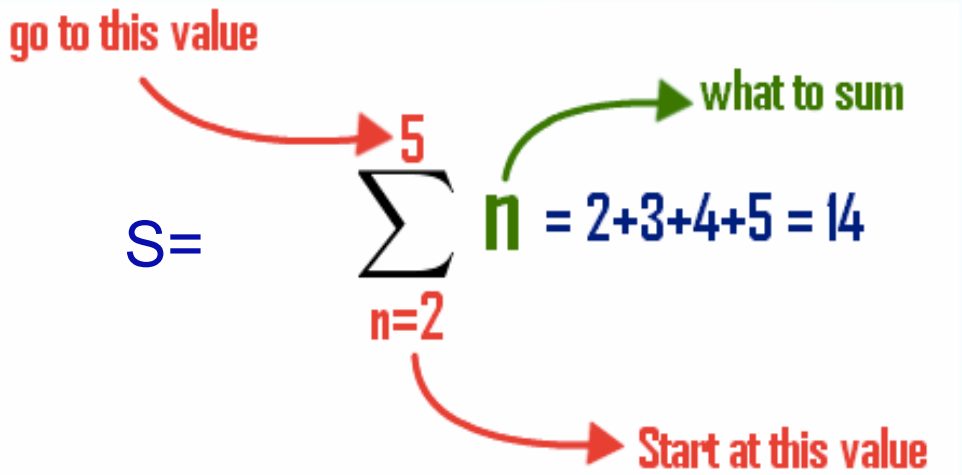
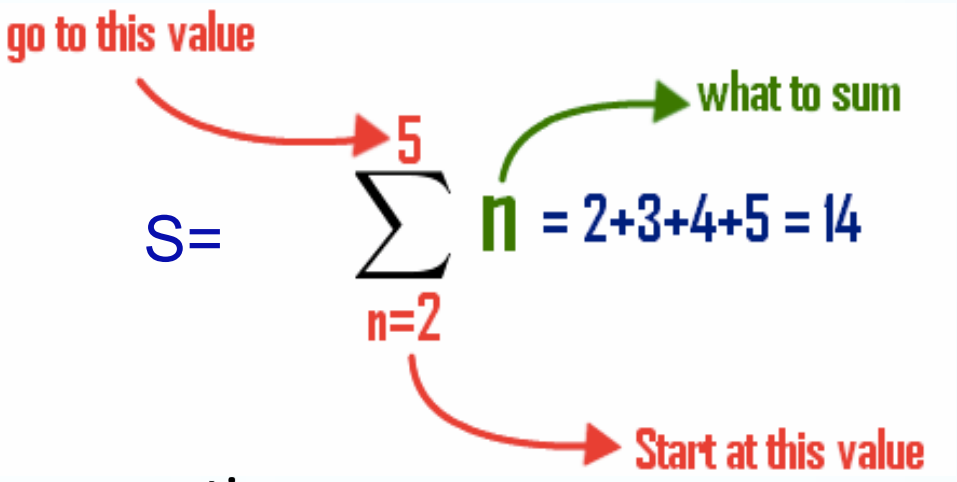
COMP20007 Workshop Week 3

0	Sigma notation & some important formulas
1	(Asymptotic) Complexity
2	mergesort, k-way merge
3	Lab: multi-file program, `make` & `Makefile`

Sigma Notation



Sigma Notation



- So, the above sigma is just for computing:

```
s ← 0

for n ← 2 to 5

   s ← s + n
```

- Q: Write the sigma notation for the sum

$x_1 + x_2 + \ldots + x_n$

W0 (Workshop Problem 0)

Give closed form expressions for the following sums.

