CS3000: Algorithms & Data — Summer 2025 — Laney Strange

Exam 1 Practice Problems

- These practice problems are to help you prepare for Exam 1. We'll release the solutions on Tuesday (May 20) so you can go over them and ask questions in your recitation.
- Exam 1 takes places during lecture on May 22, 2025.
- The exam will be due at the end of lecture (1:20pm). You'll hand it in on paper, but we'll scan it later for grading on gradescope.
- You may bring one 8.5x11-inch paper as a cheat sheet, with anything written or typed on it (one side only). You will submit this cheat sheet along with your exam, and you will not be permitted to use any other materials or notes during the exams.
- If you have a DAS accommodation for exams, please arrange to take the exam at their center. Make sure you schedule that time ASAP if you haven't yet!

Problem 1. *Practice - Loop Invariant*

Below is the Pseudocode for a procuedre called BUILD-MAX-HEAP, which turns an sorted array into a max-heap. You may assume that MAX-HEAPIFY works correctly to restore the heap properties on a structure where everything, except possibly the root at position i, is already a max-heap.

Prove that BUILD-MAX-HEAP correctly builds a max-heap by showing the following loop invariant: At the start of each iteration of the for loop of lines 2-3, each node $i+1, i+2, ..., n$ is the root of a max heap.

BUILD-MAX-HEAP(A, n)

1 $A.heapsize = n$
2 **for** $i = \lfloor n/2 \rfloor$ **downto** 1
3 MAX-HEAPIFY(A, i)

- Initialization
- Maintenance
- Termination

Solution:

- **Initialization**. Prior to the first iteration of the loop $i = \lfloor n/2 \rfloor$. Each node $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, ..., n$ is a leaf and is thus the root of a trivial max-heap.
- **Maintenance**. To see that each iteration maintains the loop invariant, observe that the children of node i are numbered higher than i. By the loop invariant, both children are roots of max-heaps. This is precisely the condition required for the call to MAX-HEAPIFY(A, i) to make node i a max-heap root. Moreover, the MAX-HEAPIFY call preserves the property that nodes $i+1, i+2, .., n$ are all roots of max-heaps. Decrementing i in the for loop update reestablishes the loop invariant for the next iteration.
- **Termination**. At termination, $i = 0$. By the loop invariant, each node $1, 2, ..., n$ is the root of a max-heap. In particular, node 1 is the root of a max-heap, showing that our procedure works correctly.

Problem 2. *Practice - Binary Search*

Suppose you are given a sorted array A of non-zero integers and you want to determine whether there exist two distinct indices $i \neq j$ such that $A[i] = -A[j]$. Give pseudocode for an algorithm that solves this problem, calling Binary Search as a subroutine. Your algorithm should return indices (i, j) if they exist (any pair can be returned if there are multiple such indices) or NIL if there is no such pair.

Solution:

```
OPPOSITE(A, n)
1  for i = 1 to n − 1
2        j = BINARYSEARCH(A, n, −1 · A[i])
3        if j ≠ NIL and j ≠ i
4              return (i, j)
5  return NIL
```

Problem 3. *Practice - Bounds*

Identify whether $f(n) = O(g(n))$, $g(n) = O(f(n))$, or both.

(a) $f(n) = 2n$, $g(n) = 4n$
Solution: Both: $g(n) = O(f(n))$ and $f(n) = O(g(n))$

(b) $f(n) = \sqrt{n}$, $g(n) = \lg n$
Solution: $g(n) = O(f(n))$

Problem 4. *Practice - Growth of Functions*

Order the following functions from slowest growing to fastest growing:

$$\lg n, \quad n^3, \quad n \lg n, \quad n!, \quad n^{500}, \quad 5^n, \quad n, \quad \lg \lg n$$

Solution:

