CSC236 Assignment #2

Yuen Wing Yau, Peiqi Wang

November 3, 2016

MarkUs.

Problem 1

Consider the Fibonacci function f:

$$f(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ f(n-2) + f(n-1) & \text{if } n > 1 \end{cases}$$

Prove $f(n-1) * f(m-1) + f(n+1) * f(m+1) + f(n)f(m) > f(n+m)$ for $n, m \geq 1$.

Proof.
Define predicate,

$$P(n, m) : f(n-1)f(m-1) + f(n+1)f(m+1) + f(n)f(m) > f(n+m)$$

We want to prove $P(n, m)$ holds for all $n, m \geq 1$. Use proof by complete induction on more than 1 variables follows,
Basis:
When $n = 1, m = 1$,

$$f(1-1)f(1-1) + f(1+1)f(1+1) + f(1)f(1) = 2 > 1 = f(1+1)$$

Therefore $P(1, 1)$ holds
When $n = 1, m = 2$,

$$f(1-1)f(2-1) + f(1+1)f(2+1) + f(1)f(2) = 3 > 2 = f(1+2)$$

Therefore $P(1, 2)$ holds
When $n = 2, m = 1$,

$$f(2-1)f(1-1) + f(2+1)f(2+1) + f(2)f(1) = 3 > 2 = f(2+1)$$

Therefore $P(2, 1)$ holds
Inductive step:

1. **Induction on n**

   Let fixed m be arbitrary such that $m \geq 1$ and $n \geq 3$. This is to make sure that $P(n-2, m)$ is well defined, which we will be using in the proof later. Let $i \in \mathbb{N}, 1 \leq i < n$, then prove $P(i, m) \Rightarrow P(n, m)$. Here the induction hypothesis is $P(i, m)$ (*I have just stated I.H. in a terse way... please dont deduct mark for brevity*)

   $$
   \begin{aligned}
   & f(n-1)f(m-1) + f(n+1)f(m+1) + f(n)f(m) \\
   =& (f(n-2) + f(n-3))f(m-1) + (f(n) + f(n-1))f(m+1) + (f(n-1) + f(n-2))f(m) \\
   & \qquad \text{(by recurrence relationship)} \\
   =& [f(n-2)f(m-1) + f(n)f(m+1) + f(n-1)f(m)] \\
   & \quad + [f(n-3)f(m-1) + f(n-1)f(m+1) + f(n-2)f(m)] \qquad \text{(arithmetics)} \\
   =& [f(a-1)f(m-1) + f(a+1)f(m+1) + f(a)f(m)] \\
   & \quad + [f(b-1)f(m-1) + f(b+1)f(m+1) + f(b)f(m)] \\
   & \qquad \text{(let } a = n-1 \text{ and } b = n-2\text{, then } 1 \leq a < n \text{ and } 1 \leq b < n \text{ as } n \geq 3) \\
   <& f(a+m) + f(b+m) \\
   & \qquad \text{(by Induction hypothesis, specifically, } P(a, m) \text{ and } P(b, m) \text{ holds)} \\
   =& f(n-1+m) + f(n-2+m) \qquad (a = n-1 \text{ and } b = n-2) \\
   =& f(n+m) \qquad \text{(by recurrence relationship)}
   \end{aligned}
   $$

   Therefore $P(n, m)$ holds. We have proved $\forall i \in \mathbb{N}, 1 \leq i < n$ then $P(i, m) \Rightarrow P(n, m)$

2. **Induction on m**

   Let fixed n be arbitrary such that $n \geq 1$ and $m \geq 3$. This is to make sure that $P(n, m-2)$ is well defined. Let $j \in \mathbb{N}, 1 \leq j < m$, then prove $P(n, j) \Rightarrow P(n, m)$. Here the induction hypothesis is $P(n, j)$

