1. Find the

1. 2^{nd} least element
2. 4^{th} least element

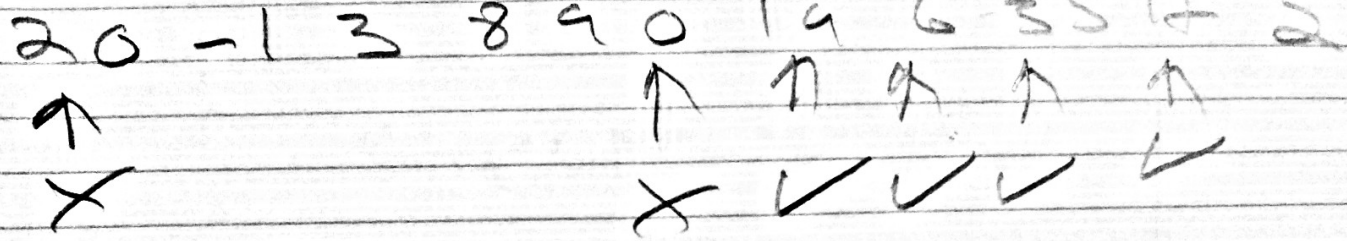
Using the random find statisics algorithm (Quickselect)

- 2,-1,3,8,9,19,6,35,17,20
- 10,11,12,13,14,15,16,17,18

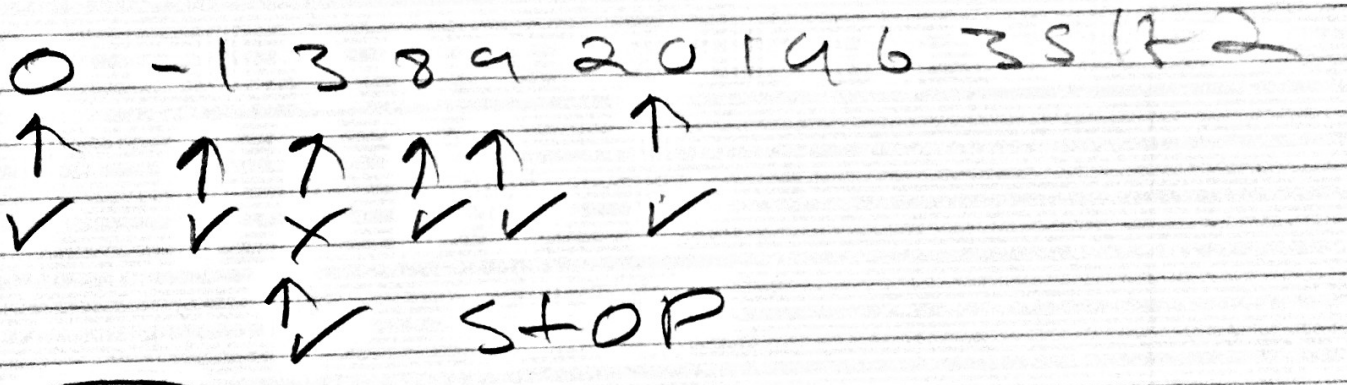
(1a) part 1

2 -1 3 8 9 0 19 6 3 5 17 20

i = 2
pivot = 2



swap 0 and 20



0 -1 2 8 9 20 19 6 3 5 17 3

QS

i = 0
pivot = 0

0 - 1

k=2

pivot = 0

-1 0

↑

↑ Step: partitioning

0 ps + 2^(k-1) left elements

(1a) part 2

10 11 12 13 14 15 16 17 18

k=2
pivot=10

18 11 12 13 14 15 16 17 10

Stop partitioning

10 (11 12 13 14 15 16 17 18)

QS

k=1 11 12 13 14 15 16 17 18
pivot= 11

18 12 13 14 15 16 17 11

11 12 13 14 15 16 17 18

11 is the 2nd least element

(1b) part 2

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

pivot = 2
k = 4

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

swap 20 and 2

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

stop partitioning

0 -1 2 8 9 20 19 6 3 5 | 3

QS

8 9 20 19 6 35 17 3

K = 1
Pivot = 8

3 9 20 19 6 35 17 8

3 6 20 19 9 35 17 8

3 6 8 19 9 35 17 20

QS

K = 1
Pivot = 3

6 3

3 6

3 is 4th least element

(b) Part 2

10 11 12 13 14 15 16 17 18

Pivot = 10
K = 4

18 11 12 13 14 15 16 17 10

Stop Partition

10 (11 12 13 14 15 16 17 18)

