ICOM 4075: Foundations of Computing
Assignment #5 SOLUTIONS

SECTION 3.1

(2) (a) yes: I, D
no: O, C, F, G

(b) yes: I, D,
no: O, C, F, G

(c) yes: I, O, D,
no: C, D, F, G

(d) yes: I, F
no: O, D, C, G.

Input
Output
Definiteness
Correctness
Finiteness
Generality

(4) procedure largest Difference ($a_1, a_2, \ldots, a_n$: integers)

```
maxDif = a1 - a2
for (i = 2 TO n-1)
    maxDif = max(maxDif, ai - ai+1)
return maxDif
```

(8) procedure largestEven ($a_1, \ldots, a_n$: integers)

largest := 0 found = false

for (i = 1 to n)

if (even (a_i))

if not (found)

largest = ai found := true

else if (a_i > largest)

largest = ai

~~if (found)~~ return ai

(10) procedure power (x : real, n : integer)

result := 1 absolute value.

for (i = 1 to |n|)

if (n > 0)

result = result * x

else result = result / x

return result

(24) procedure oneToOne ($(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$: ordered pairs.

oneToOne = ~~false~~ {assume 1-1}

$i := 1$

while $(i < n)$ and ~~one~~ToOne

$j = i$

while $(j < n)$ and oneToOne

if $(a_i \neq a_j)$ and $(b_i = b_j)$

then oneToOne := false

return oneToOne

(26) procedure binary Search (x: integer, $a_1, a_2, \ldots, a_n$ increasing integers)

$i := 1$ stop := false;

$j := n$

while $(i < j)$ and not (stop)

$m := \lfloor (i+j)/2 \rfloor$

if $(x = a_m)$

stop := true

$i = m$

else if $(x > a_m)$ then $i := m+1$

else $j := m$

if $x = a_i$ then location := i

else location := 0

return location.

changes.

Advantage: Algorithm stops as soon as x found.

(32) procedure findTerms ($a_1, a_2, \ldots, a_n$: integers)

```
runningSum := 0    terms := ∅
for (i := 1 to n)
    if (a_i > runningSum)
        then terms := terms ∪ {a_i}
    runningSum := runningSum + a_i
return terms
```

(42) procedure selectionSort ($a_1, a_2, \ldots, a_n$: integers)

```
for (i := 1 to (n - 1))
    min := ~~a_i~~ i
    for (j := i+1 to n)
        if (a_j < a_min) then min := j
    swap (a_i, a_min)
```

(46) Sorting a perfectly reversed (opposite order) list constitutes a worst case scenario for insertion sort.

+ The outise for loop iterates $(n-1)$ times

+
  + The inner while will iterate (j) times since the next element will always end up in the first position (1) in the list.
  + The last for used to shift numbers down the list will iterate $(j-1)$ times as the next element must be inserted in the first position.
  + Each for and while loop contributes one comparison on each iteration.

Total Comparisons = total loop iterations

$$= \underbrace{1}_{\text{outside for}} + \sum_{j=2}^{n}\Big[\underbrace{j}_{\text{while}} + \underbrace{(j-1)}_{\text{shift for}}\Big]$$

$$= 1 + \sum_{j=2}^{n}(2j-1) = 1 + 2\sum_{j=2}^{n} j - \sum_{j=2}^{n} 1 = 1 + \left(2\sum_{j=1}^{n} j\right) - 2 - (n-1)$$

$$= \cancel{1} - \cancel{2} + \cancel{1} - n + \cancel{2}\,\frac{(n)(n+1)}{\cancel{2}} = -\cancel{n} + n^2 + \cancel{n} = \boxed{n^2}$$

(56) The greedy algorithm would generate, for instance, 5 coins for 16¢ in change; 1 twelve + 4 pennies. The minimal result would have been 3 coins; one dime + one nickel + 1 penny.

(64) The Halting Problem (HP) could be <u>reduced</u> to the problem of determining if a program prints the digit 1 (DIGIT1). that is, if we can solve DIGIT1 then we can solve HP. using the following algorithm:

Let HP(P, I) be the algorithm that solves HP for a program P on input I.

