Problem 1

Write a recurrence relation for the pseudocode.
Solve it by the substitution, recursion-tree, and master methods.

```
power2(x, n):
   if n == 0:
      return 1
   if n == 1:
      return x
   if (n % 2) == 0:
      return power2(x, n // 2) * power2(x, n // 2)
   else:
      return power2(x, n // 2) * power2(x, n // 2) * x
```

$$\text{Base Case: } T(n) = c_1 \text{ when } n \leq 1, \text{ where } c = \text{some constant} \tag{1}$$

$$\text{Recursion Case: } T(n) = 2T\left(\frac{n}{2}\right) + c \text{ when } n \geq 1, \text{ where } c = \text{some constant} \tag{2}$$

1a) Substitution of $T(n) = 2T\left(\frac{n}{2}\right) + c$

Assign $T(n)$ as $Eq1$.

$$T(n) = 2T\left(\frac{n}{2}\right) + c \longrightarrow Eq1 \tag{3}$$

Find $T\left(\frac{n}{2}\right)$:

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{\frac{n}{2}}{2}\right) + c \tag{4}$$

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + c \text{ by product of numerator and quotient's reciprocal} \tag{5}$$

Substitute $T\left(\frac{n}{2}\right)$ in $Eq1$ to find $Eq2$:

$$T(n) = 2\left(2T\left(\frac{n}{4}\right) + c\right) + c \tag{6}$$

$$T(n) = 4T\left(\frac{n}{4}\right) + 2c \text{ by simplifying} \tag{7}$$

$$T(n) = 4T\left(\frac{n}{4}\right) + 2c \longrightarrow Eq2 \tag{8}$$

1a) Substitution of $T(n) = 2T\left(\frac{n}{2}\right) + c$ (Continued)

Find $T\left(\frac{n}{4}\right)$:

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{\frac{n}{4}}{2}\right) + c \tag{9}$$

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + c \text{ by product of numerator and quotient's reciprocal} \tag{10}$$

Substitute $T\left(\frac{n}{4}\right)$ in $Eq2$ to find $Eq3$:

$$T(n) = 4\left(2T\left(\frac{n}{8}\right) + 2c\right) + c \tag{11}$$

$$T(n) = 8T\left(\frac{n}{8}\right) + 3c \text{ by simplifying} \tag{12}$$

$$T(n) = 8T\left(\frac{n}{8}\right) + 3c \longrightarrow Eq3 \tag{13}$$

The pattern:

$$Eq1 \longrightarrow T(n) = (2^1)T\left(\frac{n}{2^1}\right) + c \tag{14}$$

$$Eq2 \longrightarrow T(n) = (2^2)T\left(\frac{n}{2^2}\right) + 2c \tag{15}$$

$$Eq3 \longrightarrow T(n) = (2^3)T\left(\frac{n}{2^3}\right) + 3c \tag{16}$$

$$k^{th}Eq \longrightarrow T(n) = (2^k)T\left(\frac{n}{2^k}\right) + kc \tag{17}$$

$T()$ recursively reaches base case $T(1)$. From the base case, $T(1) = c_1$, where $c \leq 1$.

$$T\left(\frac{n}{2^k}\right) = T(1) \tag{18}$$

Solve for k:

$$\frac{n}{2^k} = 1$$

$$\implies \quad n = 2^k \text{ by clearing the denominator} \tag{19}$$

$$\implies log_2 n = log_2 2^k \text{ by logarithms} \tag{20}$$

$$\implies log_2 n = k(log_2 2) \text{ by the power rule} \tag{21}$$

$$\implies log_2 n = k, \text{ as } log_2 2 = 1 \tag{22}$$

1a) Substitution of $T(n) = 2T\left(\frac{n}{2}\right) + c$ (Continued)

Again, from the base case, $T(1) = c_1$, and $T\left(\frac{n}{2^k}\right) = T(1)$.

$$T(n) = (2^k)T\left(\frac{n}{2^k}\right) + kc$$

$$\implies T(n) = (2^k)c_1 + kc \tag{23}$$

$$\implies T(n) = (2^{log_2 n})c_1 + log_2 n \cdot c, \text{ as } k = log_2 n \tag{24}$$

$$\implies T(n) = (2^{log_2 n})c_1 + log_2 n \cdot c \text{ by inverse property} \tag{25}$$

$$\implies T(n) = n \cdot c_1 + log_2 n \cdot c, \text{ where } 2^{log_2 n} = n \text{ by the power rule} \tag{26}$$

