Homework Assignment #5
Solutions

1. Describe an algorithm that takes as input a list of n integers in nondecreasing order and produce the list of all values that occur more than once.

```
procedure duplicates(a1, a2, …, an: integers in nondecreasing order)
    k : =0                          // this variable counts the duplicates
    j := 2
    while j ≤ n
        if aj = aj–1 then
            k := k + 1              // duplicate elements is found
            ck := aj
            while j≤n and ck= aj
                j := j + 1          // search for the next non-duplicate element
        j := j + 1
[c1, c2, …, ck is the output list.]
```

2. Describe an algorithm that takes as input a list of n integers and finds the location of the last even integer in the list or return 0 if there are no even integers in the list.

```
procedure last_even(a1, a2, …, an: integers)
    k := 0
    for i := 1 to n
        if ai is even then k := i
    return k
[k is the location of the last even integer; k = 0 if no even number is found.]
```

3. Describe an algorithm that uses only assignment statements that replaces the triple (x, y, z) with (y, z, x). What is the minimum number of assignment statements needed?

```
temp := x
x := y
y := z
x := temp
```

4. Describe an algorithm that locates the last occurrence of the smallest element in a finite list of integers, where the integers in the list are not necessarily distinct.

```
procedure last_minimum(a1, a2, …, an: integers)
    min := a1
    loc := 1
    for i := 2 to n
        if min ≥ ai then
            min := ai
            loc := i
    return loc
```

5. Specify the steps of an algorithm that locates an element in a list of increasing integers by successively splitting the list into four sublists of equal (or as close to equal as possible) size, and restricting the search to the appropriate piece.

Quarterly Search Algorithm

Input: a sequence of increasing order integers $a_1, a_2, \ldots, a_n$, and searched *key*.

Step 1: Set *low* := 1 and *high* := *n*, i.e., select the entire sequence as the segement denoted by *low* and *high*.
Step 2: Choose the lower quarter location $l := \lfloor (high-low)/4 \rfloor$, middle location $m := \lfloor (high-low)/2 \rfloor$, and upper quarter location $u := \lfloor 3(high-low)/4 \rfloor$.
Step 3: **if** $key < a_m$ **then**
if $key < a_l$ **then** set *high* := *l*–1,
(select the first quarter of the segment)
else set *low* := *l*, and *high*=*m*–1,
(select the second quarter of the segment)
else
if $key < a_u$ **then** set *low* := *m* and *high* := *u*–1,
(select the third quarter of the segment)
else set *low* := *u*,
(select the fourth quarter of the segment)
Step 4: **if** *low*<*high* **then goto** Step 2.
Step 5: **if** $key = a_{low}$ **then** *loc* := *low* **else** *loc* := 0.
Step 6: **return** *loc*.
Output: If the search succeeds then returns *loc* which is the location of *key*; otherwise returns 0.

6. Devise an algorithm that finds the first term of a sequence of positive integers that is less than the immediately preceding term of the sequence.

```
procedure last_minimum(a1, a2, …, an: positive integers)
  loc := 0
  i := 2
  while i ≤ n and loc = 0
    if ai<ai-1 then loc := i
    eles i := i + 1
  return loc
```

