Problem #1

Homework #2

problem#1

a) $T(n) = b \cdot T(n-1) + 1$ for some $b > 1$

[We're using Master Method.
$T(n) = aT(n-e) + f(n)$ for some $a, e > 0$, $d \geq 0$.]

$a = b$, $e = 1$, $f(n) = 1 = n^0$, so, $d = 0$

$f(n) = O(n^0) = O(1)$

$b > 1$, so $T(n) = O(n^0 b^n)$

$\therefore T(n) = \underbrace{O(b^n)}$ for some $b > 1$

↳ asymptotic upper bound

b) $T(n) = 3 \cdot T(n/9) + n\log n$

[We're using Master Method.
$T(n) = aT(n/b) + f(n)$ where $a \geq 1$, $b > 1$, and f is asymptotically positive.

$a = 3$, $b = 9 \Rightarrow n^{\log_b a} = n^{\log_9 3} = n^{\frac{1}{2}} = \sqrt{n}$; $f(n) = n\log n$

$f(n) = \Omega(n^{\log_9 3 + \varepsilon})$ for some $\varepsilon > 0$ and

for regularity,

$3(\frac{n}{9}\log\frac{n}{9}) \leq cn\log n$

$\frac{n}{3}\log\frac{n}{9} \leq cn\log n$, $c = \frac{1}{3} < 1$

$T(n) = \Theta(n\log n) \Leftrightarrow T(n) = O(n\log n)$ and $T(n) = \Omega(n\log n)$

$\therefore T(n) = \underbrace{O(n\log n)}$

↳ asymptotic upper bound.

Problem #2

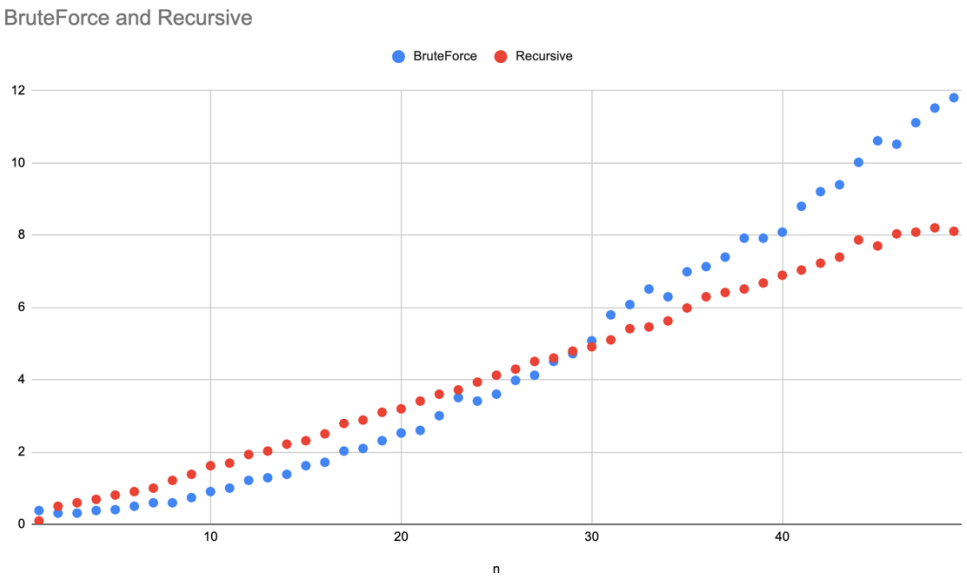
4.1-1

It will still return a subarray with the most positive elements.