⓪ P' = P + "print 1"

① OUTPUT = DIGIT1 (P', I)

② If output = "yes" OUTPUT "P halts on I"

③ If OUTPUT = "NO" OUTPUT "P loops on I"

Section 3.2 : Problems 2, 8, 16, 22, 24, 44

	$\Theta(x^2)$?	C	k
② (a)	yes	2	12
(b)	yes	2	1001
(c)	yes	1	1
(d)	$x^4/2$ no	N/A	N/A
(e)	2^x NO	N/A	N/A
(f)	$\lfloor x \rfloor \cdot \lceil x \rceil$ yes	7	1

Other Alternatives:

(a) $17x + 11 \le Cx^2$

$17x + 11 \le 17x^2 + 11x^2 = 28x^2$ $C = 29$. $k = 1$

(b) $f(x) = x^2 + 1000$

$x^2 + 1000 \le Cx^2$

$x^2 + 1000 < x^2 + 1000x^2 = 1001x^2$ $C = 1002$ $k = 1$

(c) $f(x) = x \log x$

$\log x \le x$ for $x \ge 0$

$x \log x \le x^2$ for $x \ge 0$ $C = 1$ $k = 0$.

(d) $f(x) = x^4/2$ is not $\Theta(n^2)$

Assume $x^4/2$ is $\Theta(n^2)$

Then $\frac{x^4}{2} \le C\cancel{x}^2$ for all $x > k$ for some k.

$\left(\frac{x^2}{2} \le C\right) \to x^2 \le 2C$ ~~which~~ NOT possible for all $x > k$.

(2) Continued . . .

(e) $f(x) = 2^x$ is not $\Theta(n^2)$

The proof requires calculus and limits and is outside the scope of this course

(f) $f(x) = \lfloor x \rfloor \cdot \lceil x \rceil$ is $\Theta(n^2)$

$\lfloor x \rfloor \le x < x+1$

$\lceil x \rceil \le x+1 < x+2$

$\lfloor x \rfloor \cdot \lceil x \rceil < (x+1)(x+2) = x^2+3x+2$ is $\Theta(n^2)$

(8) (a) 4

(b) 5

(c) 0

(d) -1

(16) Show $f(x)$ is $\Theta(x) \rightarrow f(x)$ is $\Theta(x^2)$

Assume $f(x)$ is $\Theta(x)$

$\exists_{K,C}$ $f(x) \le Cx$ for all $x > K$.

But $Cx \le Cx^2$ $\forall x \geq 0$

Thus $f(x) \le Cx^2$ $\forall x \geq K$.

And finally $f(x)$ is $\Theta(x^2)$ for same witnesses used for $\Theta(x)$

(22) $(\log n)^3 \le \sqrt{n}\log n \le n^{99} + n^{98} \le n^{100} \le 1.5^n \le 10^n \le (n!)^2$

(24) Algorithm A is $\Theta(n^2 2^n)$

Algorithm B is $\Theta(n!)$

Algorithm B will eventually take longer.

One way to look at this is the following.

$n^2 \le 2^n \rightarrow n^2 2^n \le 2^n 2^n \le 2^{2n} = 4^n \le n!$

(44) Prove $[f(x)$ is $\Theta(g(x))$ and $g(x)$ is $\Theta(h(x))]$

$\rightarrow [f(x)$ is $\Theta(h(x))]$

Assume $f(x)$ is $\Theta(g(x))$ and $g(x)$ is $\Theta(h(x))$

<u>PART A</u> Prove $f(x)$ is $\Theta(h(x))$

From our assumption we know $f(x)$ is $\Theta(g(x))$ and $g(x)$ is $\Theta(h(x))$

$\rightarrow \exists_{c_1,k_1}\ f(x) \le c_1 g(x)$ for all $x > k_1$

$\rightarrow \exists_{c_2,k_2}\ g(x) \le c_2 h(x)$ for all $x > k_2$.

