1. Let $T(n)$ denote the worst-case running time of the algorithm below on inputs of size n.

```
# Precondition: A is a non-empty list of integers, i is a natural number.
def recSS(A, i):
1.      if i < len(A) - 1:
2.          small = i
3.          for j in range(i + 1, len(A)):
4.              if A[j] < A[small]:
5.                  small = j
6.          temp = A[i]
7.          A[i] = A[small]
8.          A[small] = temp
9.          recSS(A, i + 1)
```

Note that the above algorithm has an implicit base case, for which it does nothing.

Recall the recurrence relation that we saw last tutorial satisfied by T. For any natural number n, let $T(n)$ denote the maximum number of steps executed by a call to recSS(A, i), where $n = \text{len}(A) - i$. We get the following definition for $T(n)$:

$$T(n) = \begin{cases} a, & n \leq 1 \\ T(n-1) + b(n-1) + c, & n \geq 2 \end{cases}$$

Give an asymptotic tight-bound for the worst-case running time of the algorithm.

Solution Elements

Assume $n \geqslant 2$. Then

$$\begin{aligned} T(n) &= T(n-1) + b(n-1) + c \\ &= [T(n-2) + b(n-2) + c] + b(n-1) + c \\ &= T(n-2) + b((n-1) + (n-2)) + 2c \\ &= [T(n-3) + b(n-3) + c] + b((n-1) + (n-2)) + 2c \\ &= T(n-4) + b((n-1) + (n-2) + (n-3)) + 3c \end{aligned}$$

It seems that after i $(1 \leqslant i \leqslant n-1)$ applications of the recursive definition we have

$$T(n) = T(n-i) + b((n-1) + (n-2) + (n-3) + \ldots + (n-i)) + i \cdot c$$

Therefore, after $n-1$ applications of the recursive definition we have

$$\begin{aligned} T(n) &= T(1) + b((n-1) + (n-2) + (n-3) + \cdots + 1) + (n-1) \cdot c \\ &= a + b\frac{(n-1)n}{2} + c(n-1) = \frac{b}{2}n^2 + (c - \frac{b}{2})n + a - c. \end{aligned}$$

Note that in a test/assignment you are expected to prove the correctness of the closed-form expression you obtained for T by induction.

Finally, we can conclude that $T(n) \in \Theta(n^2)$.

2. The following algorithm performs *breadth-first search* on a *perfect binary search trees*. A *binary search tree* is a binary tree which stores a value $u.val$ for every node u. The values satisfy: for all nodes s in the subtree of u rooted at its left-child, $s.val < u.val$, and for all nodes t in the subtree of u rooted at its right-child. $t.val > u.val$. *You can assume that the values are all unique.* Further, such a tree is *perfect* if each internal node has both a left *and* a right child. *We want to know if x appears as a value on some node of the tree.*

Let $T(n)$ denote the worst-case running time of the algorithm on trees with n nodes.

```
# Precondition: r is the root of a perfect balanced binary search tree.
def bfs(r, x):
1.      if r is None:
2.          return False
3.      if r.val == x:
4.          return True
5.      if x < r.val:
6.          return bfs(r.left, x)
7.      else:
8.          return bfs(r.rigth, x)
```

Recall the recurrence relation that we saw last tutorial satisfied by T. For any natural number n, let $T(n)$ denote the maximum number of steps executed by a call to bfs(r, x), where r is the root of a perfect balanced binary search tree with n nodes. We get the following definition for $T(n)$:

$$T(n) = \begin{cases} O(1), & n = 0 \\ T(n/2) + O(1), & n \geqslant 1 \end{cases}.$$

Give an asymptotic tight-bound for the worst-case running time of the algorithm.

Solution Elements

Recall that the Master method states that for recurrence relations of the form $T(n) = aT\left(\frac{n}{b}\right) + f(n)$, three cases are possible:

$$\text{If } f(n) = \Omega\left(n^{\log_b a + \epsilon}\right) \text{ for } \epsilon > 0, T(n) = \Theta\left(f(n)\right).$$
$$\text{If } f(n) = O\left(n^{\log_b a - \epsilon}\right) \text{ for } \epsilon > 0, T(n) = \Theta\left(n^{log_b a}\right).$$
$$\text{If } f(n) = \Theta\left(n^{\log_b a}\right), T(n) = \Theta\left(n^{log_b a} \log n\right),$$

Plugging in $a = 1$, $b = 2$, and $f(n) = O(1)$, the third case is true and we have that $T(n) = \Theta(\log n)$.

3. Give an asymptotic tight-bound for each of the following functions.

(a)

$$T_2(n) = \begin{cases} a, & n = 1 \\ 2T_2(n/2) + 4n, & n \geqslant 2 \end{cases}$$

Solution Elements
We will use the Master Theorem.

Here, $a = 2, b = 2$, and $f(n) = n$. Since $\log_2 2 = 1$ (equivalently, $2 = 2^1$), by the Master Theorem, $T_2(n) \in \Theta(n \log n)$.

(b)

$$T_3(n) = \begin{cases} a, & n = 1 \\ 2T_3(n/7) + \log n + \sqrt{n}, & n \geqslant 2 \end{cases}$$

Solution Elements
We will use the Master Theorem.

Here $a = 2$, $b = 7$, and $f(n) = \log n + \sqrt{n} \in \Theta(\sqrt{n})$. So, $f(n) = n^k$ where $k = \frac{1}{2}$. Since $2 < 7^{\frac{1}{2}}$ we have $\log_7 2 < \frac{1}{2}$. By the Master Theorem, $T_3(n) \in \Theta(\sqrt{n})$.