CSC236H
Introduction to the Theory of Computation

Theorem (Master Theorem). Let $T : \mathbb{N} \to \mathbb{R}^+$ be a recursively defined function with recurrence relation

$$T(n) = c\,T(\frac{n}{d}) + f(n)$$

for some constants $c, d \in \mathbb{Z}^+$, $d > 1$, and $f : \mathbb{N} \to \mathbb{R}^+$.
Furthermore, suppose $f(n) \in \Theta(n^k)$ for some $k \in \mathbb{R}, k \geq 0$. Then:

1. if $k = \log_d c$, then $T(n) \in \mathcal{O}(n^k \log n)$;
2. if $k < \log_d c$, then $T(n) \in \mathcal{O}(n^{\log_d c})$;
3. if $k > \log_d c$, then $T(n) \in \mathcal{O}(n^k)$.

- **Example:**

$$T(n) = \begin{cases} a, & n = 1 \\ T(\frac{n}{2}) + 2^n, & n \geq 2 \end{cases}$$

Theorem (Master Theorem). Let $T : \mathbb{N} \to \mathbb{R}^+$ be a recursively defined function with recurrence relation

$$T(n) = c\,T(\frac{n}{d}) + f(n)$$

for some constants $c, d \in \mathbb{Z}^+$, $d > 1$, and $f : \mathbb{N} \to \mathbb{R}^+$.
Furthermore, suppose $f(n) \in \Theta(n^k)$ for some $k \in \mathbb{R}, k \geq 0$. Then:

1. if $k = \log_d c$, then $T(n) \in \mathcal{O}(n^k \log n)$;
2. if $k < \log_d c$, then $T(n) \in \mathcal{O}(n^{\log_d c})$;
3. if $k > \log_d c$, then $T(n) \in \mathcal{O}(n^k)$.

- **Example:**

$$T_1(n) = \begin{cases} a, & n = 1 \\ 2T(\frac{n}{2}) + \log_2 n, & n \geq 2 \end{cases}$$

Divide-and-Conquer Algorithms

```
divide-and-conquer(P):
    if P has "small enough" size:
        solve P directly
    else:
        # Solve c sub-problems of the same size, recursively
        for i from 1 to c:
        # Solve the sub-problem P_i recursively
            s_i = divide_and_conquer(P_i)
        combine(s_1, ..., s_c)
```

Divide-and-Conquer Algorithms

```
""" Given a non-empty list A, sort the list in non-decreasing order. """
def MergeSort(A):
    if len(A) == 1:
        return A
    else:
        m = len(A) // 2  # integer division
        L1 = MergeSort(A[0..m-1])
        L2 = MergeSort(A[m..len(A)-1])
        return merge(L1, L2)
def merge(A, B):
    i = 0
    j = 0
    C = []
    while (i < len(A)) and (j < len(B)):
        if (A[i] <= B[j]):
            C.append(A[i]) # Add A[i] to the end of C
            i += 1
        else:
            C.append(B[j])
            j += 1
    return C + A[i..len(A)-1] + B[j..len(B)-1] # List concatenation
```

Theorem (Master Theorem). Let $T : \mathbb{N} \to \mathbb{R}^+$ be a recursively defined function with recurrence relation

$$T(n) = c\,T(\frac{n}{d}) + f(n)$$

for some constants $c, d \in \mathbb{Z}^+$, $d > 1$, and $f : \mathbb{N} \to \mathbb{R}^+$.
Furthermore, suppose $f(n) \in \Theta(n^k)$ for some $k \in \mathbb{R}, k \geq 0$. Then:

1. if $k = \log_d c$, then $T(n) \in \mathcal{O}(n^k \log n)$;
2. if $k < \log_d c$, then $T(n) \in \mathcal{O}(n^{\log_d c})$;
3. if $k > \log_d c$, then $T(n) \in \mathcal{O}(n^k)$.

- **Example:** Suppose $rec\_func\_1(n)$ and $rec\_func\_2(n)$ are two different algorithms for solving the same problem. Suppose T_1 represents the running-time complexity of $rec\_func\_1(n)$, and T_2 represents the running-time complexity of $rec\_func\_2(n)$, where

$$T_1(n) = \begin{cases} a_1, & n = 1 \\ 4T_1(\frac{n}{2}) + b_1, & n \geq 2 \end{cases} \qquad T_2(n) = \begin{cases} a_2, & n = 1 \\ 3T_2(\frac{n}{2}) + n, & n \geq 2 \end{cases}$$

Which algorithm should we choose?

Exercises

1. Analyze the running time of each of the following recursive algorithms.

a)

```
def sum(A):
    if len(A) == 0:
        return 1
    else:
        return A[0] + sum(A[1..len(A)-1])
```

b)

```
def fun(A):
    if len(A) < 2:
        return len(A) == 0
    else:
        return fun(A[2..len(A)-1])
```

c)

```
def double_fun(A):
    n = len(A)
    if n < 2:
        return n
    else:
        return double_fun(A[0..n-2]) + double_fun(A[1..n-1])
```

2. Find a closed-form expression for the each of the following functions. Also use the Master Theorem to find an asymptotic upper-bound for each of them.

   a) In your calculations and proof, you may assume that n is a power of 3.

$$T_1(n) = \begin{cases} a, & n = 1 \\ T_1(\frac{n}{3}) + 100, & n \geq 2 \end{cases}$$

   b) In your calculations and proof, you may assume that n is a power of 2.

$$T_2(n) = \begin{cases} a, & n = 1 \\ 4T_2(\frac{n}{2}) + n^{1.5} - 1, & n \geq 2 \end{cases}$$

3. Find an asymptotic upper-bound for the following function

$$T(n) = \begin{cases} a, & n = 1 \\ 2T(\frac{n}{2}) + \log_{10} n, & n \geq 2 \end{cases}$$