1.A Handout 5

A. Section 1.7

2(a) $a_8 = 2^{8-1} = 2^7 = 128$.
(b) $a_8 = 7$.
(c) $a_8 = 1 + (-1)^8 = 1 + 1 = 2$.
(d) $a_8 = -(-2)^8 = -256$.

10 (a) $a_0 = 3$, $a_n = (2n+1) + a_{n-1}$ will produce the sequence. Also, $a_n = n^2 + 2$ will work, if we begin the sequence with a_1.
(c) $a_n = (n+1)$ written in binary, with no leading zeroes. (Note you are only required to give a rule, and not necessarily a formula.) If we start with a_1, then clearly $a_n = n$.
(f) $a_0 = 1$, $a_n = (2n+1) \cdot a_{n-1}$ works, or $a_n = (2n+1)!/(2^n n!)$, if we begin the sequence with a_1.

16 (b) $\sum_{j=0}^{8}(3^j - 2^j) = \sum_{j=0}^{8} 3^j - \sum_{j=0}^{8} 2^j$
$= (3^{8+1} - 1)/2 - (2^{8+1} - 1)/1$ (rule for geometric series)
$= 9841 - 511 = 9330$.

18 (c) $\sum_{i=1}^{3}\sum_{j=0}^{2} j = \sum_{i=1}^{3}(0 + 1 + 2) = 3 \cdot 3 = 9$.
(d) $\sum_{i=0}^{2}\sum_{j=0}^{3} i^2 \cdot j^3 = \sum_{i=0}^{2} i^2(\sum_{j=0}^{3} j^3)$
$= \sum_{i=0}^{2} i^2(\frac{3^2(3+1)^2}{4})$ (Table 2, p76)
$= \sum_{i=0}^{2} i^2 \cdot 36 = 36\sum_{i=0}^{2} i^2$
$= 36 \cdot \frac{2(2+1)(2\cdot 2+1)}{6} = 36 \cdot 5 = 180$.

B.
The sum of all odd numbers from 1 to 99 is: the sum of 1 to 99, subtracting the sum of even numbers from 2 to 98.
$\sum_{k=1}^{99} k - \sum_{k=1}^{49} 2k = \sum_{k=1}^{99} k - 2\sum_{k=1}^{49} k = \frac{99\cdot 100}{2} - (49\cdot 50) = 99\cdot 50 - 49\cdot 25 = 4950 - 2450 = 2500 = 50^2$.
So the sum is a perfect square.
(Of course, there are other ways to show this as well...)

1.B Handout 6

A. Section 1.8

2. (a) Let $C = 18, k = 11$. Then for $x > k$, $f(x) = 17x + 11 = 17x + k < 17x + x = (17+1)x = Cx \leq Cx^2$.
(b) Let $C = 2, k = 1000$. Then for $x > k$, $f(x) = x^2 + 1000 = x^2 + k < x^2 + x < x^2 + x^2 = C \cdot x^2$.
(c) We assume the log is of base 2. Let $C = 1, k = 1$. Then for any $x > k$, $\log_2 x < x$. (We won't prove this here.) Therefore $x \cdot \log_2 x < C \cdot x^2$.
(d) $f(x) = x^4/2$ is not $O(x^2)$. For any $C, k > 0$ (integers), let $x = k \cdot C \cdot 2$. Then $x > k$, and $x^4/2 = x^2 \cdot (x^2/2) = x^2 \cdot (k \cdot C \cdot 2)^2/2 > 2C^2/2 \cdot x^2 \geq C \cdot x^2$.
(e) $f(x) = 2^x$ is not $O(x^2)$. Suppose it is. Then 2^x is $O(2^{2\log x})$. Since $\log x$ is not $O(x)$, as is referenced in (c), we have that $2\log x$ is not $O(x)$, which gives us the result.

(f) $f(x) = \lceil x \rceil \cdot \lfloor x \rfloor$ is $O(x^2)$. Let $C = 2, k = 1$. Then for any $x > k$, $\lceil x \rceil \cdot \lfloor x \rfloor \leq x \cdot \lceil x \rceil < x \cdot 2x = Cx^2$.
8a.
$f(x)$ is $O(x^4)$. It is not $O(x^3)$ since one of its terms, $x^3 \log x$, is asymptotically larger than x^3.

8b.
$f(x)$ is $O(x^5)$. The polylogarithmic factor (i.e. $(\log x)^4$) is $O(x)$, so it can be ignored.

20a.
$f(x)$ is $O(x^3 \log x)$. This term is the dominant one when the terms of the function are expanded.