Thus we can infer that

$f(x) \le c_1(c_2 h(x))$ for all $x > \overbrace{\max(k_1, k_2)}^{k_3}$

$f(x) \le c_1 c_2 h(x)$

Using $c_3 = c_1 c_2$ and $k_3 = \max(k_1, k_2)$ we have

$f(x)$ is $\Theta(h(x))$

The proof for $\Omega(h(x))$ is very similar. ← PART B

Section 3.3 2, 4, 8, 14, 20, 26, 36

(2) Give Big O estimate for # of additions:

t := 0

for i := 1 to n ← one addition per iteration $i := i+1$

for j := 1 to n ← one addition per iteration $j := j+1$

t := t + i + j ← two additions per loop.

$$\text{Iterations} = \sum_{i=1}^{n}\left[1 + \sum_{j=1}^{n} 3\right] = \sum_{i=1}^{n} 1 + n\sum_{j=1}^{n} 3$$

$$= n + n \cdot 3 \sum_{j=1}^{n} 1 = n + 3n^2 = \Theta(n^2)$$

(4) Big O estimate for additions + multiplications.

i := 1

t := 0

while $i \le n$

t := t + i ← 1 addition

i := 2i ← 1 multiplication

Iteration variable i will take values from the sequence $2^0, 2^1, 2^2, \ldots, 2^n$ until $2^k > n$ → $k > \log_2 n$. Thus the while will be repeated $\lceil \log_2 n \rceil$ times.

$$\# \text{ of } +\text{'s } \& *\text{'s} = 2\log_2 n = \Theta(\log n)$$

(8) Given real x and positive k, integer

Alg. A

result := x $\{ x^{2^n} \}$

for $i := \cancel{2}\ 1$ TO k ← k times

result = result * result ← 1 MULT.

Total MULTIPLICATIONS = k. $= \Theta(k)$

Alg. B

result := $\cancel{x}$ 1

for $i := 1$ TO 2^k ← 2^k times

result := result * x ← 1 MULT.

TOTAL MULTIPLICATIONS = 2^k $= \Theta(2^n)$

The first approach is dramatically faster.

(14) (a) $a_2 = 3$ $a_1 = 1$ $a_0 = 1$ $3x^2 + x + 1$ at $c = 2$.

i	y	
N/A	3	before for loop
2 *	7	$(3)(2)+1 = y * c + a_1$
1	15	$(7)(2)+1 = y * c + a_0$

NOTE:
This algorithm has a bug.
The loop should iterate i decreasingly as follows:

for $i := n$ TO 1

(b) The loop iterates n times and each time it does 1 add + 1 MULT.

TOTAL = $2n$ $= \Theta(n)$

(20) Let $T(n)$ be the # of milliseconds to solve problem with input of size n.

	$T(n)$	$T(2n)$		$T(n)$	$T(2n)$
(a)	$\log\log n$	$T(n)+\log(1+\frac{1}{n})$	(e)	n^2	$4T(n)$
(b)	$\log n$	$T(n)+1$	(f)	n^3	$8T(n)$
(c)	$100n$	$2T(n)=200n$			
(d)	$n\log n$	$2T(n)+2n$			

(26) The worst case will occur when the algorithm searches for an element not in the list. Since on every round of comparisons we reduce the list to 1/4 of its size we will end up with $\lfloor \log_4 n \rfloor$ iterations. Each iteration must make 3 comparisons to determine the section of the list where the element is. Thus the total # of comparisons is given by:

$$3\log_4 n = 3\,\frac{\log_2 n}{\log_2 4} = \frac{3}{2}\log_2 n = O(\log n)$$

(36) Consider the greedy algorithm for making change for n cents.

```
procedure change(c_1, c_2, ..., c_r : values of coins ordered decreasingly {},
                 n : a positive integer)
  for i := 1 to r                          <- one comparison
      d_i := 0 {count coins of value c_i}
      while n >= c_i                       <- one comparison
          d_i := d_i + 1
          n := n - c_i
  return (d_1, d_2, ..., d_r)
```

Assume that our algorithm returned $(d_1, \ldots, d_r)$ for an input n.

The for loop must have iterated r times for a total contribution of r comparisons.

The while loop must have iterated $\sum_1^r d_i$ times since on each iteration some d_i is increased by 1. The total comparisons from the while is thus $\sum_1^r d_i$.

Since the algorithm is correct (for quarters, dimes, nickels and pennies) we have:

$$\sum_1^r c_i d_i = n \leq \cancel{\sum_1^r c_1 d_i} \cancel{= c_1 \sum_1^r d_i} \cancel{= n}$$

~~c_1 is the highest value.~~

$$\text{Comparisons} = r + \sum_1^r d_i < r + \sum_1^r c_i d_i = r + n < n + n = 2n$$
$$= \Theta(n)$$