328 HW#6

1. Quickselect

1a. k= 2nd least element

[2, -1, 3, 8, 9, 0, 19, 6, 35, 17, 20]

p = 3 [2, -1, 0, 3, 8, 9, 19, 16, 35, 17, 20]

p = 0 [2, -1, 0]

p = -1 [-1, 0, 2]

[-1, 0] → 0 is 2nd least element.

2a. K= 2nd least element

p = 10 [10, 11, 12, 13, 14, 15, 16, 17, 18]

$\boxed{\emptyset}$ 10 |11|12|13|14|15|16|17|18|

k = 2 - (0+1) = 1

p = 11 $\boxed{\emptyset}$ 11 |12|13|14|15|16|17|18|

K = 2 - (1+1) = 0 ⇒ 11 is 2nd least element.

1b. K = 4th least element

p = 2 |2|-1|3|8|9|0|19|6|35|17|20|

K = 4 - (2+1) = 1 |-1|0| 2 |3|8|9|19|6|35|17|20|

p = 3 ~~|-1|0|2|~~ 3 |8|9|19|6|35|17|20|

a_L a_R → 3 is 4th least least element.

2b. K = 4 $\boxed{\emptyset}$ 10 |11|12|13|14|15|16|17|18| p = 10

K = 4 - (0+1) = 3 ~~10~~ |11|12|13| 14 |15|16|17|18| p = 11

~~|10|11|~~ 12 |13|14|15|16|17|18| p = 12

K = 4 - (2+1) = 1 |10|11|12| 13 |14|15|16|17|18| p = 13

↳ 13 is 4th least element.

2.

2. Looking at 2b, k would be the last element, 18 since it's a sorted list. Similarly to quicksort, quickselect would have a time complexity of $O(n^2)$ (worst case) Since each number in the list would need to be one set as a pivot and two partitioned either towards the left or to the right. These two tasks would need a nested for loop for the size of n, thus n^2.

3. The average-case running time of quickselect algorithm would probably be O(n) since this algorithm is potentially faster than quick sort because instead of sorting the entire list, we are only trying to find one element. Quickselect can be much faster because during partitioning we are only focusing on one side, rather than two.

4. algorithm to return max k numbers from unsorted array. O(n)
First, we would need to sort the unsorted array so that it will be easier to pick out and return the max k numbers. Since we are going for O(n), we can use Selection sort to sort the array and binary search to find the max k numbers. This should be O(n) since it searches using each input which is n.

5. T(n) for MSS:

$$\sum_{i=0}^{n-1}\sum_{j=1}^{n-1}\sum_{k=i}^{j} c = \sum_{i=0}^{n-1}\sum_{j=1}^{n-1} c(j-i+1) \rightarrow cj - ci + c$$

$$\overset{\nearrow \text{turns } 0}{}$$

$$= \sum_{i=0}^{n-1}\sum_{j=1}^{n-1} cj - \sum_{i=0}^{n-1}\sum_{j=1}^{n-1} i + \sum_{i=0}^{n-1}\sum_{j=i}^{n-1} c$$

$$\hookrightarrow \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} cj + \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} c \qquad \nearrow \frac{0(0-1)}{2} \rightarrow 0$$

$$= c\sum_{i=0}^{n-1}\left(\frac{n(n-1)}{2}\right) - \cancel{\left(\frac{i(i-1)}{2}\right)} + \sum_{i=0}^{n-1} c(n\cancel{-1}-i\cancel{+1}) \quad \nearrow cn - \cancel{ci}$$

$$= c\left(\frac{n(n\cdot(n-1))}{2}\right) + cn(n)$$

$$\hookrightarrow c(\sim n^3) + c(\sim n^2)$$

$$= O(n^3) + O(n^2) \rightarrow \boxed{O(n^3)}$$

6. $\sum_{i=0}^{n-1}\sum_{j=1}^{n-1} c(n-j) \rightarrow \sum_{i=0}^{n-1} c(n-i)$

$= \sum_{i=0}^{n-1} cn - \sum_{i=0}^{n-1} ci$

$= c \cdot n^2 - c \cdot \left(\frac{n^2-\overset{\rightarrow n}{n}}{2}\right) \Rightarrow c \cdot n^2 - cn \rightarrow \boxed{O(n^2)}$

The better programmer was able to decrease the running time by n.

7. $[6, 18, 4, \{ 11, 14, 2]$

a_L a_R

size = $n/2$ size = $n/2$

This divide and conquer approach to solving the MSS would most likely be faster than the other two student's since this algorithm somewhat mirrors merge sort where the array is divided in 2 and we recooperate the numbers recursively so that when we arrive at the base case, the program or function will stop. Dividing all the values would be logn, while "conquering" which includes recursively finding the MSS within the two arrays would be $\frac{n}{2}+\frac{n}{2}=\frac{2n}{2}$ $\Rightarrow n$ so we get $\boxed{O(n\log n)}$

8. $\sum_{i=0}^{n-1} cn$ $\Rightarrow$ This extraordinary student was able to solve the MSS problem the fastest with a running time of O(n) because they only use one for loop statemen that iterates throughout the size of n. The if and els statements just updates the max if a newMax is found and this isn't the case for all of n, we can leave it as constant runtime O(1). Taking the larger of the two, we get $\boxed{O(n)}$.

A. Q.8 method:

7	2	-14	38	52	-37	4	12	-4	6	3	2

if (sum > maxSum) → update maxSum

(7 > 0) → 7

(9 > 7) → 9

(-5 > 9) X → (-5 < 0) → sum = 0

(38 > 9) → 38

(90 > 38) → 90

(53 > 90) X → (53 < 0) X → continue

(57 > 90) X → (57 < 0) X → continue

(69 > 90) X → (69 < 0) X → 〃

(65 > 90) X → (65 < 0) X → 〃

(71 > 90) X → (71 < 0) X → 〃

(74 > 90) X → (74 < 0) X → 〃

(76 > 90) X → (76 < 0) X → 〃

↳ exits for loop with $\boxed{\text{maxSum} = 90}$

B.

3	6	-20	11	-15	26	-43	10	-14	27	0	39

(3 > 0) → 3

(9 > 3) → 9

(-11 > 9) X → (-11 < 0) → sum = 0

(11 > 9) → 11

(-4 > 11) X → (-4 < 0) → sum = 0

(26 > 11) → 26

(-17 > 26) X → (-17 < 0) → sum = 0

(10 > 26) X → (10 < 0) X → contine

(-4 > 26) X → (-4 < 0) → sum = 0

(27 > 26) → 27

(27 > 27) X → (27 < 0) X → continue

(66 > 27) → 66

↳ exits loop w/ $\boxed{\text{maxSum} = 66}$