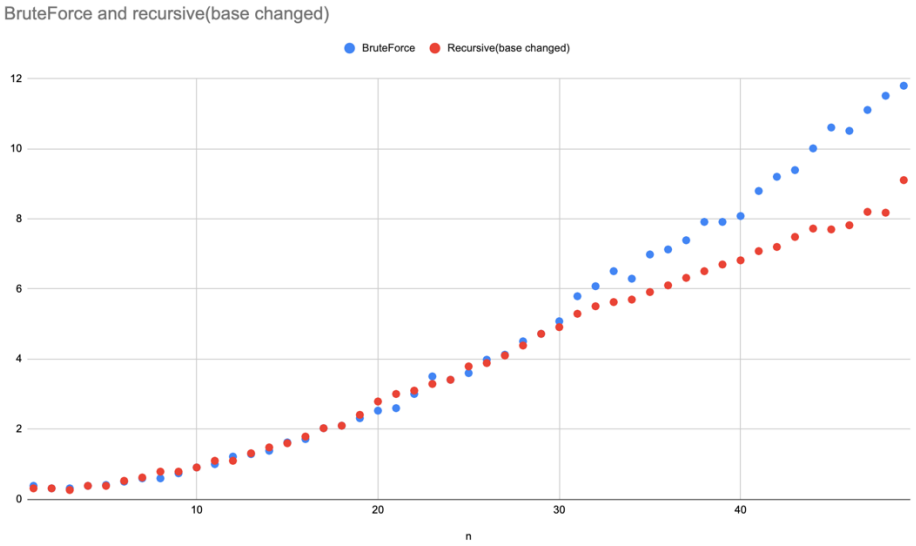
4.1-2

```
bruteMaximumArray(A):
    left = right = 0            // index starts at 0
    maxSum = 0
    n = len(A)                  // length of A

    for i in range(n):
        subSum = 0
        for j in range(i, n):
            subSum = subSum + A[j]
            if subSum > maxSum:
                maxSum = subSum
                left = i
                right = j

    return (left, right, maxSum)
```

4.1-3



On my computer, when size of array is 30, recursive algorithm beats brute-force algorithm. (x-axis: problem size, y-axis: running time)



When base case changed to use brute-force whenever the size of problem is less than 30, there no exists crossover point now. Until size gets to 29, both algorithms have the same performance.

4.1-4

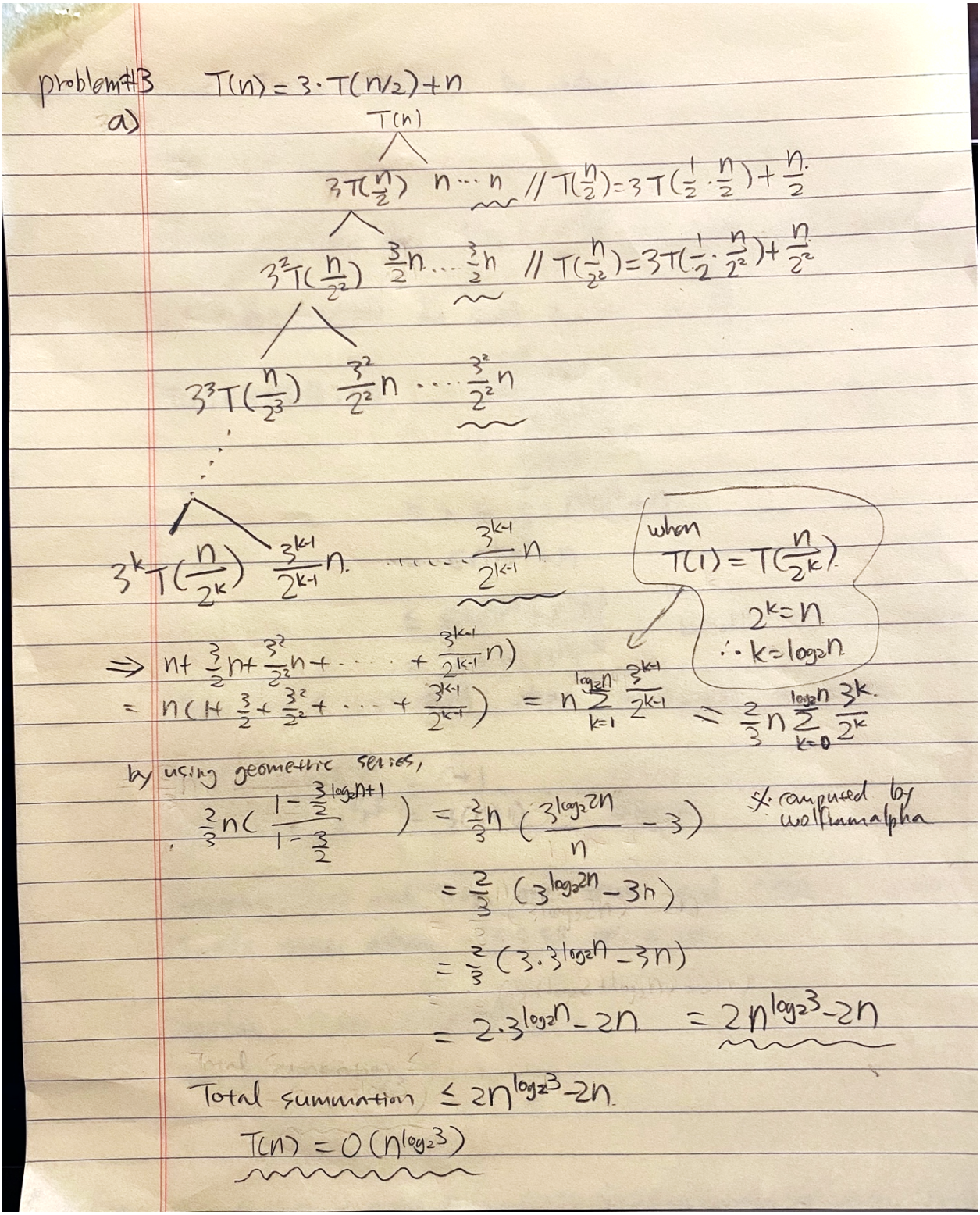
If a result returns negative sum value, return an empty subarray with sum of 0. Otherwise, run the algorithm to get subarray.

