Announcements

- Midterm 2: April 16th, it will cover Topics, 4, 5 (step-counting and recurrence relations), 6, and 7.
- SCS response rate is currently at: 42.57%
- Programming Project 4 is ongoing, due April 21

Topic 5: Review

By: Professor Lynam

Step Counting

```
int size = array.length;
for (int i = 0; i < size - 1; i++):
    for (int j = 0; j < size - i - 1; j++):
        if (array[j] > array[j + 1]):
            int temp = array[j]
            array[j] = array[j + 1]
            array[j + 1] = temp
```

- Outside for loops: 3
- Outer for loop: 4
- Inner for loop: 13
- Now… How to combine the for loops?

```
int size = array.length, o++ # variable assignment
o++ # outer for loop initialization
for (int i = 0; i < size - 1; i++):
        o++ # outer for loop true conditional evaluation
        o++ # inner for loop initialization
        for (int j = 0; j < size - i - 1; j++):
                o++ # inner for loop true conditional evaluation
                if (array[j] > array[j + 1]): (o++ * 6)
                        int temp = array[j] (o++)
                        array[j] = array[j + 1] (o++ * 2)
                        array[j + 1] = temp (o++ * 2)
                o++ # inner for loop incremenation
        o++; # inner for loop false conditional evaluation
        o++; # outer loop incrementation
o++; # outer for loop false conditional evaluation
```

Step Counting

```
int size = array.length;
for (int i = 0; i < size - 1; i++):
    for (int j = 0; j < size - i - 1; j++):
        if (array[j] > array[j + 1]):
            int temp = array[j]
            array[j] = array[j + 1]
            array[j + 1] = temp
```

- Outside for loops: 3, Outer for loop: 4, Inner for loop: 13
- The inner for loop goes from 0 to size (n) – i – 2 (n – i – 1 total iterations) where i starts at 0 and goes to n – 2 (n – 1 total iterations).
- We have to separate out the outer loop's iteration from the inner loop, using a summation for the inner loop. How many times will the inner loop run?
- $\sum_{i=0}^{n-2} n - i - 1$ is the summation for the inner loop. This is the total number of times it will run, so we just add it to the outer loop operations instead of multiplying it

Step Counting

```
int size = array.length;
for (int i = 0; i < size - 1; i++):
    for (int j = 0; j < size - i - 1; j++):
        if (array[j] > array[j + 1]):
            int temp = array[j]
            array[j] = array[j + 1]
            array[j + 1] = temp
```

- Outside for loops: 3, Outer for loop: 4, Inner for loop: 13, total number of inner loop iterations: $\sum_{i=0}^{n-2} n - i - 1$, total number of outer loop iterations: 4(n – 1).
- Combining it all together, we get: 3 + 4(n – 1) + $13 * \sum_{i=0}^{n-2} n - i - 1$.

Recurrence Relations

- Reminder of the process:

1. Determine the work required for the base and general cases of the given recursive algorithm.
2. Generate a few more equivalent recurrence for the work required in the general case.
3. Find the pattern within those expressions.
4. Create an equivalent closed-form (non-recurrence) expression in terms of the instance characteristic(s).
5. Prove that your closed-form expression is correct.
6. Determine the order of the algorithm.

Recurrence Relations

- Find the closed-form equivalent expression for the recurrence relation R(n) = R(n – 2) + k, where R(0) = c and n is an even integer. You do not need to provide a proof that your answer is correct.
- Which steps were you asked to perform for this question?
- 2-4. So let's start with 2! Generate 2 more equivalent recurrences.
- R(n) = R(n-4) + 2k
- R(n) = R(n-6) + 3k. Now step 3. What's the general form of this?
- R(n) = R(n – 2 * a) + a * k. Now step 4. How do we find what "a" is in terms of n?
- Since R(0) = c, let's find where n – 2 * a = 0. a = n/2, or n = 2 * a.
- Substitute that into our general expression: R(n) = R(0) + n/2*k, or R(n) = k * n / 2 + c

Recurrence Relations

- Prove that the closed-form of $T(n) = 2 * T\left(\frac{n}{2}\right) + n,$ where T(1) = c and n is a power of 2, is $T(n) = c * n + \text{ n} * \log_2 n.$
- Which step in the process is this? Step 5! I've already given a proposed closed-form expression, so no point in Finding the Pattern (which *cannot prove* anything)
- We need to do an inductive proof to show the closed-form expression is correct. We'll go through it, step-by-step.