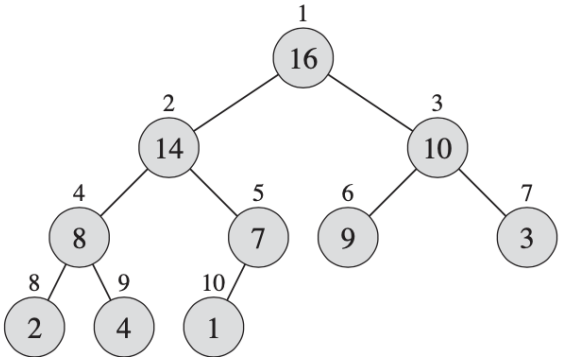
$$\lg \lg n \ll \lg n \ll n \ll n \lg n \ll n^3 \ll n^{500} \ll 5^n \ll n!$$

Problem 5. *Practice - Heaps*

(a) Draw the heap represented by the array $\langle 16, 14, 10, 8, 7, 9, 3, 2, 4, 1 \rangle$ and state whether it is a valid heap.

Solution:

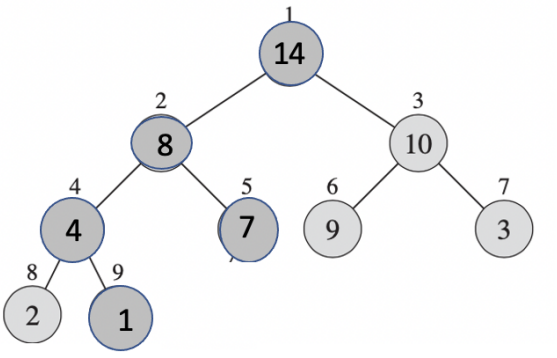
Figure 1: The given array represents the following valid max-heap.



(b) Given the heap in the previous portion, draw what it would look like after the root is removed, the final element is put into the root, and the structure is re-heapified.

Solution:

Figure 2: On removing the max element (the root, 16), we take the last leaf node, 1, and replace 16 with that and then propagate down the changing positions of the nodes.



Problem 6. *Practice - Mystery Run-Time*

Given the below pseudocode, give a tight bound.

```
MYSTERY(A, n, B, m)
1  result = 0
2  for i = 1 to n
3        for j = 1 to m
4              result = result + A[i] · B[j]
5  return result
```

Solution:

Line	Cost	Times
1	c_1	1
2	c_2	$n+1$
3	c_3	$(n)(m+1)$
4	c_4	nm
5	c_5	1

Runtime is:

$$c_1 + c_2\,(n+1) + c_3\,(n \cdot m + n) + c_4\,(nm) + 1$$

The tight bound is:

$$\Theta(nm)$$

Problem 7. *Practice - Loop Invariant*

Below is the pseudocode for Insertion Sort.

```
INSERTIONSORT(A, n)
1  for i = 2 to n
2      key = A[i]
3      j = i − 1
4      while j > 0 and A[j] > key
5          A[j + 1] = A[j]
6          j = j − 1
7      A[j + 1] = key
```

(a) Prove the correctness of Insertion Sort by showing the following loop invariant: At the start of iteration i of the for loop of lines 1-7, the subarray $A[1..i-1]$ is in sorted order.

(For this proof, you can assume that the inner **while** loop correctly finds where *key* should be placed among the already-sorted subarray, and places it there in line 7.)

- Initialization:

- Maintenance:

- Termination:

Solution:

- *Initialization*: Before the first iteration begins, we have $i = 2$. The subarray $A[1..i-1]$ consists of just the single element $A[1]$ which is trivially in sorted order.

- *Maintenance:* At the start of each iteration, we have that $A[1..i-1]$ is in sorted order. We trust that the while loop of lines 4-6 finds the correct location for *key*, somewhere from 1 to i, and places *key* in its correct sorted location in line 7. Therefore, $A[1..i]$ is in sorted order. Finally, we increment i, maintaining the loop invariant.

- *Termination:* When the loop terminates, we have $i = n + 1$. By the loop invariant, we now have $A[1..n]$ is in sorted order.

