Exercises

19. Integer round-off notation occurs often in algorithms and algorithm analysis:

$\lceil i \rceil$ = The least integer $\geq i$.

$\lfloor i \rfloor$ = The greatest integer $\leq i$.

Prove that:

A. For any real number x:

$$x-1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x+1$$

B. For any two integers i and j such that $j \neq 0$:

$$\left\lfloor \frac{i}{j} \right\rfloor + \left\lceil i - \frac{i}{j} \right\rceil = i$$

(for example, $\lceil i/2 \rceil + \lfloor i/2 \rfloor = i$)

C. For any integers i,j,k such that $j \neq 0$ and $k \neq 0$:

$$\left\lceil \frac{\lceil i/j \rceil}{k} \right\rceil = \left\lceil \frac{i}{jk} \right\rceil$$

$$\left\lfloor \frac{\lfloor i/j \rfloor}{k} \right\rfloor = \left\lfloor \frac{i}{jk} \right\rfloor$$

D. For any positive integers i,j,k such that $j \geq k$:

$$i \left\lceil \frac{j}{k} \right\rceil \geq \left\lceil \frac{ij}{k} \right\rceil$$

$$i \left\lfloor \frac{j}{k} \right\rfloor \leq \left\lfloor \frac{ij}{k} \right\rfloor$$

In addition, for both inequalities, give a class of examples where the difference can be arbitrarily close to a factor of 2.

20. It was pointed out in the presentation of O and Ω notation that $O(1)$ is a useful concept but $\Omega(1)$ is not. Suppose that we generalized our definitions of O and Ω to functions from the non-negative integers to the non-negative real numbers (e.g., $f(n) = 1/n$). How does this change the status of $\Omega(1)$? What if we allow functions to be from the real numbers to the real numbers (possibly negative)?

21. We have limited ourselves mainly to the O and Ω notation, with occasional use of Θ. Some authors also use the "little o" notation when they wish to emphasize that f is not just bounded by a constant times g, but truly grows more slowly than g. Formally:

little o: A function $f(n)$ is $o(g(n))$ if for every real number $b>0$ there exists an integer $a>0$ such that for all integers $n \geq a$, $f(n) < b*g(n)$.

Informally, if one thinks of O as $\leq$ between functions, little o can be thought of as $<$ between functions. That is, no matter how small you make b, $f(n)$ is still smaller than $bg(n)$ for all but a finite number of values.

A. Prove that:

n is not $o(2n)$ but n is $o(n^2)$

$\log_{10}(n)$ is $O(\log_2(n))$ but not $o(\log_2(n))$

for real numbers $x,y>1$, $\log_x(n)$ is $o(n^y)$

B. Prove that if the condition $f(n) < b*g(n))$ is replaced by the condition $f(n) \leq b*g(n)$, the meaning of the little o definition is unchanged.

C. Prove that $f(n)$ is $o(g(n))$ if and only if $f(n)$ is not $\Omega(g(n))$.

D. Recall from the sample exercises the stronger definition of Ω:

Stronger definition of Ω: A function $f(n)$ is $\Omega(g(n))$ if there exists real numbers $a,b>0$ such that for all integers $n \geq a$, $f(n) \geq b*g(n)$.

Prove that if $f(n)$ is $o(g(n))$ then it is not $\Omega(g(n))$ for either the standard or stronger definition of Ω. However, also prove that if $f(n)$ is not $\Omega(g(n))$ under the stronger definition of Ω, it is not necessarily true that $f(n)$ is $o(g(n))$, and hence Part C is not true for the stronger definition of Ω.

22. *Reverse bubble sort* is like bubble sort, but instead of repeatedly percolating the next largest element up, it repeatedly percolates the next smallest element down:

```
for i=2 to n do
    for j=n downto i do
        if A[j]<A[j–1] then EXCHANGE(A[j],A[j–1])
```

Give an exact expression for the number of times the *if* statement is executed (i.e., the number of times the test $A[j]<A[j-1]$ is made) and a formal proof that this expression is $\Theta(n^2)$.