Recurrence Relations

- **Conjecture**: The recurrence relation $T(n) = 2 * T\left(\frac{n}{2}\right) + n,$ where T(1) = c and n is a power of 2 is equal to the closed-form expression T(n) = c * n + n * $\log_2 n$.
- We will show this using weak induction. **Base Case**: T(1) = c for the recurrence relation (given), and T(1) = c * 1 + 1 * $\log_2 1$ = c for the closed-form expression.
- **Inductive Case**: We will define n to be 2^i, since n is a power of 2. So our recurrence relation becomes T(2^i) = 2 (T(2^{i-1})+ 2^i, and our closed-form expression becomes T(2^i)=c* 2^i + $\log_2 2^i$ = c * 2^i + i * 2^i. Our inductive hypothesis is that the conjecture holds for some value of i. We must show that it holds for i+1.
- T(2^{i+1})=2 * T(2^i) + 2^{i+1} starting with our recurrence relation and then substituting in our closed-form expression via the IH:
- T(2^{i+1})=2 * (c * 2^i + $\log_2 2^i$)+ 2^{i+1} = c * 2^{i+1}+ i * 2^{i+1} + 2^{i+1}. Take out the i + 1 from the expression: = c * 2^{i+1}+ (i+1)* 2^{i+1}. QED.

Step Counting Practice

```
int a = array_1.length;
int b = array_2.length;
int c = array_3.length;
int sum = 0;
for (int i = 0; i < a; i++):
        for (int j = 0; j < b; j++):
                for (int z = 0; z < c – 1; z++):
                        sum = array_1[i] + array_2[j] + array_3[z];
```

- Give this one a shot!
- Outside the for loops: 6
- First for loop: 4
- Second for loop: 4
- Third for loop: 11
- First for loop goes from 0 to a – 1 (a iterations), second from 0 to b – 1 (b iterations), third from 0 to c -2 (c – 1 iterations).
- Answer:
- 6 + a * (b * (11 * (c – 1) + 4) + 4)

Recurrence Relation Practice

- Prove that for the recurrence relation R(n) = R(n – 2) + k, where R(0) = c and n is an even integer, the equivalent closed-form expression is: R(n) = k * n / 2 + c
- Conjecture: The equivalent closed-form expression for the recurrence relation above is: R(n) = k * n / 2 + c
- **Base Case**: R(0) = c (given) and R(0) = k * 0 / 2 + c = c.
- **Inductive Case:** Assume conjecture holds for some value of n. Must show it holds for n + 2 (since n must be an even integer). R(n+2) = R(n) + k. By IH, R(n+2) = k * n / 2 + c + k = (k * n + 2k) / 2 +c = k/2 (n +2) + c. QED.

Sorting Review

- Can selection sort be made stable?
- I said “yes” in the slides, just take the first ‘smallest’ element you see… but I think this is a mistake. The swapping nature of selection sort is inherently unstable. Why?
- Take: $5_1\ 5_2\ 5_3\ 1$, and try to sort it with selection sort
- $1\ 5_2\ 5_3\ 5_1$would be the first swap… and there wouldn’t be any more swaps
- The issue lies with swapping non-adjacent elements, not the selection of the smallest element

Sorting Review

- Is there another way we could make selection sort stable?
- We could “insert” the smallest element to the beginning of the list, which would keep us from swapping anything… But this would involve “shifting” the entire array over with every insertion! Unless…
- We used Linked Lists, for which insertion of the smallest element would be constant time (just changing a few pointers)
- So, is selection stable, or could easily be made to be stable?
- It depends on how you’re defining selection sort! This idea of insertion is awfully similar to insertion sort, and if the stable selection sort only works for linked lists (without big efficiency issues), can we really say it can easily be made stable?