   $$
   \begin{aligned}
   & f(n-1)f(m-1) + f(n+1)f(m+1) + f(n)f(m) \\
   =& f(n-1)(f(m-2) + f(m-3)) + f(n+1)(f(m) + f(m-1)) + f(n)(f(m-1) + f(m-2)) \\
   & \qquad \text{(by recurrence relationship)} \\
   =& [f(n-1)f(m-2) + f(n+1)f(m) + f(n)f(m-1)] \\
   & \quad + [f(n-1)f(m-3) + f(n+1)f(m-1) + f(n)f(m-2)] \qquad \text{(arithmetics)} \\
   =& [f(n-1)f(a-1) + f(n+1)f(a+1) + f(n)f(a)] \\
   & \quad + [f(n-1)f(b-1) + f(n+1)f(b+1) + f(n)f(b)] \\
   & \qquad \text{(let } a = m-1 \text{ amd } b = m-2\text{, them } 1 \leq a < m \text{ and } 1 \leq b < m \text{ as } m \geq 3) \\
   <& f(a+n) + f(b+n) \\
   & \qquad \text{(by Induction hypothesis, specifically, } P(n, a) \text{ and } P(n, b) \text{ holds)} \\
   =& f(m-1+n) + f(m-2+n) \qquad (a = m-1 \text{ and } b = m-2) \\
   =& f(m+n) \qquad \text{(by recurrence relationship)}
   \end{aligned}
   $$

   Therefore $P(n, m)$ holds. We have proved $\forall j \in N, 1 \leq j < m$ then $P(n, j) \Rightarrow P(n, m)$

Therefore, by complete induction, we proved that $f(n-1)f(m-1) + f(n+1)f(m+1) + f(n)f(m) > f(n+m)$ holds for all $n, m \geq 1$

□

Problem 2

1. Find a recurrence relation, $T(n)$, for number of distinct full binary trees with n nodes. Show how you find the relation.

   Solution.
   The following lemma is proven in lecture notes, and I will not be prove it again,

   Any full binary tree has odd number of nodes

   Since full binary tree, then for a full binary tree with n nodes, let $k_n = \frac{n-1}{2}$ be the number of unique subtrees of the full binary tree with n nodes. It is easy to see that a single node does not have children, therefore $k_1 = 0$. For full binary tree with size $n > 1$, it is composed of right and left subtrees centered at root. Hence right and left subtrees has a total of $n-1$ nodes. By lemma mentioned previously, n is odd, therefore $n-1$ is even. The subtrees can be composed of any number of odd nodes i such that $i < n-1$. Because there are $\frac{n-1}{2}$ different odd numbers in $n-1$. Then we proved $k_n = \frac{n-1}{2}$ for $n > 1$. Here we have shown that k_n represent number of unique subtrees of full binary tree with n nodes

   When $n = 1$, the full binary tree is just one node and therefore $T(n) = 1$. When $n > 1$, a full binary tree can be decomposed into an arbitrary left and right subtrees. Let size of left tree be n_l, then the size of right tree is $n_r = n - 1 - n_l$. We can then recursively determine the number of distinct full binary trees for the arbitrary left and right subtrees as $T(n_l)$ and $T(n_r)$. Here, $T(n_l)T(n_r)$ represents the number of unique full binary tree for a fixed n_l number of nodes in the left subtree. And since there is $k_n = \frac{n-1}{2}$ possible unique subtrees for a full binary tree of size n proved earlier, the number of distinct full binary tree is the summation of $T(n_l)T(n_r)$ for $n_l \in \{2x-1 : x \in \{1...k_n\}\}$, or symbolically

$$\begin{aligned} T(n) &= \sum_{i=1}^{k_n} T(2i-1)T(n-1-(2i-1)) \\ &= \sum_{i=1}^{\frac{n-1}{2}} T(2i-1)T(n-2i)) \end{aligned}$$

Therefore,

$$T(n) = \begin{cases} 1, & n = 1 \\ \sum_{i=1}^{\frac{n-1}{2}} T(2i-1)T(n-2i)), & n > 1 \text{ n is odd} \end{cases}$$

Remark. Note that a full binary tree with size n has odd number of nodes.

□

2. Without using a closed form, prove $T(n) \geq 2^{(n-1)/2}$ for most odd numbers.

Proof.
Define predicate,

$$P(n) : T(n) \geq 2^{\frac{n-1}{2}}$$

Basis:
When $n = 11$, $T(11) = T(1)T(9) + T(3)T(7) + T(5)T(5) + T(7)T(3) + T(9)T(1) = 42 \geq 32 = 2^{\frac{11-1}{2}}$. Proved $P(11)$ holds.
When $n = 13$, $T(13) = T(1)T(11) + T(3)T(9) + T(5)T(7) + T(7)T(5) + T(9)T(3) + T(11)T(1) = 132 \geq 64 = 2^{\frac{13-1}{2}}$. Proved $P(13)$ holds.
Inductive step:
When $n \geq 13$ and n is odd, Prove that $P(n-2) \Rightarrow P(n)$ by simple induction. (Picked $n-2$ as the previous step because the recurrence relation only works for odd numbers mentioned previously) Here the induction hypothesis is $P(n-2)$, or $T(n-2) \geq 2^{\frac{n-3}{2}}$