QS

11 12 13 14 15 16 17 18

Pivot = 16
K = 3

18 12 13 14 15 16 17 11

Stop Partition

11 (12 13 14 15 16 17 18)

QS

12 13 14 15 16 17 18

pivot = 12

← = 2

18 13 14 15 16 17 12

Step partitioning

12 13 14 15 16 17 18

QS

pivot = 13 13 14 15 16 17 18

← = 1

18 14 15 16 17 13

Step partitioning

13 14 15 16 17 18

13 is the 4th least element

2. Calculate the worst-case running time of Quickselect.

An example of the worst-case of Quickselect is looking for the n^{th} least element in a sorted array with a length of n. The Quickselect would select the leftmost valid index then perform the partition and if the index is the not the n^{th} element would result in needing to do a recursive call on a subproblem of one less the original size. The recurrence is of the form $T(n) = T(n-1)+O(n)$ and evaluates to $T(n) = O(n^2)$.

$T(n) = T(n-1) + O(n)$

Level	Size	Cost
0	n	n
1	$n-1$	$n-1$
2	$n-2$	$n-2$
...	...	...
k	1	1

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$= O(n^2)$$

$$\boxed{T(n) = O(n^2)}$$

3. Calculate the average-case running time of Quickselect algorithm. Explain.

In the average case the pivot splits the subarrays roughly in half and if it is not the k^{th} least element will recurse on one of the subarrays. The recurrence is of the form $T(n) = T(\frac{n}{2}) + O(n)$. This evaluates to $O(n)$.

$T(n) = T(k_{n-1}) + O(n)$

Level	Size	Time
0	n	n
1	$\frac{n}{2}$	$\frac{n}{2}$
2	$\frac{n}{4}$	$\frac{n}{4}$
⋮	⋮	⋮
k	1	1

$k = \log_2(n)$

$$\sum_{i=0}^{k-1} \frac{n}{2^i} + 2^k =$$

$$n \sum_{i=0}^{k-1} \left(\frac{1}{2}\right)^i + 2^{\log_2(n)} =$$

$$n \cdot 4(1) + n =$$

$$O(n)$$

$$T(n) = O(n)$$

4. Explain an algorithm to return the max k numbers from an unsorted array.

Assume A is an array. Alter the Quickselect to return the index of the k^{th} smallest value as opposed to the value itself. Compute the k^{th} largest index by using the Quickselect with the value $len(A) - k + 1$. The elements in the right-subarray of the index are the remaining max k numbers.

5. Suppose a student wrote the below code for the Maximum Subsequence Sum problem. What is the running time of this algorithm?

$$
\begin{aligned}
\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}\sum_{k=i}^{j} c &= \sum_{i=0}^{n-1}\sum_{j=i}^{n-1} c\cdot(j-i+1) \\
&= \sum_{i=0}^{n-1}\sum_{j=i}^{n-1} c\cdot j - \sum_{i=0}^{n-1}\sum_{j=i}^{n-1}\cdot i + \sum_{i=0}^{n-1}\sum_{j=i}^{n-1} c \\
&\leq \sum_{i=0}^{n-1}\sum_{j=i}^{n-1} c\cdot j + \sum_{i=0}^{n-1}\sum_{j=i}^{n-1} c \\
&= \sum_{i=0}^{n-1} c\cdot(\sum_{j=1}^{n-1} j - \sum_{j=1}^{i-1} j) + \sum_{i=0}^{n-1} c\cdot(n-1-i+1) \\
&= \sum_{i=0}^{n-1} c\cdot(\frac{n\cdot(n-1)}{2} - \frac{i\cdot(i-1)}{2}) + \sum_{i=0}^{n-1} c\cdot(n-i) \\
&\leq \sum_{i=0}^{n-1} c\cdot\frac{n\cdot(n-1)}{2} + \sum_{i=0}^{n-1} c\cdot n \\
&= c\cdot\frac{n\cdot n\cdot(n-1)}{2} + c\cdot n\cdot(n-1+1) \\
&= O(n^3) + O(n^2) \\
&= O(n^3)
\end{aligned}
$$

