ADS 2021: Week 4 Solutions

Solutions for week 4 of Algorithms and Data Structures.

2.1.1 Selection Sort - Green

```
                a[]
i  min  0 1 2 3 4 5 6 7 8 9 10 11
        E A S Y Q U E S T I  O  N
0    1  E A S Y Q U E S T I  O  N
1    1  A E S Y Q U E S T I  O  N
2    6  A E S Y Q U E S T I  O  N
3    9  A E E Y Q U S S T I  O  N
4   11  A E E I Q U S S T Y  O  N
5   10  A E E I N U S S T Y  O  Q
6   11  A E E I N O S S T Y  U  Q
7    7  A E E I N O Q S T Y  U  S
8   11  A E E I N O Q S T Y  U  S
9   11  A E E I N O Q S S Y  U  T
10  10  A E E I N O Q S S T  U  Y
11  11  A E E I N O Q S S T  U  Y
        A E E I N O Q S S T  U  Y
```

2.1.4 Insertion Sort - Green

```
         a[]
i  j 0 1 2 3 4 5 6 7 8 9 10 11
     E A S Y Q U E S T I  O  N
1  0 A E S Y Q U E S T I  O  N
2  2 A E S Y Q U E S T I  O  N
3  3 A E S Y Q U E S T I  O  N
4  2 A E Q S Y U E S T I  O  N
5  4 A E Q S U Y E S T I  O  N
6  2 A E E Q S U Y S T I  O  N
7  5 A E E Q S S U Y T I  O  N
8  6 A E E Q S S T U Y I  O  N
9  3 A E E I Q S S T U Y  O  N
10 4 A E E I O Q S S T U  Y  N
11 4 A E E I N O Q S S T  U  Y
     A E E I N O Q S S T  U  Y
```

2.2.2 Merge Sort - Green

	a[]											
	0	1	2	3	4	5	6	7	8	9	10	11
	E	A	S	Y	Q	U	E	S	T	I	O	N
merge(a, 0, 0, 1)	A	E	S	Y	Q	U	E	S	T	I	O	N
merge(a, 0, 1, 2)	A	E	S	Y	Q	U	E	S	T	I	O	N
merge(a, 3, 3, 4)	A	E	S	Q	Y	U	E	S	T	I	O	N
merge(a, 3, 4, 5)	A	E	S	Q	U	Y	E	S	T	I	O	N
merge(a, 0, 2, 5)	A	E	Q	S	U	Y	E	S	T	I	O	N
merge(a, 6, 6, 7)	A	E	Q	S	U	Y	E	S	T	I	O	N
merge(a, 6, 7, 8)	A	E	Q	S	U	Y	E	S	T	I	O	N
merge(a, 9, 9, 10)	A	E	Q	S	U	Y	E	S	T	I	O	N
merge(a, 9, 10, 11)	A	E	Q	S	U	Y	E	S	T	I	N	O
merge(a, 6, 8, 11)	A	E	Q	S	U	Y	E	I	N	O	S	T
merge(a, 0, 5, 11)	A	E	E	I	N	O	Q	S	S	T	U	Y
	A	E	E	I	N	O	Q	S	S	T	U	Y

2.1.6 Identical Keys - Green Insertion sort runs in linear time, since it will only make one comparison per element and do no exchanges. Selection sort will always run in quadratic time.

2.1.7 Reverse Order - Green Selection sort because even though both selection sort and insertion sort will run in quadratic time, selection sort will only make N exchanges, while insertion sort will make $N * N/2$ exchanges.

2.1.8 Random Order - Yellow Insertion sort is linear when the array is already sorted (which is also the case when all elements are equal), but quadratic in other cases. As this array is randomly ordered and items can have more than one value, the running time is quadratic.

2.1 [Mehlhorn-Sanders] Analysis - Yellow

a) true for $c = 17$ and $N_0 = 0$

b) true for $c = 3$ and $N_0 = 0$

c) true for $c = 2$ and $N_0 = 1,000,000$

d) false

e) true for $c = 11$ and $N_0 = 1$

2.2.15 Bottom-up Queue Mergesort - Yellow To merge two sorted queues, we can create a new queue, mergedQueue, and then use peek() on the input queues to find the lesser (or greater depending on the order to sort in) element, which we can then dequeue from the input queue and enqueue in mergedQueue. We do this until one of the input queues are empty and then we enqueue the remaining elements in the non-empty queue.

2.1.13 Deck Sort - Yellow For a standard deck of cards, each card has a known correct position from index 0 to 51. We turn over the cards at position 0 and 1, and then exchange card 0 with the card at card 0s correct position, and card 1 with the card at card 1s correct position. Then we turn the new cards at position 0 and 1 over again and repeat the exchanges. When position 0 or 1 contains the correct card, we move upwards through the row by turning over position 2, then 3, all the way until the end of the deck. In the worst case each card is checked at most twice (once to be put in the right place, and once when we move up through the deck) and we do a maximum of 52 exchanges. We can therefore sort the deck in linear time.

2.1.14 Dequeue Sort - Red First we compare the top two cards. If the top card is bigger than the second card, we swap the two, remember/mark the top card, and then move the top card to the bottom.

Then we continue to compare the top cards, swap them if the top card is bigger than the second, and move the top card to the bottom until the marked/remembered card is second in the deck. (Essentially we move the smaller of the top two cards to the bottom.)

Once the marked card is second in the deck, we move the top card to the bottom and the given iteration is finished. We then continue to do iterations until we do an entire iteration without swapping any cards - at this point the deck is sorted.

2.2 [Mehlhorn-Sanders] O-notation - Red

a) As $f(n) = O(g(n))$ we know that there exists c and N_0 such that $f(n) \leq c \cdot g(n)$ for $n > N_0$.
Therefore:
$f(n) + g(n) \leq (c+1) \cdot g(n) \Rightarrow f(n) + g(n) = O(g(n))$ for all $n > N_0$

b) We know that $f(n) = O(f(n))$ for $n > N_0$ and $g(n) = O(g(n))$ for $n > N_1$. It follows that:

$$O(f(n)) \cdot O(g(n)) = (c_1 \cdot f(n)) \cdot (c_2 \cdot g(n)) = (c_1 \cdot c_2) \cdot (f(n) \cdot g(n)) = O(f(n) \cdot g(n))$$

for all $n > max(N_0, N_1)$