Sorting Review

	Bubble Sort K.C.	Bubble Sort D.M	Insertion Sort K.C.	Insertion Sort D.M	Selection Sort K.C.	Selection Sort D.M.
Best	$O(n)$	$O(1)$	$O(n)$	$O(1)$	$O(n^2)$	$O(1)$
Worst	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$ or $O(n)$ if LL	$O(n^2)$	$O(n)$
Average	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$ or $O(n)$ if LL	$O(n^2)$	$O(n)$

Sorting Review

- Let's talk Quicksort. Best case, worst case, and average cases? What about the Big-O for each?
- O($n * \log_2 n$) for best and average cases, O(n^2) worst case
- Why is the worst-case O(n^2)?
- We choose an extremal, leaving n-1 elements to still be partitioned. Gives us this recurrence relation:
- $Q(n) = O(n) + Q(n-1) = O(n^2)$
- What about the median of three improvement? Does that prevent this worst case? If so, why? If not, why not?

Sorting Review

- What about the median of three improvement? Does that prevent this worst case? If so, why? If not, why not?
- What if there are duplicate numbers—the extremal could be in all three spots for each median chosen. So no, it doesn't prevent the worst case…
- What if there are no duplicate elements in the list? *Then* does it prevent the worst case?
- Still no. Why? We could still get the second largest/smallest in the list each time. What recurrence relation would that give us?
- $Q(n) = O(n) + Q(n-2) + Q(1)$, Q(1) = c
- Show that this recurrence relation is $O(n^2)$

Sorting Review

- $Q(n) = O(n) + Q(n-2) + Q(1)$, Q(1) = c
- Show that this recurrence relation is $O(n^2)$
- $Q(n-2) = O(n) + Q(n-4) + c$
- $Q(n) = 2 * O(n) + Q(n-4) + 2c$
- $Q(n-2) = O(n) + Q(n-6) + c$
- $Q(n) = 3 * O(n) + Q(n-6) + 3c$
- $Q(n) = a * O(n) + Q(n-2a) + ac$
- $\mathrm{n} - 2a = 1,\ n = 1 + 2a, a = \frac{n-1}{2}$
- $Q(n) = \frac{n-1}{2} * O(n) + c + c * \frac{n-1}{2}$
- This has a term with $O(n^2)$

Sorting Review

- Sort 909 4 427 37 123 6 546 170 101 using Radix Sort.
- 1's position:

0	1	2	3	4	5	6	7	8	9
170	101		123	4		6 546	427 37		909

- 10's position

0	1	2	3	4	5	6	7	8	9
101 4 6 909		123 427	37	546			170		

Sorting Review

- Sort 909 4 427 37 123 6 546 170 101using Radix Sort.
- 10's position:

0	1	2	3	4	5	6	7	8	9
101 4 6 909		123 427	37	546			170		

- 100's position

0	1	2	3	4	5	6	7	8	9
4 6 37	101 123 170			427	546				909

Hashing Review

- Quadratic probing general function: $qp(key, i) = (h(key) + c * i + d * i^2)\ \%\ m$
- Let $h(key) = key\ \%\ 10$, c = 5, d = 5
- $qp(key, i) = (key\ \%\ 10 + 5 *\ i + 5 * i^2)\ \%\ 10$
- Let's insert 5, 10, 15

0	1	2	3	4	5	6	7	8	9
10					5				

qp(5, 0) = 5
qp(10, 0) = 0
qp(15, 0) = 5 (collision!)
qp(15, 1) = 5 (collision!)
qp(15, 2) = 5 (collision!)

qp(15, 3) = 5 (collision!)
qp(15, 4) = 5 (collision!)
...

Hashing Review

- Double hashing general function: $dh(key, i) = (h_1(key) + i * h_2(key)) \% m$
- Let's do a specific example. Let $h_1(key) = \lfloor 7 * (key * .2 - \lfloor key * .2 \rfloor) \rfloor$ and $h_2(key) = |5 * key + 1| \% 7$
- Let's insert 3, 4, 5, 13

0	1	2	3	4	5	6
5				3	4	13

dh(3, 0) = 4
dh(4, 0) = 5
dh(5, 0) = 0
dh(13, 0) = 4 (collision!)
dh(13, 1) = 0 (collision!)

dh(13, 2) = 6

BST Review

- Splay trees move a searched for value to the root for every single search operation! How could this possibly be worth our time?
- An individual search operation might be expensive, but as popular items are more frequently searched than less popular items, those popular items will tend to be near the top of the tree over time
- This search/splay process, if done to an item near the top of the tree, isn’t actually that expensive