CSC236H

Introduction to the Theory of Computation

Running-Time Complexity of Algorithms - Review

- Measure running time by counting **steps** in algorithm: arithmetic operations, assignments, array accesses, comparisons, return statements, etc.
- Measure as a function $T(n)$ of **input size** n.
- $T(n)$ measures worst-case running time: maximum number of steps over all inputs of size n.
- We don't care about exact step counts, an estimate for $T(n)$ is sufficient:
  - every **chunk** of instructions is represented by a constant.
  - **chunk:** sequence of instructions that always gets executed together.
- Most of the time, NO simple algebraic expression for $T(n)$. Instead we prove bounds on $T(n)$, using asymptotic notation.
- **Upper bound:** $T(n) \in \mathcal{O}(f(n))$.
- **Lower bound:** $T(n) \in \Omega(f(n))$.
- **Tight bound:** $T(n) \in \Theta(f(n))$:
  $T(n) \in \mathcal{O}(f(n))$ and $T(n) \in \Omega(f(n))$.

Some Rules for Big-Oh Notation - Review

- If $T(n)$ is a **polynomial** of degree k, then $T(n) \in \mathcal{O}(n^k)$.

- An $\mathcal{O}(1)$ algorithm is **more** efficient than an $\mathcal{O}(\log n)$ algorithm,
  which is **more** efficient than an $\mathcal{O}(n)$ algorithm,
  which is **more** efficient than an $\mathcal{O}(n \log n)$ algorithm,
  which is **more** efficient than an $\mathcal{O}(n^2)$ algorithm,
  which is **more** efficient than an $\mathcal{O}(2^n)$ algorithm.

- If $T(n) = g(n) + f(n)$, and $f(n)$ is **less** efficient than $g(n)$, then $T(n) \in \mathcal{O}(f(n))$.

Some Rules for Big-Oh Notation - Review

```
""" Given a list A, return an index i such that x == A[i].
Otherwise, return -1."""
    def LS(A,x):
    1.      i = 1
    2.      while i < len(A):
    3.          if A[i] == x:
    4.              return i
    5.          i = i + 1
    6.      return -1
```

Running-Time Complexity of Algorithms: Recursive Algorithms

```
""" Given a natural number n>=1, return n!"""
    def fact(n):
1.      if n == 1:
2.          return 1
3.      else:
4.          return n * fact(n-1)
```

Closed-Form Formulas

A **closed-form** formula:

- allows to calculate a function by applying a **fixed** number of **basic** operations to its argument(s):
  - Basic Operations: addition, subtraction, multiplication, division, and exponentiation.

- $n^3 + n - 1$.
- $1 + 2 + ... + n$.
- $f(n-1) + f(n-2)$.

Closed-Form of Recursive Functions – Example

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

Closed-Form of Recursive Functions – Example

$$f(n) = \begin{cases} 0 & n = 0 \\ f(n-1) + 2n - 1 & n \geq 1 \end{cases}$$

Example – Running-Time Complexity of Recursive Binary Search

```
""" Given a sorted list A of length at least 1, return t such that
start<= t <= end and A[t] == x, if such a t exists;
otherwise return -1."""
def RecBinSearch(A, start, end, x):
1.    if (start == end) then:
2.       if (A[start] == x) then:
3.          return start
4.       else:
5.          return -1
6.    else:
7.       m = (start + end) // 2            # integer division
8.       if (A[m] >= x) then:
9.          return RecBinSearch(A, start, m, x)
10.      else:
11.         return RecBinSearch(A, m + 1, end, x)
```

Example – Running-Time Complexity of Recursive Binary Search

```
""" Given a sorted list A of length at least 1, return t such that
start<= t <= end and A[t] == x, if such a t exists;
otherwise return -1."""
def RecBinSearch(A, start, end, x):
1.    if (start == end) then:
2.       if (A[start] == x) then:
3.          return start
4.       else:
5.          return -1
6.    else:
7.       m = (start + end) // 2              # integer division
8.       if (A[m] >= x) then:
9.          return RecBinSearch(A, start, m, x)
10.      else:
11.         return RecBinSearch(A, m + 1, end, x)
```

Example – Running-Time Complexity of Recursive Binary Search

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

IMPORTANAT: In asymptotic analysis, we may ignore floors and ceilings.
Thus, we may always assume that the size of the input is so that the recursive calls always divide the input list evenly. (Take Home Exercise: convince yourself why)

$\rightarrow$ In the running-time analysis of binary search, we assume that n is a power of 2:
There exists some j s.t. $n = 2^j$.

Example – Running-Time Complexity of Recursive Binary Search

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

There exists some j s.t. $n = 2^j$.

Example – Running-Time Complexity of Merge Sort

```
""" Given a non-empty list A, sort the list in non-decreasing order. """
    def MergeSort(A):
2.      if len(A) == 1:
3.          return A
4.      else:
5.          m = len(A) // 2  # integer division
6.          L1 = MergeSort(A[0..m-1])
7.          L2 = MergeSort(A[m..len(A)-1])
8.          return merge(L1, L2)
    def merge(A, B):
9.      i = 0
10.     j = 0
11.     C = []
12.     while (i < len(A)) and (j < len(B)):
13.         if (A[i] <= B[j]):
14.             C.append(A[i]) # Add A[i] to the end of C
15.             i += 1
16.         else:
17.             C.append(B[j])
18.             j += 1
19.     return C + A[i..len(A)-1] + B[j..len(B)-1] # List concatenation
```

Example – Running-Time Complexity of Merge Sort

```
""" Given a non-empty list A, sort the list in non-decreasing order. """
    def MergeSort(A):
2.      if len(A) == 1:
3.          return A
4.      else:
5.          m = len(A) // 2  # integer division
6.          L1 = MergeSort(A[0..m-1])
7.          L2 = MergeSort(A[m..len(A)-1])
8.          return merge(L1, L2)
    def merge(A, B):
9.      i = 0
10.     j = 0
11.     C = []
12.     while (i < len(A)) and (j < len(B)):
13.         if (A[i] <= B[j]):
14.             C.append(A[i]) # Add A[i] to the end of C
15.             i += 1
16.         else:
17.             C.append(B[j])
18.             j += 1
19.     return C + A[i..len(A)-1] + B[j..len(B)-1] # List concatenation
```

Example – Running-Time Complexity of Merge Sort

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

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

Review – Logarithm and Exponent Rules

- $x^{a \times b} = (x^a)^b$.
- If $z = \log_x y$, then $x^z = y$.
- $x^{\log_x y} = y$.

A General Recurrence Form

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

The Master Theorem

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 – Applying the Master Theorem

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:** Recursive Binary Search:

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

Example – Applying the Master Theorem

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:** Merge Sort:

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

Example – Applying the Master Theorem

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:** $T(n) = 9T(\frac{n}{3}) + 100n^{\sqrt{2}}$.

Example – Applying the Master Theorem

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:** $T(n) = 10T(\frac{n}{4}) + 10n^\pi$.