4.1-5

```
findMaximumSubarray_linear(A, low, high):
    left = right = 0
    maxSum = 0
    subSum = 0
    n = len(A)
    tempLeft = 0

    for i in range(n):
        subSum = max(A[i], subSum + A[i])
        if subSum > maxSum:
            maxSum = subSum
            right = i
            left = tempLeft
        if subSum == A[i]:
            tempLeft = i

    return (left, right, maxSum)
```

⇨ This algorithm's asymptotic running time is Θ(n) where n is size of array. The iteration inside of the function dominates all constant running times.

Problem #3


problem#3 $T(n) = 3 \cdot T(n/2) + n$
a)
$T(n)$
$3T(\frac{n}{2})$ $n \cdots n$ // $T(\frac{n}{2}) = 3T(\frac{1}{2} \cdot \frac{n}{2}) + \frac{n}{2}$
$3^2T(\frac{n}{2^2})$ $\frac{3}{2}n \cdots \frac{3}{2}n$ // $T(\frac{n}{2^2}) = 3T(\frac{1}{2} \cdot \frac{n}{2^2}) + \frac{n}{2^2}$
$3^3T(\frac{n}{2^3})$ $\frac{3^2}{2^2}n \cdots \frac{3^2}{2^2}n$
$3^kT(\frac{n}{2^k})$ $\frac{3^{k-1}}{2^{k-1}}n \cdots \frac{3^{k-1}}{2^{k-1}}n$
when $T(1) = T(\frac{n}{2^k})$
$2^k = n$
$\therefore k = \log_2 n$
$\Rightarrow n + \frac{3}{2}n + \frac{3^2}{2^2}n + \cdots + \frac{3^{k-1}}{2^{k-1}}n)$
$= n(1 + \frac{3}{2} + \frac{3^2}{2^2} + \cdots + \frac{3^{k-1}}{2^{k-1}}) = n\sum_{k=1}^{\log_2 n}\frac{3^{k-1}}{2^{k-1}} = \frac{2}{3}n\sum_{k=0}^{\log_2 n}\frac{3^k}{2^k}$
by using geometric series,
$\frac{2}{3}n(\frac{1-\frac{3}{2}^{\log_2 n+1}}{1-\frac{3}{2}}) = \frac{2}{3}n(\frac{3^{\log_2 2n}}{n} - 3)$ * computed by wolframalpha
$= \frac{2}{3}(3^{\log_2 2n} - 3n)$
$= \frac{2}{3}(3 \cdot 3^{\log_2 n} - 3n)$
$= 2 \cdot 3^{\log_2 n} - 2n = 2n^{\log_2 3} - 2n$
Total summation $\leq 2n^{\log_2 3} - 2n$
$T(n) = O(n^{\log_2 3})$

b) prove $T(n) = O(n^{\log_2 3})$ by induction.

$T(n) = 3T(n/2) + n$

we need to show that $T(n) \le cn^{\log_2 3}$ for some $c > 0$, $n \ge n_0$

Let $T(k) \le ck^{\log_2 3}$ for some $k < n$, $k = \frac{n}{2}$

$$\begin{aligned} T(n) = 3T(\tfrac{n}{2}) + n &\le 3 \cdot c(\tfrac{n}{2})^{\log_2 3} + n \\ &= 3 \cdot c \cdot \frac{n^{\log_2 3}}{2^{\log_2 3}} + n \\ &= 3 \cdot c \cdot \tfrac{1}{3} \cdot n^{\log_2 3} + n \\ &= cn^{\log_2 3} + n \\ &\le cn^{\log_2 3} + n^{\log_2 3} \\ &= (c+1)n^{\log_2 3} \end{aligned}$$

Thus $T(n) \le (c+1)n^{\log_2 3}$ whae $-1 < c < 0$ or $c > 0$

$T(1) \le (c+1)1^{\log_2 3} = c+1$ $\Rightarrow$ true

$T(2) \le (c+1)2^{\log_2 3} = 3(c+1)$ $\Rightarrow$ true.