(a) $\sum_{i=1}^{n} 1$

(b) $\sum_{i=1}^{n} i$

(c) $\sum_{i=1}^{n} (2i + 3)$

(d) $\sum_{i=0}^{n-1} \sum_{j=0}^{i} 1$

(e) $\sum_{i=1}^{n} \sum_{j=1}^{m} ij$

(f) $\sum_{k=0}^{n} x^k$

Sum Manipulation Rules

1. $$\sum_{i=l}^{u} ca_i = c \sum_{i=l}^{u} a_i$$

2. $$\sum_{i=l}^{u} (a_i \pm b_i) = \sum_{i=l}^{u} a_i \pm \sum_{i=l}^{u} b_i$$

Important Sums

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

$\Theta(n)$

$$\sum_{i=1}^{n} i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \approx \frac{1}{2}n^2$$

$\Theta(n^2)$

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{1}{3}n^3$$

$\Theta(n^3)$

$$\sum_{i=0}^{n} a^i = 1 + a + \cdots + a^n = \frac{a^{n+1} - 1}{a - 1} \ (a \neq 1);$$

$$\begin{cases} \Theta(a^n) \text{ if } a>1 \\ \Theta(n) \text{ if } a=1 \\ \Theta(1) \text{ if } a<1 \end{cases}$$

$$\sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \cdots + \frac{1}{n} \approx \ln n + \gamma, \text{ where } \gamma \approx 0.5772\ldots$$

$\Theta(\log n)$

Method 1: Reduce a function to its respective class

Basic Classes: $1 \prec \log n \prec n^{\epsilon} \prec n^{c} \prec n^{\log n} \prec c^{n} \prec n^{n}$

Rule 1: In a sum, keep only the highest order element

Rule 2: Replace any free (ie. not inside any function) constant with 1

Method 2 (Powerful): Use lim

$$\lim_{n\to\infty}\frac{t(n)}{g(n)}=\begin{cases}0 & \text{implies } t \text{ grows asymptotically slower than } g\\ c & \text{implies } t \text{ and } g \text{ have same order of growth}\\ \infty & \text{implies } t \text{ grows asymptotically faster than } g\end{cases}$$

$$t(n)=O(g(n))$$
$$t(n)=\theta(g(n))$$
$$t(n)=\Omega(g(n))$$

Important

L'Hôpital Rule

$$\lim_{n\to\infty}\frac{t(n)}{g(n)}=\lim_{n\to\infty}\frac{t'(n)}{g'(n)}$$

W1

(Workshop Problem 1)

For each of the pairs of functions *f(n)* and *g(n)*, determine if $f \in O(g)$, $f \in \Omega(g)$, or both (ie., $f \in \Theta(g)$). Show your workout.

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

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

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

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

(e) $f(n) = (\log n)^2$ and $g(n) = \log(n^2)$

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

(g) $f(n) = 2^n$ and $g(n) = 3^n$

(h) $f(n) = n!$ and $g(n) = n^n$

W2: Sequential Search & Complexity

Use O, Ω and/or Θ to make strongest possible claims about the runtime complexity of sequential search in:

(a) general (*i.e.*, all possible inputs)

(b) the best case

(c) the worst case

(d) the average case

Note: also see the lecture recording, where Lars did a detailed analysis for the average case

W5 (optional)

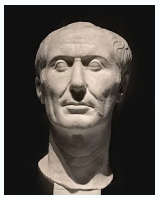
Mergesort is a divide-and-conquer sorting algorithm made up of three steps (in the recursive case):

1. Sort the left half of the input (using mergesort)
2. Sort the right half of the input (using mergesort)
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.

Philip II of Macedon

Divide & Conquer



To solve a size-*n* problem:

- Break the problem into a set of similar sub-problems, each of a *smaller-than-n* size,
- Solve each sub-problem in the same way (if simple enough, solve it directly), and
- Combine the solutions of sub-problems into the total solution.

