COMP20007 Design of Algorithms

Week 3 Workshop Solutions

Tutorial

0. Sums

(a)

$$\sum_{i=1}^{n} 1 = \underbrace{1 + 1 + \ldots + 1}_{n \text{ times}} = n$$

(b)

$$\sum_{i=1}^{n} i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \quad \text{(Triangle numbers formula)}$$

(c)

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

(d)

$$\sum_{i=0}^{n-1}\sum_{j=0}^{i} 1 = \sum_{i=0}^{n-1}(i+1) = 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

(e)

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

(f)

$$\sum_{k=0}^{n} x^k = \frac{1 - x^{n+1}}{1 - x} \quad \text{(Geometric series)}$$

1. Complexity classes

(a) $\frac{1}{2}n^2 \in \Omega(3n)$ – ignore constants, n^2 grows faster than n

(b) $n^2 + n \in \Theta(3n^2 + \log n)$ – ignore constants, the fastest growing terms are both n^2

(c) $n \log n \in O\left(\frac{n}{4}\sqrt{n}\right)$ – ignoring constants the difference here is $\log n$ vs. $\sqrt{n}$

(d) $\log(10n) \in \Theta(\log(n^2))$ – using log laws $f(n) = \log 10 + \log n$ and $g(n) = 2 \log n$

(e) $(\log n)^2 \in \Omega(\log(n^2))$ – we can see that $(\log n)^2/(2 \log n) = \frac{1}{2} \log n$ so $f(n)$ grows faster

(f) $\log_{10} n \in \Theta(\ln n)$ – using change of base formula $\log_{10} n = \frac{1}{\ln 10} \ln n$

(g) $2^n \in O(3^n)$ – unlike logarithms, the base in the exponential changes the growth rate

(h) $n! \in O(n^n)$ – $1 \times 2 \times \cdots \times n \in O(n \times n \times \cdots \times n)$ but not vice versa

2. Sequential search

(a) general (*i.e.*, all possible inputs): the best case for sequential search is when the element we're searching for is at index 0. So $C_{\text{best}}(n) = 1$. Also $C_{\text{worst}}(n) = n$ occurs when the element is not in the array. So $C_{\text{best}}(n) \leq C(n) \leq C_{\text{worst}} \implies C(n) \in \Omega(1)$ and $C(n) \in O(n)$.

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

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

(d) the average case: let the probability of the element being in the array be p. If the element is not in the array the cost is n. If the element is in the array we assume it's equally likely to be in any of the n indices. Taking the average of the number of comparisons when the element is in each index:

$$\frac{1}{n}(1 + 2 + \cdots + n) = \frac{1}{n} \cdot \frac{n(n+1)}{2} = \frac{n+1}{2}.$$

So the cost is $\frac{n+1}{2}$ with probability p, or n with probability $(1-p)$, *i.e.*,

$$C_{\text{average}}(n) = p \times \frac{n+1}{2} + (1-p) \times n.$$

Since p is a constant here, $C_{\text{average}} \in \Theta(n)$.

3. Solving recurrence relations Solve the following recurrence relations, assuming $T(1) = 1$.

(a) $T(n) = 4n - 3$
$T(n) = T(n-1)+4$ means $T(n-1) = T((n-1)-1)+4 = T(n-2)+4$, and $T(n-2) = T(n-3)+4$, and so on. Repeatedly substitute into the first equation, until the pattern becomes clear. Then, produce the base case to eliminate T.

$$\begin{aligned}
T(n) &= T(n-1) + 4 \\
&= T(n-2) + 4 + 4 \\
&= T(n-3) + 4 + 4 + 4 \\
&\vdots && \text{(starting to see the pattern?)} \\
&= T(n-k) + k \times 4 \\
&\vdots && \text{(bring in the base case)} \\
&= T(n-(n-1)) + (n-1) \times 4 \\
&= T(1) + (n-1) \times 4 \\
&= 1 + 4n - 4 \\
&= 4n - 3
\end{aligned}$$