$$\begin{aligned} T(n) &= \sum_{i=1}^{\frac{n-1}{2}} T(2i-1)T(n-2i)) \\ &\geq T(1)T(n-2) + T(n-2)T(1) && \text{(take out first and last term)} \\ &= 2T(1)T(n-2) \\ &= 2T(n-2) && (T(1) = 1) \\ &\geq 2(2^{\frac{n-3}{2}}) && \text{(by I.H., } P(n-2) \text{ holds)} \\ &= 2^{\frac{n-1}{2}} \end{aligned}$$

Hence $T(n) \geq 2^{\frac{n-1}{2}}$, or $P(n)$ holds. We proved $P(n-2) \Rightarrow P(n)$
By simple induction, we proved that $T(n) \geq 2^{\frac{n-1}{2}}$ holds for all odd size $n \geq 11$ □

Problem 3

1. Find a recurrence relation, $T(n)$, for number of microorganisms in a microbial culture in which every 2 hours the number of microorganisms is quadrupled and also three

times as many of the microorganisms die 4 hours after creation. There are initially 4 microorganisms in the culture.

Solution. Since number of microorganism changes only upon every 2 hour mark, we can model the growth of microorganism every two hours. Let $n = 2t$, where t is the number of hours elapsed. $T(0) = 4$ because there are initially 4 microoganisms. $T(1) = 16$ because the number of microorganisms quadrupled from 4 to 16 after 2 hours. For $n \geq 2$, $T(n) = 4T(n-1) - 3T(n-2)$ because the number of microoganisms both quadruple from the number 2 hours earlier and decrease by three times of the number of microogansims at $n = 2$, i.e. $t = 4$ hours earlier.

$$T(n) = \begin{cases} 4, & n = 0 \\ 16, & n = 1 \\ 4T(n-1) - 3T(n-2), & n \geq 2 \end{cases}$$

□

2. Without using a closed form, prove $T(n)$ is strictly increasing.

Proof.
Define,

$$P(n) : T(n) > T(n-1)$$

Basis:
When $n = 1$, $T(1) = 16 > 4 = T(0)$. Then $P(1)$ holds.
Wehn $n = 2$, $T(2) = 4T(2-1) - 3T(2-2) = 52 > 16 = T(1)$, Then $P(2)$ holds.
Inductive step:
When $n \geq 2$. Let $j \in \mathbb{N}$ such that $1 \leq j < n$, prove $P(j) \Rightarrow P(n)$. Here the induction hypothesis is $T(j) > T(j-1)$ for all j.

$$\begin{aligned} T(n) &= 4T(n-1) - 3T(n-2) \\ &= T(n-1) + 3(T(n-1) - T(n-2)) \\ &> T(n-1) \qquad \text{(by I.H., } P(n-1) \text{ holds as } 1 \leq n-1 < n) \end{aligned}$$

Proved $T(n) > T(n-1)$, therefore $P(n)$ holds. Proved $\forall j \in \mathbb{N}, 1 \leq j < n$, then $P(j) \Rightarrow P(n)$
Therefore by complete induction, $T(n) > T(n-1)$ for all $n \in \mathbb{N}$, and equivalently, $T(n)$ is strictly increasing because every element is larger than the previous in the recurrence relationship. □

3. Compute the closed form of $T(n)$ using unwinding (repeated substitution).

Solution.
Assume n is sufficiently large.

$$\begin{aligned}
T_c(n) &= 4T(n-1) - 3T(n-2) && \text{(by recurrence relation)}\\
&= 4(4T(n-2) - 3T(n-3)) - 3T(n-2) && \text{(by recurrence relation)}\\
&= 13T(n-2) - 12T(n-3)\\
&= (1 + 3^1 + 3^2)T(n-2) - (3^1 + 3^2)T(n-3)\\
&= 13(4T(n-3) - 3T(n-4)) - 12T(n-3) && \text{(by recurrence relation)}\\
&= 40T(n-3) - 39T(n-4)\\
&= (1 + 3^1 + 3^2 + 3^3)T(n-3) - (3^1 + 3^2 + 3^3)T(n-4)\\
&\vdots\\
&= (1 + \sum_{i=1}^{n-1}(3^i))T(1) - \sum_{i=1}^{n-1}(3^i)T(0)\\
&= (1 + \sum_{i=1}^{n-1}(3^i))16 - \sum_{i=1}^{n-1}(3^i)4 && (T(1) = 16,\ T(0) = 4)\\
&= 12\sum_{i=1}^{n-1} 3^i + 16
\end{aligned}$$

