Collaborators:
Brennan Rieger,
Aftab Chitalwala,
Aaron Tucker,
George Wu,
Diane Yang,
Armi Behlum,
Nick Induni

1.

A	B	O	o	Ω	ω	Θ
$\log n$	$\log(\log n)$	yes	no	yes	no	yes
$\log(n!)$	$\log(n^n)$	yes	no	yes	no	yes
$\sqrt[7]{n}$	$(\log n)^2$	no	no	yes	yes	no
$n^2 2^n$	3^n	yes	yes	no	no	no
$(n^3)!$	n^n	no	no	yes	yes	no
$\frac{n^2}{\log n}$	$n \log(n^2)$	no	no	yes	yes	no
$(\log n)^{\log n}$	$4^{(\log n)(\log\log n)}$	yes	yes	no	no	no
$n + \log n$	$100n + \sqrt{n}$	yes	no	yes	no	yes

2. • Find (with proof) a function f_1 such that $f_1(2n)$ is $O(f_1(n))$.
- Suppose $f_1(n) = n$. Then $f_1(2n) = 2n$. Suppose $N = 1$, and $c = 3$. For all $n \geq 1$, $f_1(n) \leq 3 f_1(n)$ because it is true that $2n \leq 3n$ for all $n \geq 1$, so $f_1(2n)$ is $O(f_1(n))$.

• Find (with proof) a function f_2 such that $f_2(2n)$ is not $O(f_2(n))$.
- Suppose $f_2(n) = 2^n$. Then $f_2(2n) = 2^{2n} = 4^n$. Taking the limit $\lim_{n\to\infty} \frac{2^n}{4^n} = \lim_{n\to\infty} \left(\frac{1}{2}\right)^n = 0$, meaning that $f_2(2n)$ is $\omega(f_2(n))$, meaning it cannot be $O(f_2(n))$.

• Prove that if $f(n)$ is $O(g(n))$, and $g(n)$ is $O(h(n))$, then $f(n)$ is $O(h(n))$.
- For $f(n)$ to be $O(g(n))$, there exist by definition an N_1 and c_1 such that for $n \geq N_1$, $f(n) \leq c_1 g(n)$. Similarly, there exist an N_2 and c_2 such that for all $n \geq N_2$, $g(n) \leq c_2 h(n)$. We know that for $n \geq \max(N_1, N_2)$, $f(n) \leq c_1 g(n) \leq c_1 c_2 h(n)$, so for $c = c_1 c_2$ and $n \geq \max(N_1, N_2)$, $f(n) \leq c_1 c_2 h(n)$, so $f(n)$ is $O(h(n))$.

• Give a counterexample: If f is not $O(g)$, then g is $O(f)$.
- Suppose $f(n) = \begin{cases} 1, & n \text{ is even} \\ n, & n \text{ is odd} \end{cases}$ and $g(n) = \begin{cases} 1, & n \text{ is odd} \\ n, & n \text{ is even.} \end{cases}$
f is not $O(g)$: Suppose it is. Then there exist an N, c such that for $n \geq N$, $f(n) \leq c\,g(n)$. Once $n > c$ for odd n, $f(n)$ must be greater than $cg(n)$ because $n > c$, so the statement is false, and f is not $O(g)$. By symmetry, g is not $O(f)$.

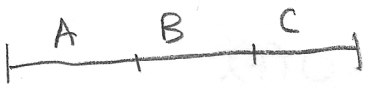
2. (continued)
- Give a proof: If f is $o(g)$, then f is $O(g)$.
  - If f is $o(g)$, then $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$. By definition of a limit, this implies that for all $c > 0$, there exists N such that $\left|\frac{f(n)}{g(n)}\right| < c$ whenever $n \geq N$, which can be rewritten as $f(n) < cg(n)$ whenever $n \geq N$ and $c > 0$ since $f(n)$ and $g(n)$ are positive, which means that $f(n)$ is $O(g(n))$.

3. Insertion Sort: $L[n]$ is at most 1 different from a $\Theta(T(n))$ function, so we only need to consider integer differences between runtimes for consecutive array sizes. We can consider each subsequent input as an additional element in the array, and for an array that was originally size k, an additional element at the end will require at least 1 (greater than all of the other elements) and at most k (less than all of the other elements) additional swaps, meaning that the times will grow at least $\Omega(n)$ (1 step for each addition) and at most $O(n^2)$ (k steps for each addition). For any $T(n)$ such that $T(n) = \Omega(n)$ and $T(n) = O(n^2)$, of the form n^g, each additional element must add k^{g-1} to the runtime, and it is possible to place a new element at the end such that it requires k^{g-1} swaps, since $1 \leq g \leq 2$.