Problem 8. *Practice - Best Case Run-Time*

Consider the pseudocode below for a Mystery Function that calls Linear Search as a subroutine. For simplicity throughout this problem, you can assume n will always be even.

```
MYSTERY-2(A, n)
1  for i = 1 to ⌊n/2⌋
2        if A[i] mod 2 == 0
3              found = LINEAR-SEARCH(A, n, 2 · A[i])
4              if found ≠ NIL
5                    return FALSE
6  return TRUE
```

(a) Complete the table below capturing the best-case run-time of this algorithm.

line	cost	times
1		
2		
3		
4		
5		
6		

Solution:

line	cost	times
1	c_1	$\frac{n}{2}+1$
2	c_2	$\frac{n}{2}$
3	c_3	0
4	c_4	0
5	c_5	0
6	c_6	1

(b) Formally show the run-time $T(n)$. Show all of your work and don't skip any steps in the arithmetic.

$$\begin{aligned} T(n) &= c_1 \cdot \left(\frac{n}{2}+1\right) + c_2 \cdot \frac{n}{2} + c_6 \\ &= \frac{c_1 n}{2} + c_1 + \frac{c_2 n}{2} + c_6 \\ &= \frac{c_1 + c_2}{2} n + (c_1 + c_6) \end{aligned}$$

(c) Give a tight bound on the run-time.

$T(n) = \Theta(n)$

(d) Give an array example for A that would result in the best-case run-time.

Solution:
$A = \langle 3, 9, 7, 5 \rangle$

Problem 9. *Practice - Sorting*

Consider the problem of sorting an Array $A[1, ..., n]$. In general, sorting algorithms are known to have a lower bound of $\Omega(n \lg n)$. For this problem, we're going to try to beat that bound in certain cases.

(a) Give an efficient algorithm for sorting a Boolean array $B[1, ..., n]$ that is more efficient than $O(n \lg n)$. Assume that you want FALSE to come before TRUE.

Solution:

```
BOOLEAN-SORT(B, n)
 1  number_of_false = 0
 2  for i = 1 to n
 3       if B[i] == FALSE
 4            number_of_false = number_of_false + 1
 5  for i = 1 to n
 6       if number_of_false > 0
 7            B[i] = FALSE
 8            number_of_false = number_of_false − 1
 9       else
10            B[i] = TRUE
```

Alternate solution, which has slightly better constants but the same tight bound:

```
BOOLEAN-SORT-2(B, n)
1  falseCount = 0
2  for i = 1 to n
3       if B[i] == FALSE
4            falseCount = falseCount + 1
5  for i = 1 to falseCount
6       B[i] = FALSE
7  for i = falseCount + 1 to n
8       B[i] = TRUE
```

(b) Give a tight bound on the worst-case run-time for your algorithm.

Solution:

$$\Theta(n) = n$$

The complexity of this algorithm is linear.

Problem 10. *Practice - Rotation*

Prior to being passed to your function, an array of sorted integers $A = \langle A_1, A_2, \ldots, A_n \rangle$ is possibly rotated at an unknown pivot index $1 \leq k < n$, such that the resulting array is $\langle A_{k+1}, A_{k+2}, \ldots, A_n, A_1, A_2, \ldots, A_k \rangle$. For example, if $A = \langle 0, 1, 2, 4, 5, 6, 7 \rangle$ is rotated and becomes $\langle 4, 5, 6, 7, 0, 1, 2 \rangle$, then the pivot index $k = 3$.

Give pseudocode for an algorithm that runs in linear time which calculates the value of k given a rotated input array. Describe your approach in a few sentences and write pseudocode.

Solution:

We know that the array is already sorted and is just rotated about an index. Thus, if we can find the index of the minimum number in the array, we will know where A_1 is. Then, we can do some math to figure out what k is based on the length of the array.

```
FIND-PIVOT-INDEX(A, n)
1  min_number = A[1]
2  index = 1
3  for i = 2 to n
4       if A[i] < min_number
5            min_number = A[i]
6            index = i
7  return n − index + 1
```

Problem 11. *Practice - Insertion Sort*

Given the following array $A = \langle 1, 10, 2, 3, 15, 4 \rangle$ write out what the array would look like after each iteration of the for loop of insertion sort.