In summary,

$$T_c(n) = \begin{cases} 4, & n = 0 \\ 16, & n = 1 \\ 12\sum_{i=1}^{n-1} 3^i + 16, & n \geq 2 \end{cases}$$

□

4. Prove the closed form, computed in part (c), is correct.

Proof.
Define predicate P,

$$P(n) : T(n) = T_c(n) \qquad (\text{ where } T_c(n) \text{ is the closed form of } T(n))$$

Basis:
When $n = 0$, $T(0) = 4 = T_c(0)$ as defined. Then $P(0)$ holds.
When $n = 1$, $T(1) = 16 = T_c(1)$ as defined. Then $P(1)$ holds
When $n = 2$, $T(2) = 4T(1) - 3T(0) = 52 = 12(3^1) + 16 = T_c(2)$ as defined. Then $P(2)$ holds

When $n = 3$, $T(3) = 4T(2) - 3T(1) = 160 = 12(3^1 + 3^2) + 16 = T_c(3)$ as defined. Then $P(3)$ holds

Inductive step:

Let $n \geq 4$. Let $k \in \mathbb{N}$ such that $2 \leq k < n$. This is to ensure that expanded recurrence relation of $T(n)$ is valid **and** that substitution with closed form yields valid summation upper bounds. $P(k)$ is the induction hypothesis, specifically, $T(k) = T_c(k) = 12\sum_{i=1}^{k-1} 3^i + 16$. We prove that $P(k) \Rightarrow P(n)$

$$
\begin{aligned}
T(n) &= 4T(n-1) - 3T(n-2) && \text{(by recurrence relationship)} \\
&= 4(12\sum_{i=1}^{n-2} 3^i + 16) - 3(12\sum_{j=1}^{n-3} 3^j + 16) \\
&\qquad \text{(by I.H. as } 2 \leq n-2 < n \text{ and } 2 \leq n-1 < n \text{ because } n \geq 4) \\
&= 12(4\sum_{i=1}^{n-2} 3^i - 3\sum_{j=1}^{n-3} 3^j) + 16 \\
&= 12(3 \times 4 + \sum_{i=2}^{n-2} 4 \times 3^i - \sum_{i=2}^{n-2} 3^i) + 16 \\
&\text{(extract term at } i = 2 \text{ for the first summation and let } i = j + 1 \text{ in the second summation)} \\
&= 12(3^1 + 3^2 + \sum_{i=2}^{n-2} 3^{i+1}) + 16 && \text{(by summation identity)} \\
&= 12(3^1 + 3^2 + \sum_{m=3}^{n-1} 3^m) + 16 && \text{(let } m = i + 1\text{)} \\
&= 12\sum_{m=1}^{n-1} 3^m + 16 \\
&= T_c(n)
\end{aligned}
$$

Therefore $P(n)$. we have proved whenever $n \geq 4$ and $2 \leq k < n$, $P(k) \Rightarrow P(n)$

By complete induction, $T_c(n) = T(n)$ holds for all $n \in \mathbb{N}$. Hence, the closed form computed is correct. □

Problem 4

1. Find a recurrence relation, $T(n)$, for number of distinct ways that a postage of n cents can be made by 4-cent, 6-cent, and 10-cent stamps for most even numbers.

   Solution.

   We first realize that the domain of $T(n)$ is strictly even, this is because summation

of any possible even numbers, 4, 6, and 10 in this case, cannot be odd. First we identify the base cases for even numbers within the range (0, 10). Then for $n \geq 12$, we can think of the unique combination of postage n as a recussive relation of number of unique combinations for a smaller n. There are 3 ways to simplify any arbitrary postage of n cents where $n \geq 12$, namely recursively compute the unique numbers of postage of size $n-4$, $n-6$, and $n-10$ and represent the postage of size n with $T(n-4)$, $T(n-6)$, and $T(n-10)$. We first realize that $T(n-10)$ is redundant as it is encompassed within both $T(n-4)$ and $T(n-6)$. More specifically, we can first consider postage of size $m = n-4$ and then $m-6$, or vice versa, to arrive at the same point when considering directly a postage of size $n-10$. As the question implies that order of the postage does not matter, it is in fact necessary to remove redundancies arised from the two alternative route to postage of size $n-10$. We can take this into consideration by subtracting $T(n-10)$ from the summation of $T(n-4)$ and $T(n-6)$. The recurrence relation represent *most even numbers*,

$$T(n) = \begin{cases} 0, & n = 0 \\ 0, & n = 2 \\ 1, & n = 4 \\ 1, & n = 6 \\ 1, & n = 8 \\ 2, & n = 10 \\ T(n-4) + T(n-6) - T(n-10), & n \geq 12 \wedge \text{ n is not multiple of 10} \end{cases}$$