4. Proof by induction that StoogeSort sorts correctly:
Base cases: For a list of 1 element, StoogeSort returns the element. For a list of 2 elements, StoogeSort returns either the list as is if the first element is less than the second, or swaps the two elements and returns the swapped list otherwise. Inductive hypothesis: If StoogeSort sorts correctly for lists of size 1, 2, ..., k then it is possible to prove that it sorts correctly for a list of size $k+1$. The algorithm itself is: StoogeSort: If the list is size 1, return it, ~~else if the list is size two, if the larger element is first, swap the elements and return the list, otherwise~~

else if the list is size 2
  if the larger element is first
    swap the elements and return the list
  else
    return the list

else
  find the size of $\frac{1}{3}$ of the list by doing $L(\text{end} - \text{start} + 1)/3]$.
  StoogeSort start to end $-\frac{1}{3}$
  StoogeSort start $+\frac{1}{3}$ to end
  StoogeSort start to end $-\frac{1}{3}$.



for a list of size $k+1$, it will sort $\frac{2}{3}$ of $k+1$ correctly because we assumed that all lists of size less than $k+1$ sort correctly. Taking each region to be $\frac{1}{3}$ of the list as drawn above, everything in A < everything in B after the first step. Everything in A < C after the second step, and everything in A < B < C after the third step.

4. (continued) A recurrence for StoogeSort is $T(n) = 3T(\lceil \frac{2n}{3} \rceil) + 1$, since there are 3 subproblems of size $\lceil \frac{2n}{3} \rceil$. $T(n)$ is monotonically nondecreasing, ~~$T(1)$ is [illegible]~~ Proof by induction: Base case: $T(1)=1$, $T(2)$ is more complex, since we need to compare the two elements and switch them if necessary.
If $T(1), T(2) \ldots T(k)$ are nondecreasing, then $T(k+1) \geq T(k)$. $T(k+1) = 3T(\lceil \frac{2k+1}{3} \rceil) + 1$. $T(\lceil \frac{2k+1}{3} \rceil) \geq T(\lceil \frac{2(k)}{3} \rceil)$ if our inductive hypothesis holds, so $T(k+1) \geq T(k)$.
$T(n) \leq 3T(\frac{2n}{3} + 1) + c(n)$. Define $S(n) = T(n+a)$. Then we want $S(n) \leq 3S(\frac{2n}{3}) + O(n)$, i.e. $T(n+a) \leq 3T(\frac{2n}{3} + a) + O(n)$. We know that $T(n+a) \leq 3T(\frac{2(n+a)}{3} + 1) + O(n)$ by the given recurrence. Solving for a gives $a=3$, so we have a Master Theorem type recurrence for $S(n)$. By a similar proof, we can use the Master Theorem even if we have non-integral arguments for $\frac{2n}{3}$. The base case is $n=1$, in which case $S(n) = \Theta(1)$. Applying the Master Theorem, we have $S(n) = \Theta(n^{\log_{1.5} 3}) = \Theta(n^{\frac{\log 3}{\log 1.5}})$.

5. • $T(1)=1$, $T(n) = T(n-1) + 3n - 3$. Guess: $T(n) = 1 + \frac{3n^2 - 3n}{2}$.
Base case: $T(1) = 1 + \frac{3(1)^2 - 3(1)}{2} = 1$.
Inductive step: Assume $T(n-1) = 1 + \frac{3(n-1)^2 - 3(n-1)}{2} = 1 + \frac{3n^2 - 9n}{2}$.
We also know based on the recurrence that $T(n) = 4 + \frac{3n^2 - 9n}{2} + 3n - 3 = 1 + \frac{3n^2 - 3n}{2}$. But our guess was that $T(n) = 1 + \frac{3n^2 - 3n}{2}$, so our guess is correct and solves the recurrence.

• $T(1)=1$, $T(n) = 2T(n-1) + 2n - 1$. Guess: $T(n) = 1 + 2(-2 + 3 \cdot 2^{n-1} - n)$.
Base case: $T(1) = 1 + 2(-2 + 3 \cdot 2^0 - 1) = 1$. Inductive step: Assume $T(k-1) = 1 + 2(-1 + 3 \cdot 2^{k-2} - k)$. Then $T(k) = 2T(k-1) + 2k - 1 = 2 + 4(-1 + 3 \cdot 2^{k-2} - k) + 2k - 1 = 2 + 4 - 1 + 2k - 4k + 12 \cdot 2^{k-2} = 1 + 2(-2 + 6 \cdot 2^{k-2} - k) = 1 + 2(-2 + 3 \cdot 2^{k-1} - k)$, But our guess was that $T(k) = 1 + 2(-2 + 3 \cdot 2^{k-1} - k)$, so our guess is correct and solves the recurrence.