Solution:

$$\langle 1, 10, 2, 3, 15, 4 \rangle$$
$$\langle 1, 2, 10, 3, 15, 4 \rangle$$
$$\langle 1, 2, 3, 10, 15, 4 \rangle$$
$$\langle 1, 2, 3, 10, 15, 4 \rangle$$
$$\langle 1, 2, 3, 4, 10, 15 \rangle$$

Problem 12. *Practice - Recursive Sequences*

(a) Given that $a_k = 2k + 1$, find a_0, a_1, ..., a_5 and determine the recurrence relation a_n:

Solution:

$$a_0 = 1$$
$$a_1 = 3$$
$$a_2 = 5$$
$$a_3 = 7$$
$$a_4 = 9$$
$$a_5 = 11$$

Recurrence: $a_n = a_{n-1} + 2$, base case $a_0 = 1$.

(b) Given the recurrence relation $a_n = a_{n-1} + 4$ with base case $a_1 = 2$, find a_2, a_3, ..., a_5 and find the closed form for a_k.

Solution:

$$a_2 = 6$$
$$a_3 = 10$$
$$a_4 = 14$$
$$a_5 = 18$$

Closed form: $a_k = 2 + 4(k - 1)$, for all $k \geq 1$.

(c) Given the recurrence relation $a_n = 2a_{n-1} - 1$ with the base case $a_1 = 3$, find the terms a_2, a_3, ..., a_5 and a closed form for a_k.

Solution:

$$a_2 = 5$$
$$a_3 = 9$$
$$a_4 = 17$$
$$a_5 = 33$$

Closed form: $a_k = 2^k + 1$, for all $k \geq 1$.

(d) Given the following sequence, determine the recurrence relation a_n:

$$
\begin{aligned}
a_1 &= 20 \\
a_2 &= -10 \\
a_3 &= 5 \\
a_4 &= -2.5 \\
a_5 &= 1.25
\end{aligned}
$$

Solution:

$a_n = -\frac{a_{n-1}}{2}$, base case $a_1 = 20$.

(e) Given the following sequence, determine the reccurence relation a_n:

$$
\begin{aligned}
a_1 &= 4 \\
a_2 &= 11 \\
a_3 &= 25 \\
a_4 &= 53 \\
a_5 &= 109
\end{aligned}
$$

Solution:

$a_n = 2a_{n-1} + 3$, base case $a_1 = 4$.

Problem 13. *Practice - Solve a Recurrence*

(a) Use the iteration method to solve the recurrence given by $T(n) = T(n-1) + n + c$ with base case $T(1) = 1$

Solution:

$$\begin{aligned}
T(n) &= T(n-1) + n + c && \textit{(first iteration)} \\
T(n-1) &= T(n-2) + (n-1) + c \\
T(n) &= T(n-2) + 2n + 2c - 1 && \textit{(second iteration)} \\
T(n-2) &= T(n-3) + (n-2) + c \\
T(n) &= T(n-3) + 3n + 3c - 3 && \textit{(third iteration)} \\
T(n-3) &= T(n-4) + (n-3) + c \\
T(n) &= T(n-4) + 4n + 4c - 6 && \textit{(fourth iteration)}
\end{aligned}$$

Now we've established a pattern! On the kth iteration, we have:

$$T(n) = T(n-k) + kn + kc - \frac{(k)(k-1)}{2}$$

To get the constant term you can look at each line and notice the following pattern:

$$T(n) = T(n-k) + kn + kc - 1 - 2 - 3 - ...(k-1)$$

This can be written as:

$$T(n) = T(n-k) + kn + kc - \sum_{i=1}^{k-1} i$$

We know $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ so plugging in our value for k we get:

$$T(n) = T(n-k) + kn + kc - \frac{(k-1)(k-1+1)}{2} = T(n-k) + kn + kc - \frac{(k)(k-1)}{2}$$

Choose a value of k to get us to the base case $T(1)$. We choose $k = (n-1)$.

$$\begin{aligned}
T(n) &= T(n-(n-1)) + (n-1)n + (n-1)c + \frac{(n-1)(n-2)}{2} \\
&= T(1) + n^2 - n + cn - c + \frac{1}{2}n^2 - \frac{1}{2}3n + \frac{2}{2} \\
&= (1 + \frac{1}{2})n^2 + (c - \frac{3}{2} - 1)n + 2 - c
\end{aligned}$$

(b) Give a tight bound on the recurrence.

