurrence relation, the algorithm must search no more and no less than $\log_2 n$ elements to determine that the element does not exist in the array.

#5 Notation Definitions

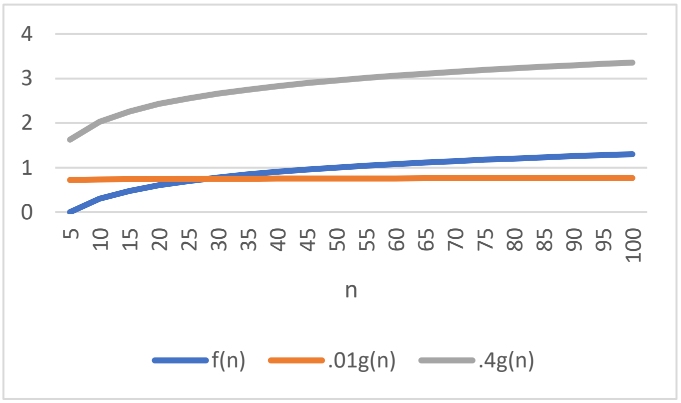
a) What is big-O for f(n) = 3n^2 + 5n - 3?

$$3n^2 + 5n - 3 = O(n^2)$$
$$f(n) = 3n^2 + 5n - 3$$
$$g(n) = n^2$$
$$0 \le 3n^2 + 5n - 3 \le cn^2$$
$$0 \le 3 + \frac{5}{n} - \frac{3}{n^2} \le c$$
$$n_0 = 1, \qquad c = 5$$



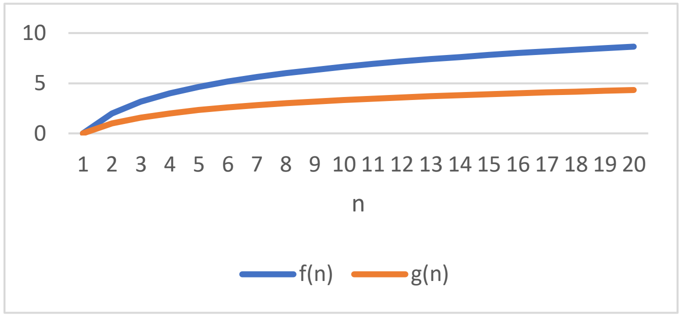
b) What is Θ for f(n) = log(n/5)?

$$\log\frac{n}{5} = \Theta(\lg n)$$
$$f(n) = \log\frac{n}{5}$$
$$g(n) = \log n$$
$$c_1 \lg n \le \log\frac{n}{5} \le c_2 \lg n$$
$$c_1 \lg n \le \log n - \log 5 \le c_2 \lg n$$
$$c_1 \lg n + \log 5 \le \log n \le c_2 \lg n + \log 5$$
$$n_0 = 30, \qquad c_1 = \frac{1}{100}, \qquad c_2 = \frac{2}{5}$$



c) What is Ω for f(n) = 2lg(n)?

$$2\lg n = \Omega(\lg n)$$
$$f(n) = 2\lg n$$
$$g(n) = \lg n$$
$$2\lg n \ge c\lg n$$
$$n_0 = 1, \qquad c = 1$$



#6 Brick Search Algorithm

The best way to find a single lightweight brick would be to perform a search similar to a binary search:

1. Divide bricks in half and weigh (removing a single brick if you have an odd number)
    - If both sides weigh the same, the brick you removed is the contaminated brick
    - If only one brick remains on each side, the lighter brick is the contaminated brick
2. Discard the heavier group and repeat the process until the brick is found

A minimum of 2 and a maximum of 5 weighings would be needed to find the contaminated brick:

Weigh #	Total Bricks	On Each Side	In Hand	Possible Find?
1	50	25	0	No
2	25	12	1	Yes (4% probability)
3	12	6	0	No
4	6	3	0	No
5	3	1	1	Yes (100% probability)

$$T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 1$$

Like the binary search algorithm, we throw out half the search area each time for a worst case $O(\lg n)$. Unlike the binary search, we only have a chance of a find if our current search area quantity is an odd number or we're down to our last 2 elements. Thus, the average case would be much closer to the worst case and the best case would vary depending on how many elements we start with.

#7 Merge Sort/Insertion Sort

a) Show that the n/k sublists, each of length k, can be sorted by insertion sort in O(nk) worst-case time.

There are $\frac{n}{k}$ quantity sublists and each sublist will be sorted in a worst case of $\mathrm{O}(k^2)$.

$$\frac{n}{k}k^2 = nk = \mathrm{O}(nk)$$

b) Show that the sublists can be merged in O(n lg(n/k)) worst-case time.