Final equation:

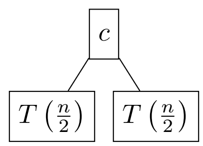
$$T(n) = n(c_1) + log_2 n(c), \text{ where } c_1 \text{ and } c \text{ are some constants}$$

$$T(n) \in \Theta(n)$$

Problem 1 (Continued)

1b) Recursion Tree of $T(n) = 2T\left(\frac{n}{2}\right) + c$

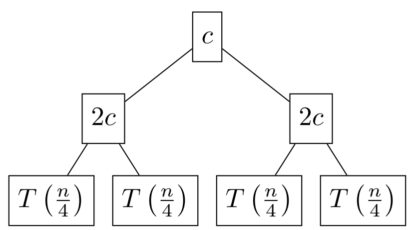
The recurrence relation $T(n)$ spends c cost of execution and makes two recursive calls $T\left(\frac{n}{2}\right)$.



The cost of execution is c for root node of size n, so:

$$\text{Root (Level 0): } c = n \text{ or } (2^0)c = n \tag{27}$$
$$\text{Level 1 Subtrees: } c = \frac{n}{2} \tag{28}$$
$$\implies 2c = n \text{ or } (2^1)c = n \tag{29}$$

The recurrence relation $T(n)$ spends $2c$ cost of execution and makes four recursive calls $T\left(\frac{n}{4}\right)$.



The cost of execution is $2c$ for two child nodes of size $\frac{n}{2}$, so:

$$\text{Root: } c = n \text{ or } (2^0) = n \tag{30}$$
$$\text{Level 1 Subtrees: } 2c = n \text{ or } (2^1)c = n \tag{31}$$
$$\text{Level 2 Subtrees: } c = \frac{n}{4} \tag{32}$$
$$\implies 4c = n \text{ or } (2^2)c = n \tag{33}$$

1b) Recursion Tree of $T(n) = 2T\left(\frac{n}{2}\right) + c$ (Continued)

At level k, there's $(2^k)c$ cost of execution. Consider the expansions:

$$\text{Root (Level 0): } T(n) \Longrightarrow T\left(\frac{n}{2^0}\right) \tag{34}$$

$$\text{Level 1 Subtrees: } \left(\frac{n}{2}\right) \Longrightarrow T\left(\frac{n}{2^1}\right) \tag{35}$$

$$\text{Level 2 Subtrees: } \left(\frac{n}{4}\right) \Longrightarrow T\left(\frac{n}{2^2}\right) \tag{36}$$

$$\text{Level } k \text{ Subtrees: } T(n) \Longrightarrow T\left(\frac{n}{2^k}\right) \tag{37}$$

Each level is the expansion and number of recursive calls plus that level's cost of execution.

$$T(n) = (2^k)T\left(\frac{n}{2^k}\right) + kc$$

At level k, base case $T(1)$ is an expansion of $T\left(\frac{n}{2^k}\right)$. Solve for k:

$$\frac{n}{2^k} = 1 \tag{38}$$

$$\Longrightarrow \quad n = 2^k \text{ by clearing the denominator} \tag{39}$$

$$\Longrightarrow log_2 n = log_2 2^k \text{ by logarithms} \tag{40}$$

$$\Longrightarrow log_2 n = k(log_2 2) \text{ by the power rule} \tag{41}$$

$$\Longrightarrow log_2 n = k, \text{ as } log_2 2 = 1 \tag{42}$$

From the base case, $T\left(\frac{n}{2^k}\right) = T(1) = c_1$.

$$T(n) = (2^{\log_2 n})c_1 + (\log_2 n)c \tag{43}$$

$$\Longrightarrow T(n) = (n)c_1 + (\log_2 n)c \text{ by the power rule} \tag{44}$$

$$T(n) \in \Theta(n) \tag{45}$$

1c) Master of $T(n) = 2T\left(\frac{n}{2}\right) + c$

Use the master method formula.

$$T(n) = aT\left(\frac{n}{b}\right) + f(n), \text{ where } a \geq 1, b \geq 1, \text{ and } f(n) \geq 0$$