There is a special case, however, when n is a multiple of 10. In this case, we add 1 to the expression, which is equivalent to adding $\lfloor 1 - \frac{n \mod 10}{10}) \rfloor$. I dont know why but this is an obvious pattern on the python output...

$$T(n) = \begin{cases} 0, & n = 0 \\ 0, & n = 2 \\ 1, & n = 4 \\ 1, & n = 6 \\ 1, & n = 8 \\ 2, & n = 10 \\ T(n-4) + T(n-6) - T(n-10) + \lfloor 1 - \frac{n \mod 10}{10} \rfloor, & n \geq 12 \end{cases}$$

□

2. Without using a closed form, prove $T(n)$ is non-decreasing.

Proof. Here we prove the first recurrence relation in previous question.
Define

$$P(n) : T(n) \geq T(n-2)$$

Basis:

(a) $n = 2$, $T(2) = 0 \geq 0 = T(0)$, then $P(2)$ holds.
(b) $n = 4$, $T(4) = 1 \geq 0 = T(2)$, then $P(4)$ holds.
(c) $n = 6$, $T(6) = 1 \geq 1 = T(4)$, then $P(6)$ holds.
(d) $n = 8$, $T(8) = 1 \geq 1 = T(6)$, then $P(8)$ holds.
(e) $n = 10$, $T(10) = 2 \geq 1 = T(8)$, then $P(10)$ holds.
(f) $n = 12$, $T(12) = T(8) + T(6) - T(2) = 2 \geq 2 = T(10)$, then $P(12)$ holds.

Inductive step:
Let $n \geq 14$, let $j \in \mathbb{N}$, such that $2 \leq j < n$, hen $P(j)$, or $T(j) \geq T(j-2)$, holds. Prove that $P(j) \Rightarrow P(n)$.

$$\begin{aligned} T(n) &= T(n-4) + T(n-6) - T(n-10) \\ &= T(n-8) + T(n-10) - T(n-14) + T(n-6) - T(n-10) \\ &\qquad \text{(by recurrence relation)} \\ &= T(n-6) + T(n-8) - T(n-14) \\ &\geq T(n-6) + T(n-8) - T(n-12) \\ &\text{(by I.H. } P(n-12), i.e. T(n-12) \geq T(n-14), \text{ holds because } 2 \leq n-2 < n \text{ as } n \geq 14) \\ &= T(n-2) \end{aligned}$$

Hence $P(n)$ holds. Therefore, for $n \geq 14$ and $2 \leq j < n$, $P(j) \Rightarrow P(n)$
By complete induction $P(n)$ holds for all even numbers $n \geq 0$. Then since every element is larger or equal than the previous element, $T(n)$ is non-decreasing.

□

Problem 5

1. Devise a brute-force algorithm in Python notation, **max-sum**, to find the largest sum of consecutive terms of a sequence of n positive and negative numbers.

```
1    def MaxSumBrute(array):
2        """ Brute force method for finding largest sum for contiguous
     subarrays of positive and negative number array of size n
3
4        @param: Array array: target array
5        @rparam: int maxSum: max sum of contiguous subarrays
6        """
```

```
7         # set default max sum
8         # true because array consists of both negative and positive
      numbers
9         maxSum = 0
10        for i in xrange(0, len(array)):
11            startPosition = i
12            endPosition = startPosition + 1
13            while(endPosition <= len(array)):
14                currentSubarraySum = sum(array[startPosition: endPosition
      ])
15                if(currentSubarraySum > maxSum):
16                    maxSum = currentSubarraySum
17                endPosition += 1
18        return maxSum
19
```

2. Find the worst-case time complexity of **max-sum**.

Solution.
Let $T_b(n)$ be the worst case time required to run the brute force max sum algorithm on an array of length n. The algorithm runs line 9 and 18 once as well as one guard step for the outer loop. The outer for-loop runs n iterations. Let i be the index of each run from 0 to $n-1$, then the for-loop requires a constant a constant of 4 time to execute line 10-13. The inner while-loop runs $5(n-i)$ amount of time.