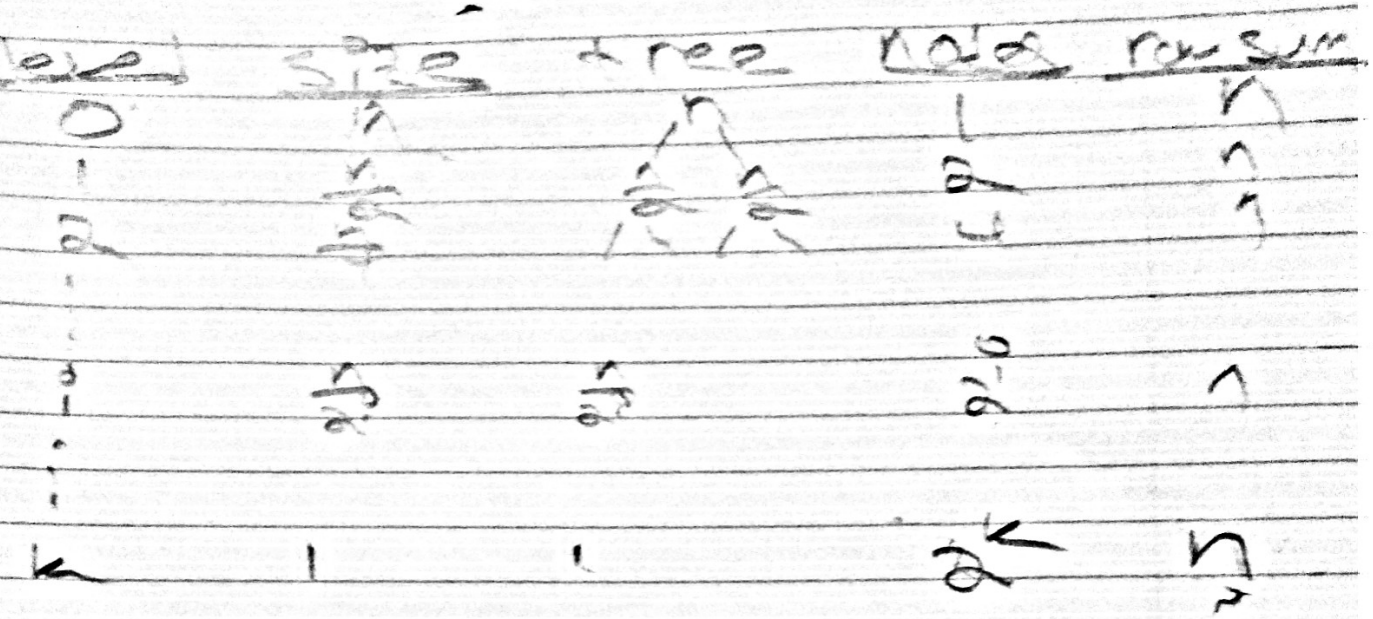
6. Suppose another student who is a better programmer wrote the below code for the Maximum Subsequence Sum Problem. What is the running time of this algorithm?

$$\begin{aligned}
\sum_{i=0}^{n-1}\sum_{j=i}^{n-1} c &= \sum_{i=0}^{n-1} c \cdot (n-i) \\
&= c \cdot \sum_{i=0}^{n-1} n - c \cdot \sum_{i=0}^{n-1} i \\
&= c \cdot n \cdot n - c \cdot \frac{(n-1)(n)}{2} \\
&= c \cdot \frac{n^2}{2} - c \cdot \frac{n}{2} \\
&= \Theta(n^2)
\end{aligned}$$

7. Suppose another students decides to solve the Maximum Subsequence problem using the divide and conquer technique. what is the running time fo this algorithm.

The recurrence of the algorithm will be of the form $T(n) = 2 \cdot T(\frac{n}{2}) + O(n)$, this evaluates to $O(n \cdot log(n))$.

$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n)$



$k = \log_2(n)$

$\sum_{i=0}^{k} n = n(k+1)$

$= n(\log_2(n)+1)$

$= O(n \log_2(n))$

$$T(n) = O(n \log_2(n))$$

8. Suppose an extraordinary student decides to solve the Maximum Subsequence Sum problem. using the below code. What is the running time of this algorithm.

$$\begin{aligned}\sum_{i=0}^{n-1} c &= c \cdot (n-1+1) \\ &= c \cdot n \\ &= \Theta(n)\end{aligned}$$