School of Computing and Information Systems
COMP90038 Algorithms and Complexity Tutorial Week 4

Sample Answers

The exercises

15. For each of the following pairs f, g, determine whether $f(n) \in O(g(n))$, or $g(n) \in O(f(n))$, or both:

    (a) $f(n) = (n^2 + 1 - n^2)/2$ and $g(n) = 2n$
    (b) $f(n) = n^2 + n\sqrt{n}$ and $g(n) = n^2 + n$
    (c) $f(n) = n \log n$ and $g(n) = \frac{n}{4}\sqrt{n}$
    (d) $f(n) = n + \log n$ and $g(n) = \sqrt{n}$
    (e) $f(n) = 4n \log n + n$ and $g(n) = (n^2 - n)/2$
    (f) $f(n) = (\log n)^2$ and $g(n) = 2 + \log n$

    Answer:

    (a) $f(n) \in O(g(n))$ (b) $f(n) \in O(g(n))$ and $g(n) \in O(f(n))$
    (c) $f(n) \in O(g(n))$ (d) $g(n) \in O(f(n))$
    (e) $f(n) \in O(g(n))$ (f) $g(n) \in O(f(n))$

16. Show the steps of selection sort, when given the keys S, O, R, T, X, A, M, P, L, E.

    Answer: Have fun :)

17. One possible way of representing a polynomial

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

    is as an array A of length $n + 1$, with $A[i]$ holding the coefficient a_i.

    (a) Design a brute-force algorithm for computing the value of $p(x)$ at a given point x. Express this as a function PEVAL(A, n, x) where A is the array of coefficients, n is the degree of the polynomial, and x is the point for which we want the value of p.
    (b) If your algorithm is $\Theta(n^2)$, try to find a linear algorithm.
    (c) Is it possible to find an algorithm that solves the problem in sub-linear time?

    Answer:

    (a) The following algorithm is perhaps the most natural formulation:

```
function PEVAL(A, n, x)
    result ← 0.0
    for i ← n downto 0 do
        summand ← 1.0
        for j ← 1 to i do
            summand ← x × summand
```

result ← *result* + $A[i] \times$ *summand*

return *result*

The complexity is $\Theta(n^2)$.

(b) The previous algorithm contains many redundant computations of powers of x. Here is an algorithm that is both simpler and more efficient:

function PEVAL(A, n, x)
result ← $A[n]$
for $i \leftarrow n-1$ **downto** 0 **do**
result ← *result* $\times x + A[i]$
return *result*

This algorithm effectively uses *factoring* of the polynomial. For example, consider the polynomial $7x^4 + 3x^3 + 4x^2 - x - 5$ (which is represented by the array $A[0]$–$A[4]$ with content -5, -1, 4, 3, 7). We can rewrite the polynomial as $(((7x + 3)x + 4)x - 1)x - 5$ (this is what we mean by factoring it). The second algorithm evaluates the polynomial this way. Note that, for the example, this requires only four multiplications—far fewer than the first algorithm used.

(c) We cannot solve the problem in less than linear time, because we clearly need to access each of the $n + 1$ coefficients.

18. Trace the brute-force string search algorithm on the following input: The path p is '`needle`', and the text t is '`there_need_not_be_any`'. How many comparisons (successful and unsuccessful) are made?

Answer: 21 character comparisons are made.

19. Assume we have a text consisting of one million zeros. For each of these patterns, determine how many character comparisons the brute-force string matching algorithm will make:

(a) `010001` (b) `000101` (c) `011101`

Answer: (a) 2×999995 comparisons (b) 4×999995 comparisons (c) 2×999995 comparisons

20. Give an example of a text of length n and a pattern of length m, which together constitute a worst-case scenario for the brute-force string matching algorithm. How many character comparisons, as a function of n and m, will be made for the worst-case example. What is the value of m (the length of the pattern) that maximises this function? i.e. What is the worst case pattern length?

Answer: The worst case happens when we have a text of length n consisting of the same character c repeated n times, together with a pattern of length m, consisting of $m - 1$ occurrences of c, followed by a single character different from c. In this case, the outer loop is traversed $n - m + 1$ times, and each time, m character comparisons are made before failure is detected. Altogether we have $(n - m + 1)m = (n + 1)m - m^2$ comparisons. As a function of m, this has its maximal value where $n + 1 - 2m = 0$, that is, when the length of the pattern is about half that of the text.

21. The *assignment problem* asks how to best assign n jobs to n contractors who have put in bids for each job. An instance of this problem is an $n \times n$ *cost matrix* C, with $C[i, j]$ specifying what it will cost to have contractor i do job j. The aim is to minimise the total cost. More formally, we want to find a permutation $\langle j_1, j_2, \ldots j_n \rangle$ of $\langle 1, 2, \ldots, n \rangle$ such that $\sum_{i=1}^{n} C[i, j_i]$ is minimized. Use brute force to solve the following instance:

	Job 1	**Job 2**	**Job 3**	**Job 4**
Contractor 1	9	2	7	8
Contractor 2	6	4	3	7
Contractor 3	5	8	1	8
Contractor 4	7	6	9	4

Answer:

Permutation	**Cost**
1,2,3,4	9+4+1+4 = 18
1,2,4,3	9+4+8+9 = 30
1,3,2,4	9+3+8+4 = 24
1,3,4,2	9+3+8+6 = 26
1,4,2,3	9+7+8+9 = 33
1,4,3,2	9+7+1+6 = 23
2,1,3,4	2+6+1+4 = 13
2,1,4,3	2+6+8+9 = 25
⋮	

and so on. The minimal cost is 13, for permutation $\langle 2, 1, 3, 4 \rangle$.

22. Give an instance of the assignment problem which does not include the smallest item $C[i, j]$ of its cost matrix.

Answer:

	Job 1	**Job 2**
Contractor 1	1	2
Contractor 2	2	4