Basecases, $T(1)$ and $T(2)$, are true and inductive hypothesis is also true where $-1 < c < 0$ or $c > 0$.

Therefore, $T(n) = O(n^{\log_2 3})$. □

Problem #4

problem #4

a) ☐ Let $\alpha = \frac{1}{2}$, $n = 7$, and $A = [5, 2, 4, 1, 6, 9, 3]$

$m = \lceil \frac{1}{2} \cdot 7 \rceil = 4$

badSort ([5,2,4,1]) // first half

→ $m = \lceil \frac{1}{2} \cdot 4 \rceil = 2$, $A = [5, 2]$ swap → $A = [2, 5]$

→ $m = \lceil \frac{1}{2} \cdot 4 \rceil = 2$, $A = [4, 1]$ swap → $A = [1, 4]$

→ $A = [2, 5, 1, 4]$ index 0, 1, 2, 3

badSort ([4,6,9,3]) // second half

→ $m = 2$, $A = [4, 6]$ → $A = [4, 6]$

→ $m = 2$, $A = [9, 3]$ swap → $A = [3, 9]$

→ $A = [4, 6, 3, 9]$ index 3, 4, 5, 6

badSort ([2, 5, 1, 4]) ……> $A = [2, 5, 1, 4]$ // first half

So, final result of this algorithm is $A = [2, 5, 1, 4, 6, 3, 9]$ which is not sorted.

☐ Let $\alpha = \frac{1}{3}$, $n = 7$.

$m = \lceil \frac{1}{3} \cdot 7 \rceil = 3$

badSort (A[0 … 2])

badSort (A[4 … 6])

badSort (A[0 … 2])

When $\alpha = \frac{1}{3}$, the sorting algorithm even gets worse because there's a gap (index 3) which never get sorted. This applies to all α greater than 0 and less than 1/2.

☐ From the observations, we see that when $\alpha = \frac{1}{2}$ or $\alpha < \frac{1}{2}$, the sorting algorithm does not work since the recursion functions inside of the function have incorrect parameters with the wrong range of array.

b)

badSort(A[0...2]) // if n=3
m = $\lceil \frac{3}{4} \cdot 3 \rceil = \lceil \frac{9}{4} \rceil = 3$
badSort(A[0...2])
badSort(A[0...2]) } -> never ends....
badSort(A[0...2])

- This shows that badSort never ends because the array with size of 3 will reproduce the parameters which is the same with the original array. SO, this algorithm does not work when $\alpha = 3/4$.

- By using the floor value of $\alpha \cdot n$, this problem can be fixed.

badSort(A[0...2])
  m = $\lfloor \frac{3}{4} \cdot 3 \rfloor = 2$.
  badSort(A[0...1])
  badSort(A[1...2])
  badSort(A[0...1])

- or, we can add a conditional statement after assigning m lik below,
  if m==n and m>2
    m = n-1

then, the algorithm sort the array when n=3.

c)

if (n=2) and (A[0] > A[1]) } → Θ(1)
  swap A[0] and A[1]
else if (n > 2)
  m = ⌈α·n⌉ → Θ(1)
  badSort(A[0 ··· m-1])
  badSort(A[n-m ··· n-1])
  badSort(A[0 ··· m-1])

$$T(n) = \begin{cases} \Theta(1) & \text{if } n=2 \\ 3T(\alpha n) + \Theta(1) & \text{if } n>2 \end{cases}$$

d) $T(n) = 3T(\frac{2}{3}n) + 1$, if $n > 2$ (which dominates $n=2$ case)