20b.
$f(x)$ is $O(2^n \cdot 3^n) = O(6^n)$. This is clearly the largest term when the terms of function are expanded.

28a.
Let $C_1 = 1, C_2 = 2, k = 2$. Then for any $x > k > 0$,
$C_1 \cdot 3x^2 \leq 3x^2 + x + 1 \leq 3x^2 + x^2 = 4x^2 \leq 6x^2 = C_2 \cdot 3x^2$.

In the preceding exercises, if you have a &1 marked on your sheet, it means you did not prove some result you were supposed to prove.

B.

Claim: $2^n = O(n!)$.
Proof. Let $f(x) = 2^x$, $g(x) = x!$, $C = 1$, and $k = 3$. Given x, if we assume $x > 3$, then
$|f(x)| = |2^x| = 2^x = 2 \cdot 2 \cdot \cdots^x \cdots \cdot 2 \leq x \cdot (x-1) \cdot \quad \cdots \quad \cdot 2 = x! = C \cdot x! = C \cdot g(x)$
where $\cdots^x \cdots$ means that the total quantity is iterated x times.
On the left hand side of the $\leq$, 2 is multiplied x times. On the right hand side, $x - 1$ numbers *greater than 2* are multiplied, and 2 is multiplied with that. So for each 2 in the product on the left hand side, there is a corresponding number in the right hand side product which is ≥ 2.

$\square$

Claim: $n!$ is not $O(2^n)$.
Proof. Here is one short proof; there are of course many others. Given $C, k > 0$, (C and k integers), let $x = k \cdot 2C + 3$. Clearly, $x > k$. We must show that $|x!| = x! > C \cdot 2^x$.
$x! = (k \cdot 2C + 3)! = (k \cdot 2C + 3)(k \cdot 2C + 2)!$
$\geq (k \cdot 2C + 3) \cdot 2^{k \cdot 2C + 2}$ (by the previous problem, and $k \cdot 2C + 2 > 3$)
$> C \cdot 2^1 \cdot 2^{k \cdot 2C + 2}$
$= C \cdot 2^x$.

$\square$

1.C Handout 7

A. Section 2.1
24
We allow an "output" command in which the algorithm can output part of the final answer at that point in the program.
procedure modefind($a_1, \ldots, a_n$: nondecreasing integers)

ints := 0
{ compute matrices containing each integer in the sequence and a corresponding count of how many times the integer occurs }
for $i := 1$ to n
  int$[i] := \infty$ (∞ = some unique value different from all integers)
  count$[i] := 0$
for $i := 1$ to n
  flag := $\top$
  highcount := 0;
  for $j := 1$ to ints
    if $a_i = \text{int}[j]$ **then** count$[i]$:= count$[i] + 1$; flag := $\perp$
    if count$[j]$ > highcount **then** highcount := count$[j]$
  if flag **then** ints := ints + 1; int$[ints] := a_i$; count$[ints] := 1$
{ output any integer with a count equal to the last computed highcount; these are the modes }
for $j := 1$ to ints
  if count$[j]$ = highcount **then output** int$[j]$

B. Section 2.2
2
procedure sortfour($a_1, \ldots, a_n$: elements of a list with a linear order)
$S := \{a_1, a_2, a_3, a_4\}$
for $i = 1$ to 4
  { recall that $\min(S)$ = the minimum element in S }
  $A_i := \min(S)$
  $S := S - \{A_i\}$
{ the sequence A_1, A_2, A_3, A_4 is sorted in increasing order }

The key observation is that the for loop requires four iterations no matter what n is. Every step of this algorithm is wholly independent of the input size. Counting each non-commented line as a step, 13 steps are always taken. Thus we have $O(1)$ time complexity.

6b
Each bitwise AND removes exactly one 1 from S, since $S := S \wedge (S-1)$ simply amounts to changing the rightmost 1 to a 0 ...

$$
\begin{array}{cccccccccccl}
 & & & & & \downarrow & & & & & & \\
 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & S \\
\wedge & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & S-1 \\
\hline
 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \text{new } S \\
 & & & & & \uparrow & & & & & &
\end{array}
$$

Pairing each bitwise AND with the 1 that it eliminates, we immediately see that the number of bitwise ANDs is equal to the number of 1's in the input string.
[To get a coarser estimate, one that refers only to the size of the input, we can note that a bit string of length n can have at most n 1's, so the worst case number of bitwise ANDs is clearly $O(n)$.]

18

This will vary depending on what your algorithm was, obviously. The largest bottleneck in our algorithm are the nested for loops, the outer loop ranging from 1..n, the inner loop ranging from 1..int (a quantity which can be as large as n). Thus our running time is $O(n^2)$.