Question 1: Modified Quick Sort (14 points = 2 + 2 + 2 + 4 + 4)

Recall the following QuickSort algorithm discussed in class.

```
1 QuickSort(A, p, r)
2 if p < r then
3  |  q = Partition(A, p, r);
4  |  QuickSort(A, p, q − 1) ;
5  |  QuickSort(A, q + 1, r) ;
6 end
```

Suppose that the Partition function at line 2 was replaced by the following function Modified_Partition.

```
1 Modified_Partition(A, p, r)
2 x = A[p];
3 i = p;
4 j = r;
5 while TRUE do
6  |  while j > p and A[j] ≥ x do
7  |  |  j = j − 1;
8  |  end
9  |  while i < r and A[i] ≤ x do
10 |  |  i = i + 1;
11 |  end
12 |  if i < j then
13 |  |  Exchange A[i] with A[j];
14 |  else
15 |  |  Exhange A[p] with A[j];
16 |  |  return j;
17 |  end
18 end
```

x is the pivot

x

i

j

add one element to both partitions because i and j stand on bad elements

x ≤

i

≥ x

j

pivot will be at index j

i = j

They cannot exceed each other because contradiction will happe

i j

i — j

(a) Demonstrate the operation of Modified_Partition when called with $A = [13, 19, 9, 5, 14, 8, 7, 4, 21]$, $p = 1$, and $r = 9$. Show the values of the array after each iteration of the while loop in lines 5-18 and the final return value.

x=13 [13, 19, 9, 5, 14, 8, 7, 4, 21]
p i r j

first iteration
[13, 19, 9, 5, 14, 8, 7, 4, 21]
i j
[13, 4, 9, 5, 14, 8, 7, 19, 21]
i j

second iteration
[13, 4, 9, 5, 14, 8, 7, 19, 21]
i j
[13, 4, 9, 5, 7, 8, 14, 19, 21]
i j

[13, 4, 9, 5, 7, 8, 14, 19, 21]
p j i
[8, 4, 9, 5, 7, 13, 14, 19, 21]
p j r

(b) Explain the functionality of Modified_Partition.

The first loop extends the partition of j where all the elements with index>j are greater than pivot.
The second loop extends the partition of i where all the elements with index<i are less than pivot.
If i<j i swap elements at i,j increasing partition i and j, then repeating again.
If i>j (they are never equal) then i am done we only need to put the pivot at index j putting it the last element of partition i.

(c) Is the modified QuickSort algorithm correct when it uses Modified_Partition? Explain your reasoning.

Yes, because I partition the array so that every element to my left is ≤ pivot and every element to my right is ≥ pivot.
The pivot is always the first element of the range.
So, after placing pivot and putting all the elements in the right place, I repeat the process for the left half and the right half.

(d) What is the best and worst cases of the modified QuickSort?

Best case that the pivot chosen makes the array split in half a balanced split so I divided my problem in half and then repeating same process for the smaller problems, resulting in $O(n \log n)$

Worst case that the array is sorted, so the resulting array split based on the pivot will not be a balanced split It will just decrease the array size by 1 resulting in $O(n^2)$

(e) Write the recurrences representing the best and worst case running times of the modified QuickSort (you don't need to solve them).

Best Case: $T(n) = 2T(\frac{n}{2}) + 2\Theta(n)$

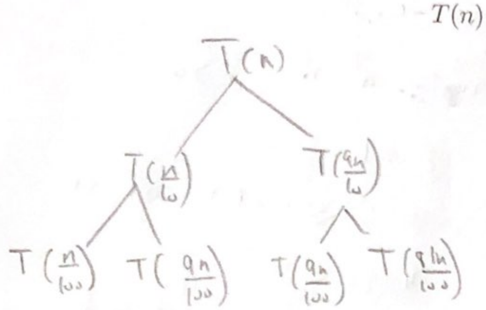
Worst Case: $T(n) = T(n-1) + 2\Theta(n)$

Question 2: Recursion Trees (10 points = 8 + 2)

(a) Get an **upper bound** on the running time of the following recurrence by using the recursion tree method. Draw the recursion tree and show all of your workout.

$$T(n) = T(\frac{n}{10}) + T(\frac{9n}{10}) + \Theta(n)$$

← longer branch



height	size	cost
$i=0$	n	n
$i=1$	$\frac{9n}{10}$	$\frac{9n}{10}+\frac{n}{10}=n$
$i=2$	$\frac{81n}{100}$	$\frac{(81+9+9+1)n}{100}=n$
$i=3$	$\left(\frac{9}{10}\right)^i n$	n