23. *Odd-even bubble sort* has two inner *for* loops; one that checks $A[1]$, $A[3]$, $A[5]$, ... and one that checks $A[2]$, $A[4]$, $A[6]$, ... Although odd-even bubble sort is similar in structure to normal bubble sort, it does not have the same percolation property; in one iteration of the outer *for* loop, an element can move at most two positions up in the array. In addition, unlike normal bubble sort, the inner *for* loops always go all the way to n:

```
repeat
    flag := 0
    for j := 1 to n–1 by 2 if A[j]>A[j+1] then begin
        Exchange A[j] and A[j+1].
        flag := 1
        end
    for j := 2 to n–1 by 2 if A[j]>A[j+1] then begin
        Exchange A[j] and A[j+1].
```

```
        flag := 1
        end
    until flag=1
```

A. Prove that odd-even bubble sort correctly sorts A.

B. Determine an upper for the maximum number of iterations of the repeat loop, as a function of n.

24. *Position sort* copies an array $A[1]...A[n]$ into an array $B[1]...B[n]$ in sorted order by, for each element $A[i]$, computing its position in the sorted order by counting the number of elements that must come before it.

```
for 1 ≤ i ≤ n do begin
    count[i] := 0
    for j := 1 to n do
        if A[j]<A[i] or (j<i and A[j]=A[i]) then count[i] := count[i]+1
    end
for i := 1 to n do B[1+count[i]] := A[i]
```

A. Analyze the time and space used.

B. Prove that position sort places the elements of A into B such that:

1. B is in sorted order; that is, if $i<j$, then $B[i] \le B[j]$.
2. Duplicates in A appear in the same relative order in B; that is, if $i<j$, $A[i]=A[j]$, $A[i]$ is copied to $B[x]$, and $A[j]$ is copied to $B[y]$, then $x<y$.

25. *Displacement sort* copies an array $A[1]...A[n]$ into an array $B[1]...B[n]$ in sorted order by, for each element $A[i]$, counting the number of elements that are out of order with respect to $A[i]$, and then displacing elements up or down according to their counts:

```
for 1 ≤ i ≤ n do begin
    count[i] := 0
    for j := 1 to i–1 do if A[j]>A[i] then count[i] := count[i]–1
    for j := i+1 to n do if A[j]<A[i] then count[i] := count[i]+1
    end
for i := 1 to n do B[i+count[i]] := A[i]
```

A. Analyze the time and space used.

B. Prove that displacement sort places the elements of A into B such that:

1. B is in sorted order; that is, if $i<j$, then $B[i] \le B[j]$.
2. Duplicates in A appear in the same relative order in B; that is, if $i<j$, $A[i]=A[j]$ $A[i]$ is copied to $B[x]$, and $A[j]$ is copied to $B[y]$, then $x<y$.

26. We analyzed the time and space of run-length decoding as a function of the numbe of bits output. Discuss its complexity as a function of the number of input bits.

29 Prove each by applying directly the definitions of O, Ω, and Θ:

A. $53n$ is $\Theta(n)$

B. $n^2\sqrt{n}$ is not $\Omega(n^3)$

C. n^2 is $\Omega(20n)$

D. $2n^4 - 3n^2 + 32n\sqrt{n} - 5n + 60$ is $\Theta(n^4)$

E. $100n^2$ is $O(n^3)$

F. $2n^3$ is $O(2^n)$

G. $10n^{10}$ is $O(2^n)$

H. n^{10} is $\Omega(1000n)$

I. $n^3 + \log_2(n)^{10}$ is $O(n^3)$

J. $n\sqrt{n}\log_2(n)$ is $O(n^2)$

30. The ASCII character code is the most commonly used standard to represent characters in a machine. The code goes from 0 to 127 and includes the lower and uppercase letters of the alphabet, the digits 0 through 9, punctuation marks, and special characters. Although 7 bits suffice to store each character, it is typical to store characters one per byte.

000 nul	001 soh	002 stx	003 etx	004 eot	005 enq	006 ack	007 bel
008 bs	009 ht	010 nl	011 vt	012 np	013 cr	014 so	015 si
016 dle	017 dc1	018 dc2	019 dc3	020 dc4	021 nak	022 syn	023 etb
024 can	025 em	026 sub	027 esc	028 fs	029 gs	030 rs	031 us
032 sp	033 !	034 "	035 #	036 $	037 %	038 &	039 '
040 (	041)	042 *	043 +	044 ,	045 -	046 .	047 /
048 0	049 1	050 2	051 3	052 4	053 5	054 6	055 7
056 8	057 9	058 :	059 ;	060 <	061 =	062 >	063 ?
064 @	065 A	066 B	067 C	068 D	069 E	070 F	071 G
072 H	073 I	074 J	075 K	076 L	077 M	078 N	079 O
080 P	081 Q	082 R	083 S	084 T	085 U	086 V	087 W
088 X	089 Y	090 Z	091 [	092 \	093]	094 ^	095 _
096 `	097 a	098 b	099 c	100 d	101 e	102 f	103 g
104 h	105 i	106 j	107 k	108 l	109 m	110 n	111 o
112 p	113 q	114 r	115 s	116 t	117 u	118 v	119 w
120 x	121 y	122 z	123 {	124 |	125 }	126 ~	127 del

ASCII Character Code