Compare $f(n)$ to $n^{\log_b a}$, derived from subproblems when approaching the base case.

$$f(n) = c$$

From $T(n) = 2T\left(\frac{n}{2}\right) + c$, $a = 2$ and $b = 2$.

$$n^{log_b a} = n^{log_2 2} = n^1 = n$$

So, $n^{\log_b a}$ is linear, and $f(n)$ is constant. This meets Case 1.

$$c <<< n$$

$$T(n) = \Theta(n^{\log_b(a)}) \tag{46}$$
$$\implies T(n) = \Theta(n^{\log_2(2)}) \tag{47}$$
$$\implies T(n) = \Theta(n^1) \tag{48}$$
$$T(n) \in \Theta(n) \tag{49}$$

Problem 2

Solve the recurrence relations using any method. Find the time complexity, assuming base case $T(0) = 1$ and-or $T(1) = 1$.

a) $T(n) = 4T\left(\frac{n}{2}\right) + n$

Use the master method formula.

$$T(n) = aT\left(\frac{n}{b}\right) + f(n), \text{ where } a \geq 1, b \geq 1, \text{ and } f(n) \geq 0$$

Compare $f(n)$ to $n^{\log_b a}$, derived from subproblems when approaching the base case.

$$f(n) = n$$

From $T(n) = 4T\left(\frac{n}{2}\right) + n$, $a = 4$ and $b = 2$.

$$\implies n^{log_b a} = n^{log_2 4} = n^2$$

So, $n^{\log_b a}$ is quadratic, and $f(n)$ is linear. This meets Case 1.

$$n <<< n^2$$

$$\begin{aligned} & T(n) = \Theta(n^{\log_b(a)}) && (50) \\ \implies & T(n) = \Theta(n^{\log_2(4)}) && (51) \\ \implies & T(n) = \Theta(n^2) && (52) \\ & T(n) \in \Theta(n^2) && (53) \end{aligned}$$

Problem 2 (Continued)

b) $T(n) = 2T\left(\frac{n}{4}\right) + n^2$

Use the master method formula.

$$T(n) = aT\left(\frac{n}{b}\right) + f(n), \text{ where } a \geq 1, b \geq 1, \text{ and } f(n) \geq 0$$

Compare $f(n)$ to $n^{\log_b a}$, derived from subproblems when approaching the base case.

$$f(n) = n^2$$

From $T(n) = 2T\left(\frac{n}{4}\right) + n^2$, $a = 2$ and $b = 4$.

$$\implies n^{log_b a} = n^{log_4 2} = n^{\frac{1}{2}} = \sqrt{(}n)$$

So, $n^{\log_b a}$ is fractional, and $f(n)$ is quadratic. This meets Case 3.

$$n^2 >>> \sqrt{(}n)$$

$$T(n) = \Theta(f(n)) \tag{54}$$
$$\implies T(n) = \Theta(n^2) \tag{55}$$
$$T(n) \in \Theta(n^2) \tag{56}$$

Problem 3

Write a divide-and-conquer algorithm that finds the k^{th} indice in a combined, sorted array from two sorted arrays m and n.

a) Write the pseudocode for a function kthElement(Arr1, Arr2, k) that takes sorted arrays Arr1, Arr2 and position k as inputs and returns the item at the k^{th} indice.

b) Write the code in external file KthElement.py.

a) Pseudocode

Function kthElement() takes two arrays (Arr1, Arr2) and position k as inputs:
Concatenate Arr1, Arr2 as Arr
Call merge sort()

Function merge sort() takes the concatenated array and its start and end indices as inputs
If start is less than end:
Compute the middle indice
Recursively call the left array side, start to mid
Recursively call the right array side, mid to end
Call merge()

Function merge() takes the concatenated array, start, mid, and end as inputs.
After the recursive calls, inputs start, mid, and end serve as pointers for broken-down subarrays of one element apiece
Set temp arrays leftArr and rightArr of the same size
Using the start, mid, and end pointers, copy the subarray elements from the concatenated Arr to leftArr (start + indice) and rightArr (mid + 1 + indice)
Assign k as start, with leftIndice and rightIndice set to zero
"Merge" the two subarrays by reassigning the correct indices in the concatenated array
Increment k
Copy remaining elements, if any, to concatenated array

b) Code

See external file KthElement.py.