$$\begin{aligned} T_b(n) &= 2 + 1 + \sum_{i=0}^{n-1}(3 + 1 + 5(n-i)) \\ &= 3 + \sum_{i=0}^{n-1}(4 + 5n - 5i) \\ &= 3 + 4n + 5n^2 - 5\frac{n(n+1)}{2} \\ &= \frac{5}{2}n^2 + \frac{3}{2}n + 3 \end{aligned}$$

Now we prove $T_b(n) \in \mathcal{O}(n^2)$,
Let $c_0 = 7$ and $N_0 = 1$, let $n \geq N_0$, then

$$\begin{aligned} T_b(n) &= \frac{5}{2}n^2 + \frac{3}{2}n + 3 \\ &\leq \frac{5}{2}n^2 + \frac{3}{2}n^2 + 3n^2 && (n \geq N_0 = 1) \\ &= c_0 n^2 && (c_0 = 7) \end{aligned}$$

Then $\exists c \in \mathbb{R}, N \in \mathbb{N}, \forall n \geq N, T_b(n) \leq cn^2$.
Hence $T_b(n) \in \mathcal{O}(n^2)$

□

3. Devise a divide-and-conquer algorithm to find **max-sum**, by splitting the sequence to halves, and finding **max-sum** in each. Note that the maximum sum of consecutive terms can include terms in both halves.

   Solution.
   We can divide the input array into two halves and call MAX-SUM recursively on both subarrays. Here we get the max-sum for arrays completely contained in both the left and the right subarray. Then we compute the max sum for arrays that overlaps the left and the right subarray. The idea is that we find the max sum going from the center of the array to either the left or the right side, and then summing the max sum from both sides. Then we compute and return the max sum considering the three cases mentioned above. Details are implemented below.

```
    def MaxSumDivCon(array):
        """ Divide and conquer method for finding largest sum for
contiguous subarrays of positive and negative number array of size n

        @param: Array array: target array
        @rparam: int maxSum: max sum of contiguous subarrays
        """
        n = len(array)

        if (n == 1):
            return array[0]

        m = n / 2
        leftMaxSum = MaxSumDivCon(array[:m])
        rightMaxSum = MaxSumDivCon(array[m:])

        # find max sum of overlapping subarray starting from the middle
        leftStart = m - 1
        leftPartMaxSum = array[m - 1]
        while (leftStart >= 0):
            leftPartSum = sum(array[leftStart: m])
            if (leftPartSum > leftPartMaxSum):
                leftPartMaxSum = leftPartSum
            leftStart -= 1

        rightEnd = m
        rightPartMaxSum = array[m]
        while (rightEnd <= n):
            rightPartSum = sum(array[m: rightEnd])
            if (rightPartSum > rightPartMaxSum):
                rightPartMaxSum = rightPartSum
            rightEnd += 1
```

```
32
33
34          return max(leftMaxSum, rightMaxSum, (leftPartMaxSum +
    rightPartMaxSum))
35
```

□

4. Find a recurrence relation for the worst-case time complexity of the divide-and-conquer **max-sum**.

   Solution.

$$T_d(n) = \begin{cases} c_1, & n = 1 \\ T_d(\lfloor \frac{n}{2} \rfloor) + T_d(\lceil \frac{n}{2} \rceil) + c_2 n, & n > 1 \end{cases}$$

   If $n = 1$, then the function terminates till line 10, which takes a constant time c_1. If $n > 1$, we first take into account the two recursive calls taking $T_d(\lfloor \frac{n}{2} \rfloor)$ and $T_d(\lceil \frac{n}{2} \rceil)$ time respectively. The subsequent steps to compute the max sum of subarrays contained in both halves takes time proportional to n, which we denote as $c_2 n$. Hence the recurrence relation. □

5. How does the time complexity of the the divide-and-conquer algorithm compare with that of the brute-force one?

   Solution.
   We can rewrite $T_d(n)$, without altering its complexity, as $T_d(n) = 2T_d(\frac{n}{2}) + c_2 n$. Since $2 = 2^1$, $T(n) \in \mathcal{O}(n \log_2 n)$ by master theorem, for some c_3. In contrast, the brute force max-sum has $T_b(n) \in \mathcal{O}(n^2)$. We know that $\mathcal{O}(n^2)$ has a worse time complexity than $\mathcal{O}(n \log_2 n)$. We can say that the divide and conquer method for finding max sum of array is theoretically faster than the brute force method. □