A pure merge sort will eventually reach the base case of *n* quantity sublists (each containing 1 element) and must eventually merge each sublist into a single list with *n* elements: $\mathrm{O}(n\lg n)$. However, at the base level of our modified merge sort, only $\frac{n}{k}$ quantity sublists (each containing *k* elements) must eventually be merged into a list with *n* elements:

$$\mathrm{O}\left(n\lg\frac{n}{k}\right)$$

c) $\mathrm{O}\left(nk + n\lg\frac{n}{k}\right) = \mathrm{O}(n\lg n)$

$\mathrm{O}(n\lg n) = \mathrm{O}(nk + n\lg n - n\lg k)$

$\mathrm{O}(n\lg n + n\lg k) = \mathrm{O}(nk + n\lg n)$

$n\lg n \geq n\lg k\,,\qquad n\lg n \geq nk$

$\lg n \geq \lg k\,,\qquad \lg n \geq k$

$n \geq k\,,\qquad \lg n \geq k$

$k \leq \lg n \leq n$

$k = \mathrm{O}(\lg n)$

d) *k* should be chosen based on actual runtime constants in the specific implementation of the algorithm.

#8 Iterative Substitution

Fill Values

$$T(n) = 3T\left(\frac{n}{2}\right) + n$$
$$T\left(\frac{n}{2}\right) = 3T\left(\frac{n}{4}\right) + \frac{n}{2}$$
$$T\left(\frac{n}{4}\right) = 3T\left(\frac{n}{8}\right) + \frac{n}{4}$$

Substitute & Simplify

$$T(n) = 3\left(3\left(3T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + \frac{n}{2}\right) + n$$
$$T(n) = n + 3\frac{n}{2} + 9\frac{n}{4} + 27T\left(\frac{n}{8}\right)$$

Create Recurrence Relation & solve for Θ

$$T(n) = n + \frac{3}{2}n + \frac{9}{4}n + \cdots + n\left(\frac{3}{2}\right)^{i-1} + 3^i T\left(\frac{n}{2^i}\right)$$
$$T(n) = n\sum_{k=0}^{i-1}\left(\frac{3}{2}\right)^k + 3^i T(1)$$
$$T(n) = n\sum_{k=0}^{\lg n - 1}\left(\frac{3}{2}\right)^k + 3^{\lg n}\Theta(1)$$
$$T(n) \le n\frac{\left(\frac{3}{2}\right)^{\lg n} - 1}{\frac{3}{2} - 1} + n^{\lg 3}\Theta(1)$$
$$T(n) \le 2nn^{\lg\frac{3}{2}} - 2n + n^{\lg 3}\Theta(1)$$
$$T(n) \le 2n^{\lg 3} - 2n + \Theta\left(n^{\lg 3}\right)$$

$$\frac{n}{2^i} = 1$$
$$n = 2^i$$
$$i = \lg n$$
$$\sum_{k=0}^{m} x^k = \frac{x^{m+1} - 1}{x - 1}$$

Determine O

$$n < n^{\lg 3}$$
$$O\left(n^{\lg 3}\right)$$

#9 Master Method

a) $T(n) = 4T\left(\frac{n}{2}\right) + n$

$a = 4, \qquad b = 2, \qquad f(n) = n$

1) $f(n) = O\left(n^{\log_b a - \varepsilon}\right)$

$n = O\left(n^{\lg 4 - \varepsilon}\right), \qquad \varepsilon = 2, \qquad \checkmark$

$n = O(n^1)$

$T(n) = \Theta\left(n^{\lg 4}\right) = \Theta(n^2)$

b) $T(n) = 4T\left(\frac{n}{2}\right) + n^2$

$a = 4, \qquad b = 2, \qquad f(n) = n^2$

1) $f(n) = O\left(n^{\log_b a - \varepsilon}\right)$

$n^2 = O\left(n^{\lg 4 - \varepsilon}\right), \qquad \varepsilon = C^+, \qquad ✗$

2) $f(n) = \Theta\left(n^{\log_b a}\right)$

$n^2 = \Theta\left(n^{\lg 4}\right) = \Theta(n^2), \qquad \checkmark$

$T(n) = \Theta\left(n^{\lg 4} \lg n\right) = \Theta(n^2 \lg n)$

c) $T(n) = 4T\left(\frac{n}{2}\right) + n^3$

$a = 4, \qquad b = 2, \qquad f(n) = n^3$

1) $f(n) = O\left(n^{\log_b a - \varepsilon}\right)$

$n^3 = O\left(n^{\lg 4 - \varepsilon}\right), \qquad \varepsilon = C^+, \qquad ✗$

2) $f(n) = \Theta\left(n^{\log_b a}\right)$

$n^3 = \Theta\left(n^{\lg 4}\right) = \Theta(n^2), \qquad ✗$

3) $f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$

$n^3 = \Omega\left(n^{\lg 4 + \varepsilon}\right), \qquad \varepsilon = 4, \qquad \checkmark$

$af\left(\frac{n}{b}\right) \leq cf(n)$

$4\left(\frac{n}{2}\right)^3 \leq cn^3 \rightarrow \frac{1}{2}n^3 \leq cn^3, \qquad c = \frac{1}{2}, \qquad \checkmark$

$T(n) = \Theta(f(n)) = \Theta(n^3)$