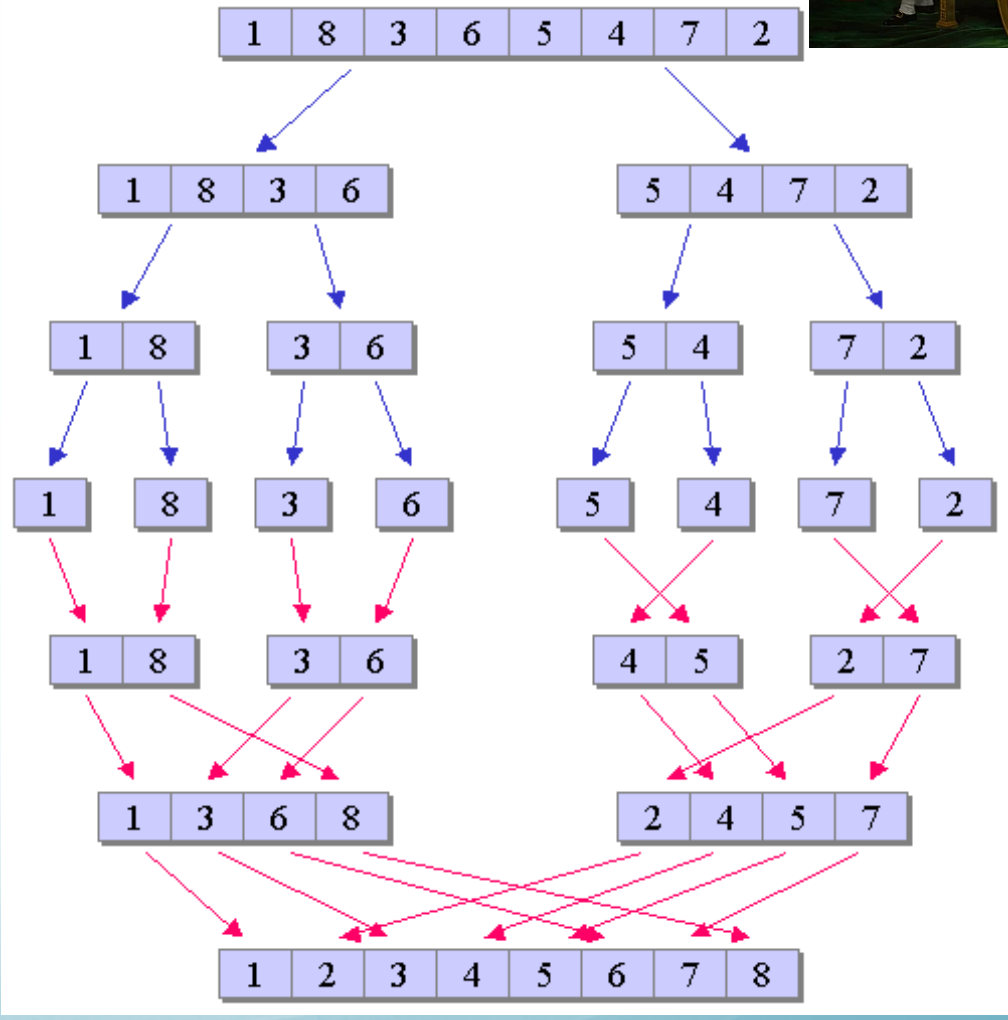


Image Sources: Wikipedia, cs.amherst.edu

W4: k-way mergesort

Consider a modified sorting problem where the goal is to merge k lists of n sorted elements into one list of kn sorted elements.

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?

Lab Week 3: Modular Programming

Modular programming: separating the functionality of a program into independent, interchangeable **modules**, such that each contains everything necessary to execute only one aspect of the desired functionality.

- Follow steps 1—2 of the lab for building and testing module `racecar`
- Follow step 3 for using `make` and `Makefile` to auto-compile a multi-file program
- See `github.com/anhvir/c207` for a few different (and equivalent) versions of `Makefile` for `racecar` and `racecar_test`. First, just copy and paste `Makefile1` from `github` to your `Makefile` and try to "`make`".
- Follow step 4 to learn more about, and to add a function into, module `racecar`
- Do Step 5: build your own `list.h, list.c, list_test.c`. As a reference, you can use Alistair's `listops.c` (just google "listops.c" to get the file).
- Use the list module to build a stack module in a least effort way. Test your stack by writing `stack_test.c` that input a series of integers, and then print them in reverse order. Build a single `Makefile` for testing both list and stack