(b) $T(n) = \dfrac{n(n+1)}{2}$

$$\begin{aligned}
T(n) &= T(n-1) + n \\
&= T(n-2) + (n-1) + n \\
&= T(n-3) + (n-2) + (n-1) + n \\
&\vdots \\
&= T(n-k) + (n-(k-1)) + \cdots + (n-2) + (n-1) + n \\
&\vdots \\
&= T(n-(n-1)) + (n-(n-1-1)) + \cdots + (n-2) + (n-1) + n \\
&= T(1) + (n-n+1+1) + \cdots + (n-2) + (n-1) + n \\
&= 1 + 2 + \cdots + (n-2) + (n-1) + n \\
&= \frac{n(n+1)}{2} && \text{(triangle numbers)}
\end{aligned}$$

(c) $T(n) = 2^n - 1$

$$\begin{aligned}
T(n) &= 2T(n-1) + 1 \\
&= 2(2T(n-2)+1) + 1 = 2^2 T(n-2) + 2 + 1 \\
&= 2^2(2T(n-3)+1) + 2 + 1 = 2^3 T(n-3) + 2^2 + 2 + 1 \\
&\vdots \\
&= 2^k T(n-k) + 2^{k-1} + \cdots + 2^2 + 2 + 1 \\
&\vdots \\
&= 2^{(n-1)} T(n-(n-1)) + 2^{(n-1)-1} + \cdots + 2^2 + 2 + 1 \\
&= 2^{n-1} T(1) + 2^{n-2} + \cdots + 2^2 + 2 + 1 \\
&= 2^{n-1} + 2^{n-2} + \cdots + 2^2 + 2 + 1 \\
&= 2^n - 1 && \text{(sum of powers of 2 is the next power of 2, minus 1)}
\end{aligned}$$

4. k-Merge Merging two lists of sizes a and b takes about $a + b$ steps. Merging the first two lists (sizes n and n) will take $2n$ steps. Merging this list with the next list (sizes $2n$ and n) will take $3n$ steps. The next will be $4n$ steps, and so on, until the last list of n is merged with the rest of the $(k-1)n$ items, taking kn steps. In total,

$$2n + 3n + 4n + \cdots + kn = n(2 + 3 + 4 + \cdots + k)$$

Simplifying by recognising the triangle numbers, we're looking at $\Theta(k^2 n)$:

$$n(2 + 3 + 4 + \cdots + k) = n(1 + 2 + 3 + 4 + \cdots + k) - n = n\frac{k(k+1)}{2} - n \in \Theta(k^2 n)$$

A faster algorithm would use the merging strategy from mergesort, merging the lists in pairs. First, merge $k/2$ pairs of length n lists, resulting in $k/2$ lists of length $2n$ (and taking $k/2 \times 2n = kn$ steps). Next, merge $k/4$ pairs of length $2n$ lists, resulting in $k/4$ lists of length $4n$ (and taking $k/4 \times 4n = kn$ steps). Continue, a total of $\log k$ times, and you will have one list of length kn in $\Theta(kn \log k)$ time.

5. Mergesort complexity (optional) Recurrence relation:

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + \Theta(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

where $T(n)$ is the runtime of mergesort sorting n elements. The first $T(\frac{n}{2})$ is the time it takes to sort the left half of the input using mergesort. The other $T(\frac{n}{2})$ is the time it takes to sort the right half. $\Theta(n)$ is a bound on the time it takes to merge the two halves together.

We haven't seen the master theorem in class yet, and for this question we aren't required to solve the recurrence relation. However if we did want to solve this we can recognise that this recurrence relation fits the master theorem, with $a = 2$, $b = 2$, and $d = 1$.

$$\log_b(a) = \log_2(2) = 1 = d$$

so, by the master theorem, $T(n) \in \Theta(n \log n)$.