COMP20007 Design of Algorithms

Week 3 Workshop

Tutorial

1. Sums

Give closed form expressions for the following sums.

a. $\sum_{i=1}^{n} 1$ $\sum_{i=1}^{n} 1 = n$

b. $\sum_{i=0}^{n-1} \sum_{j=0}^{i} 1 = \sum_{i=0}^{n-1} (i+1) = \sum_{k=1}^{n} k = \frac{(1+n)\times n}{2} = \frac{n(n+1)}{2}$

c. $\sum_{i=1}^{n} i = \frac{(1+n)n}{2}$

d. $\sum_{i=1}^{n} \sum_{j=1}^{m} ij = \sum_{i=1}^{n} i\left(\sum_{j=1}^{m} j\right) = \sum_{i=1}^{n} i\frac{(1+m)m}{2} = \frac{m(m+1)n(n+1)}{4}$

e. $\sum_{i=1}^{n}(2i+3) = 2\sum_{i=1}^{n} i + 3\sum_{i=1}^{n} 1 = (n+1)n + 3n = n^2+4n$

f. $\sum_{k=0}^{n} x^k = \frac{1(1-x^{n+1})}{1-x} = \frac{x^{n+1}-1}{x-1}$

2. Complexity classes

For each of the following pairs of functions, $f(n)$ and $g(n)$ determine whether $f(n) \in O(g(n))$, $f(n) \in \Omega(g(n))$ or both (*i.e.*, $f(n) \in \Theta(g(n))$).

a. $f(n) = \frac{1}{2}n^2$ and $g(n) = 3n$ $f(n) \in \Omega(g(n))$

b. $f(n) = n^2 + n$ and $g(n) = 3n^2 + \log n$ $f(n) \in \Theta(g(n))$

c. $f(n) = n\log n$ and $g(n) = \frac{n}{4}\sqrt{n}$ $f(n) \in O(g(n))$

d. $f(n) = \log(10n)$ and $g(n) = \log(n^2)$ $f(n) \in \Theta(g(n))$

e. $f(n) = (\log n)^2$ and $g(n) = \log(n^2)$ $f(n) \in O(g(n))$ $\times$ $(\log n)^2 \in \Omega(\log(n^2))$ $\lim_{n\to\infty} \frac{(\log n)^2}{\log n^2} = \lim_{n\to\infty} \frac{(\log n)^2}{2\log n} = \lim_{n\to\infty} \frac{1}{2}\log n \to \infty$ $\Rightarrow (\log n)^2$ faster than $\log n^2$

f. $f(n) = \log_{10} n$ and $g(n) = \ln n$ $f(n) \in \Theta(g(n))$

g. $f(n) = 2^n$ and $g(n) = 3^n$ $f(n) \in \Theta(g(n))$ $\lim_{n\to\infty} \frac{2^n}{3^n} = \lim_{n\to\infty} (\frac{2}{3})^n = 0 \Rightarrow 2^n \in O(3^n)$

h. $f(n) = n!$ and $g(n) = n^n$ $f(n) \in O(g(n))$ $n! = n^{n+\frac{1}{2}} = \exp(n+\frac{1}{2})\ln n$

3. Sequential search

Adapted from *Levitin* [2nd Ed.] 2.2.1. Use O, Ω and/or Θ to make the strongest possible claim about the runtime complexity of sequential search in,

a. general (*i.e.*, all possible inputs) $C(n) \in \Omega(1)$ and $C(n) \in \Theta(n)$

b. the best case $C_{best}(n) = 1 \in \Theta(1)$

c. the worst case $C_{worst}(n) = n \in \Theta(n)$

d. the average case $C_{aver}(n) = \frac{n+1}{2} \cdot p + n(1-p) \Rightarrow C_{aver} \in \Theta(n)$

4. *k*-Merge

Adapted from *DPV* 2.19. Consider a modified sorting problem where the goal is to sort k lists of n sorted elements into one list of kn sorted elements.

$\sum_{m=2}^{k} mn = n \cdot \sum_{m=2}^{k} m = \frac{(k+2)(k-1)}{2} n \Rightarrow$ time complexity is $\Theta(k^2 n)$

One approach is to merge the first two lists, then merge the third with those, and so on until all k lists have been combined. What is the time complexity of this algorithm? Can you design a faster algorithm using a divide-and-conquer approach?

merge all $2k$ lists to $4k$ over and over again until $\Theta(kn \log k)$

Considering the devide and conquer. merge all 2-pair lists $\Rightarrow 2k$ and all have been merged

5. Mergesort complexity (Homework)

Mergesort is a divide-and-conquer sorting algorithm made up of three steps (in the recursive case):

1. Sort the left half of the input (using mergesort)
2. Sort the right half of the input (using mergesort) $\Theta(n \log n)$
3. Merge the two halves together (using a merge operation)

Using your intuition, see what you might expect the complexity of the algorithm to be using the faster merge operation derived in question 4.

Next week we will revisit this question and formally verify what you find using recurrence relations.