$\left(\frac{9}{10}\right)^h n = 1$

$\left(\frac{9}{10}\right)^h = \frac{1}{n}$

$\left(\frac{10}{9}\right)^h = n$

$h = \log_{\frac{10}{9}}(n)$

$\cancel{C.L = \left(\frac{9}{10}\right)^h n(n)}$

$\cancel{= \left(\frac{9}{10}\right)^h n^2}$

$\cancel{\left(\frac{9}{10}\right)^{\log_{10/9}(n)} n^2}$

$cost = \sum_{i=0}^{h\cancel{+1}} n + \cancel{CL}$

$= \sum_{i=0}^{\log_{10/9}(n)} n = n(\log_{\frac{10}{9}}(n) - 0 + 1)$

$= n[\log_{\frac{10}{9}}(n) + 1]$

$T(n) = O(n\log_{\frac{10}{9}}(n) \cancel{+n})$

(b) Can the recurrence in part (a) be solved using the master method? Justify your answer.

No, because the splitting is not balanced

The splitting must be of the same size.

Question 3: Master Theorem (20 points = 4 + 4 + 4 + 4 + 4)

Can the following recurrences be solved using the master method? If yes, solve them. If not, explain why. Show all of your workout.

(a) $T(n) = 4T(\frac{n}{2}) + n^3$

$a=4, b=2, f(n)=n^3$

$C.R = f(n) = n^3$

$C.L = n^{\log_b a} = n^{\log_2 4} = n^2$

$C.R > C.L$ looks like Case 3

$f(n) = \Omega(n^{2+\varepsilon})$

$n^3 \geq n^2 \cdot n^{\varepsilon}$, $\varepsilon > 0$

$\varepsilon = 1$

$af(\frac{n}{b}) \leq c f(n)$ $0<c<1$

$4(\frac{n}{2})^3 \leq cn^3$

$\cancel{4}\frac{\cancel{n^3}}{\cancel{4}(2)} \leq c\cancel{n^3}$

$c \geq \frac{1}{2}$

$c = \frac{1}{2}$

$T(n) = \Theta(n^3)$

(b) $T(n) = 7T(\frac{n}{2}) + n^2$

$a=7, b=2, f(n)=n^2$

$C.R = f(n) = n^2$

$C.L = n^{\log_b a} = n^{\log_2 7} = n^{2.8}$

$C.L > C.R$ looks like Case1

$f(n) = O(n^{2.8-\varepsilon})$

$n^2 \leq \frac{n^{2.8}}{n^{\varepsilon}}$, $\varepsilon > 0$

$\varepsilon = 0.1$

$n^2 \leq n^{2.7}$

$T(n) = \Theta(n^{\log_2(7)})$

(c) $T(n) = T(\frac{n}{2}) + 1$

$a=1, b=2, f(n)=1$

$C.R = f(n) = 1$

$C.L = n^{\log_b a} = n^{\log_2 1} = n^0 = 1$

$C.R = C.L$ Case 2

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

(d) $T(n) = 3T(\frac{n}{2}) + \frac{n^{\log_2(3)}}{\log_2(n)}$

$a=3, b=2, f(n) = \frac{n^{\log_2(3)}}{\log_2(n)}$

$C.R = f(n) = \frac{n^{\log_2(3)}}{\log_2(n)} = \frac{n^{1.6}}{\log_2(n)}$

$C.L = n^{\log_b a} = n^{\log 3} = n^{1.6}$

$C.L > C.R$ looks like Case1

$f(n) = O(n^{1.6-\varepsilon})$

$\frac{\cancel{n^{1.6}}}{\log_2(n)} \leq \frac{\cancel{n^{1.6}}}{n^{\varepsilon}} \rightarrow \frac{1}{\log_2(n)} \leq \frac{1}{n^{\varepsilon}} \rightarrow \log_2(n) \geq n^{\varepsilon}$

falls in a gap between Case 1 and Case2

(e) $T(n) = 2T(\frac{n}{2}) - \frac{1}{n}$

$a=2, b=2, f(n) = -\frac{1}{n}$

Can not be solved using master theorem.

f(n) is not an asymptotically positive function.

Question 4: Divide and Conquer Algorithms (10 points)

Write an $O(log_2(n))$ time divide and conquer algorithm in **Pseudo Code** to find the smallest number of an array A in which the input array A first strictly decreases then strictly increases. For example, if $A = [6, 4, 2, 4, 7, 9]$, it first strictly decreases from 6 to 4 to 2, then it strictly increases from 2 to 4 to 7 to 9. Your algorithm is supposed to return the unique smallest element 2. No credit will be given to any algorithm that runs in more than $O(log_2(n))$.