Solution:

$$T(n) = \Theta(n^2)$$

Problem 14. *Practice - Solve a Recurrence*

(a) Use the iteration method to determine the closed form of the recurrence given by $T(n) = n^3 + 2T\left(\frac{n}{2}\right)$ with the base case $T(1) = 1$.

Solution:

$$T(n) = n^3 + 2T\left(\frac{n}{2}\right) \tag{1}$$
$$T\left(\frac{n}{2}\right) = (\frac{n}{2})^3 + 2T\left(\frac{n}{4}\right) \tag{2}$$
$$T\left(\frac{n}{4}\right) = (\frac{n}{4})^3 + 2T\left(\frac{n}{8}\right) \tag{3}$$
$$T\left(\frac{n}{8}\right) = (\frac{n}{8})^3 + 2T\left(\frac{n}{16}\right) \tag{4}$$

Plugging Equation 2 into Equation 1 we get:

$$T(n) = n^3 + 2\left[\left(\frac{n}{2}\right)^3 + 2T\left(\frac{n}{4}\right)\right] = n^3 + \frac{n^3}{2^2} + 4T\left(\frac{n}{4}\right) \tag{5}$$

Next plug Equation 3 into 5:

$$T(n) = n^3 + \frac{n^3}{2^2} + 4\left[\left(\frac{n}{4}\right)^3 + 2T\left(\frac{n}{8}\right)\right] = n^3 + \frac{n^3}{2^2} + \frac{n^3}{4^2} + 8T\left(\frac{n}{8}\right) \tag{6}$$

Now plug Equation 4 into 6:

$$\begin{aligned} T(n) &= n^3 + \frac{n^3}{2^2} + \frac{n^3}{4^2} + 8\left[\left(\frac{n}{8}\right)^3 + 2T\left(\frac{n}{16}\right)\right] \\ &= n^3 + \frac{n^3}{2^2} + \frac{n^3}{4^2} + \frac{n^3}{8^2} + 16T\left(\frac{n}{16}\right) \\ &= \frac{n^3}{4^0} + \frac{n^3}{4^1} + \frac{n^3}{4^2} + \frac{n^3}{4^3} + 2^4T\left(\frac{n}{2^4}\right) \end{aligned}$$

Now we see a pattern!

$$T(n) = n^3 \sum_{i=0}^{k-1} \left(\frac{1}{4}\right)^i + 2^kT\left(\frac{n}{2^k}\right) \tag{7}$$

(b) Give a bound on the recurrence. You may find it helpful to remember that any finite geometric sequence with $r \neq 1$ sums to a constant as shown in the below equation

$$\sum_{k=0}^{n-1} ar^k = \begin{cases} a\left(\frac{1-r^n}{1-r}\right) & r \neq 1 \\ an & otherwise \end{cases}$$

Solution:

We can simplify Equation 7 a little further by noting that we are summing a geometric series in the first term. We could calculate exactly what it is, but as mentioned in the problem hint, it's going to be some constant for each k so we can so we can bound that term. This gives:

$$T(n) = c'n^3 + 2^k T\left(\frac{n}{2^k}\right)$$

Now we need to figure out how to use the base case, $T(1)$. To get this we can say:

$$\frac{n}{2^k} = 1 \implies n = 2^k \implies k = \lg n$$

Substituting this in we get:

$$T(n) = c'n^3 + 2^{\lg n} T\left(\frac{n}{2^{\lg n}}\right) = c'n^3 + nT(1) = c'n^3 + n$$

We can drop constants and lower-order terms to get the tight bound: $T(n) = \Theta\left(n^3\right)$