Using master method, $T(n) = aT(\frac{n}{b}) + f(n)$

$a = 3 \geq 1$, $b = \frac{3}{2} > 1$, and $f(n) = 1$

$\Rightarrow n^{\log_{3/2} 3}$

$f(n) = O(n^{\log_{3/2} 3 - \epsilon})$ for some constant $\epsilon > 0$

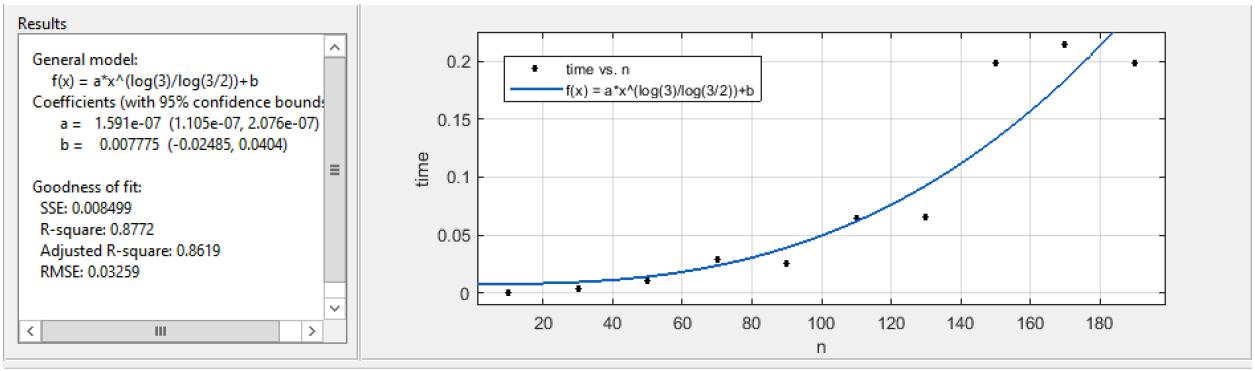
$\therefore T(n) = \Theta(n^{\log_{3/2} 3})$

Problem #5
b)

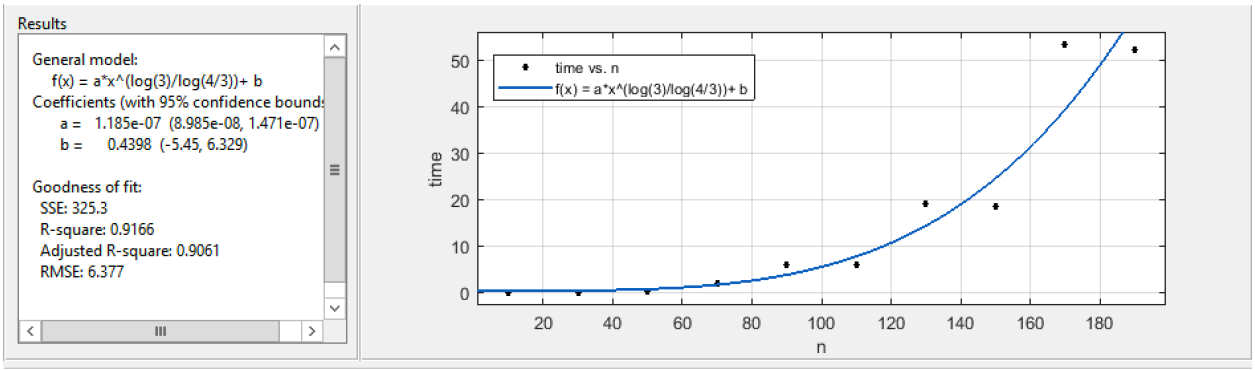
badSortTime		
n	**time when a = 2/3**	**time when a = 3/4**
10	0.000129222869873046	0.000391006469726562
30	0.00338625907897949	0.086014986038208
50	0.010854959487915	0.205365896224975
70	0.0285718441009521	2.00729489326477
90	0.0259640216827392	5.90985989570617
110	0.0641460418701171	6.13331699371337
130	0.0660350322723388	19.2630069255828
150	0.198567152023315	18.659126996994
170	0.214653253555297	53.5301828384399
190	0.198381900787353	52.3212463855743

c)

badSort when a = 2/3



badSort when a = 3/4



Theoretical running time of badSort when a=2/3: Θ(n^(log3/log(3/2)) = Θ(n^2.7095…)
Theoretical running time of badSort when a=3/4: Θ(n^(log3/log(4/3)) = Θ(n^3.8188…)

There's not much discrepancy between experimental running times and theoretical running times in badSort for both a=2/3 and a=3/4.

d)
Comparing two plots, badSort when a=2/3 definitely better performs. Theoretically badSort when a=2/3 sorts elements much faster. Overall a=2/3 provides better performance.

<citations>
https://atekihcan.github.io/CLRS/E04.01-05/
https://piazza.com/class/k4yfecd7h6gf?cid=135
https://www.geeksforgeeks.org/stooge-sort/