6. • $T(n) = 5T(\frac{n}{3}) + n^3$. Use the Master Theorem. $a=5$, $b=3$, $c=1$, $k=3$. $a < b^k$, so $T(n)$ is $\Theta(n^3)$.

• $T(n) = 25T(\frac{n}{4}) + n^2$. Use the Master Theorem. $a=25$, $b=4$, $c=1$, $k=2$. $a > b^k$, so $T(n)$ is $\Theta(n^{\log_4 25})$.

• $T(n) = 8T(\frac{n}{2}) + n^3$. Use the Master Theorem. $a=8$, $b=2$, $c=1$, $k=3$. $a = b^k$, so $T(n)$ is $\Theta(n^3 \log n)$.

• $T(n) = T(\sqrt[4]{n}) + 1$. We can define a new function $S(\log n) = T(n)$. Set $x = \log n$, so $S(x) = T(2^x) = T(2^{\frac{x}{4}}) + 1$. $T(2^{\frac{x}{4}}) = S(\frac{x}{4})$, so $S(x) = S(\frac{x}{4}) + 1$. By Master Theorem, $S(x)$ is $\Theta(4 \log x)$, and $T(n)$ is $\Theta(\log \log n)$.

7. Guess: $-2^{\lceil \log_2 n \rceil} + n\lceil \log_2 n \rceil + 1$.

Consider the case where n is even: In that case, we can simplify the recurrence to $T(n) = 2T(\frac{n}{2}) + n - 1$. Proof By Induction: Base Case: $T(2) = 1$. $T(\frac{n}{2}) = -2^{\lceil \log_2 \frac{n}{2} \rceil} + \frac{n}{2}\lceil \log_2 \frac{n}{2} \rceil + 1$. $T(n) = 2\left(-2^{\lceil \log_2 \frac{n}{2} \rceil} + \frac{n}{2}\lceil \log_2 \frac{n}{2} \rceil + 1\right) = -2^{\lceil \log_2 n \rceil} + n\lceil \log_2 n \rceil + 1$, which is what $T(n)$ gives us, meaning the solution is correct for even n.

Consider the case where n is odd: $\lceil \frac{n}{2} \rceil = \frac{n+1}{2}$, and $\lfloor \frac{n}{2} \rfloor = \frac{n-1}{2}$.

Consider the subcase where $\lceil \log_2 (\frac{n+1}{2}) \rceil = \lceil \log_2 (\frac{n-1}{2}) \rceil = \lceil \log_2 \frac{n}{2} \rceil$.

$T(\frac{n+1}{2}) = -2^{\lceil \log_2 \frac{n+1}{2} \rceil} + \frac{n+1}{2}\lceil \log_2 \frac{n+1}{2} \rceil + 1$.

$T(\frac{n-1}{2}) = -2^{\lceil \log_2 \frac{n+1}{2} \rceil} + \frac{n-1}{2}\lceil \log_2 \frac{n+1}{2} \rceil + 1$.

Base case: $T(1) = 0$.

$T(n) = 2(-2^{\lceil \log_2 \frac{n+1}{2} \rceil}) + \frac{n+1}{2}\lceil \log_2 \frac{n+1}{2} \rceil + \frac{n-1}{2}\lceil \log_2 \frac{n+1}{2} \rceil + 2 + n - 1$.

$= -2^{\lceil \log_2 (n+1) \rceil} + n\lceil \log_2 (n+1) \rceil - n + n + 1$

$= -2^{\lceil \log_2 n \rceil} + n\lceil \log_2 n \rceil + 1$, meaning the solution is correct for this subcase.

Consider the subcase where $\lceil \log_2 (\frac{n-1}{2}) \rceil + 1 = \lceil \log_2 (\frac{n+1}{2}) \rceil$. $T(\frac{n+1}{2})$ is the same. $T(\frac{n-1}{2}) = -2^{\lceil \log_2 (\frac{n+1}{2}) - 1 \rceil} + \frac{n-1}{2}\lceil \log_2 \frac{n+1}{2} - 1 \rceil + 1$.

$T(n) = -2^{\lceil \log_2 \frac{n+1}{2} \rceil} + \frac{n+1}{2}\lceil \log_2 \frac{n+1}{2} \rceil - 2^{\lceil \log_2 \frac{2n+1}{2} - 1 \rceil} + \frac{n-1}{2}\lceil \log_2 \frac{n+1}{2} - 1 \rceil + 2 + n - 1$,

which comes out to $-2^{\lceil \log_2 n \rceil} + n\lceil \log_2 n \rceil + 1$, meaning the solution is correct for this subcase and all subcases.