7. Use the insertion sort to sort 6, 2, 3, 1, 5, 4, showing the lists obtained at each step.

Red color: sorted elements; green color: inserted element

Input sequence

a_1	a_2	a_3	a_4	a_5	a_6
6	2	3	1	5	4

Phase 1: insert a_2

a_1	a_2	a_3	a_4	a_5	a_6
6	2	3	1	5	4

m=2

a_1	a_2	a_3	a_4	a_5	a_6
6		3	1	5	4

a_1 is the position and move a_1 to the right

a_1	a_2	a_3	a_4	a_5	a_6
	6	3	1	5	4

Insert *m to* a_1, the resulting list at phase 1

a_1	a_2	a_3	a_4	a_5	a_6
2	6	3	1	5	4

Phase 2: insert a_3

a_1	a_2	a_3	a_4	a_5	a_6

2 6 3 1 5 4

m=3

a_1	a_2	a_3	a_4	a_5	a_6
2	6		1	5	4

a_2 is the position and move a_1 to the right

a_1	a_2	a_3	a_4	a_5	a_6
2		6	1	5	4

Insert m *to* a_2, the resulting list at phase 2

a_1	a_2	a_3	a_4	a_5	a_6
2	3	6	1	5	4

Phase 3: insert a_4

a_1	a_2	a_3	a_4	a_5	a_6
2	3	6	1	5	4

m=1

a_1	a_2	a_3	a_4	a_5	a_6
2	3	6		5	4

a_1 is the position and move a_1, a_2, a_3 to the right

a_1	a_2	a_3	a_4	a_5	a_6
	2	3	6	5	4

Insert m *to* a_2, the resulting list at phase 3

a_1	a_2	a_3	a_4	a_5	a_6
1	2	3	6	5	4

Phase 4: insert a_5

a_1	a_2	a_3	a_4	a_5	a_6
1	2	3	6	5	4

m=5

a_1	a_2	a_3	a_4	a_5	a_6
1	2	3	6		4

a_4 is the position and move a_4 to the right

a_1	a_2	a_3	a_4	a_5	a_6
1	2	3		6	4

Insert m *to* a_4, the resulting list at phase 4

a_1	a_2	a_3	a_4	a_5	a_6
1	2	3	5	6	4

Phase 5: insert a_6

a_1	a_2	a_3	a_4	a_5	a_6
1	2	3	5	6	4

m=4

a_1	a_2	a_3	a_4	a_5	a_6
1	2	3	5	6	

a_4 is the position and move a_4 and a_5 to the right

a_1	a_2	a_3	a_4	a_5	a_6
1	2	3		5	6

Insert m *to* a_4, the resulting list at phase 5

a_1	a_2	a_3	a_4	a_5	a_6
1	2	3	4	5	6

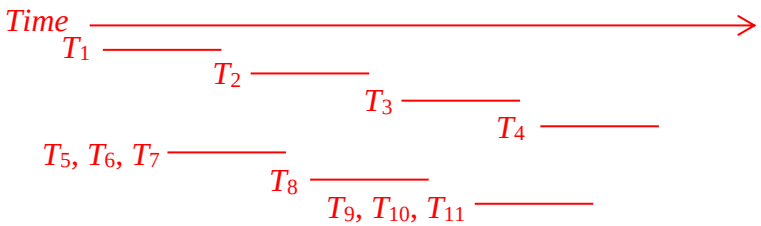
8. Show that if there were a coin worth 12 cents, the greedy algorithm using quarters, 12-cent coins, dimes, nickels, and pennies would not always produce change using the fewest coins possible.

We will give a counterexample to show the greedy algorithm does not use the fewest coins possible in the problem. For the case of 15 cents, the greedy algorithm will use four coins: one 12-cent coin and three pennies. However, only two coins, i.e., one dime and one nickel, will make up to 15 cents.

9. Show that a greedy algorithm that schedules talks in a lecture hall, by selecting at each step the talk that overlaps the fewest other talks, does not always produce an optimal schedule.

We will give a counterexample to show the greedy algorithm that schedules talks in a lecture hall, by selecting at each step the talk that overlaps the fewest other talks, is not optimal.

Consider the following eleven talks T_1 to T_{11}. If the greedy algorithm schedules talks by selecting at each step the talk that overlaps the fewest other talks, T_8, which is overlapped with two talks, must be the first talk scheduled. After T_8 is scheduled, only other two talks T_1 and T_4, each of which is overlapped with three talks, are scheduled. Each of all other talks, T_2, T_3, T_5, T_6, T_7, T_9, T_{10}, and T_{11}, is overlapped with four talks and cannot be scheduled after T_8, T_1, and T_4 are scheduled. Hence, totally, only three talks are scheduled using the greedy algorithm. However, it is possible schedule four talks, i.e., T_1, T_2, T_3, and T_4.



10. Determine whether each of these functions is $O(x^2)$.

a) $f(x) = 17x + 11$ b) $f(x) = x^2 + 1000$ c) $f(x) = x \log x$

d) $f(x) = x^4/2$ e) $f(x) = 2^x$ f) $f(x) = \lfloor x \rfloor \cdot \lceil x \rceil$

a) For all $x > 11$, $f(x) = 17x + 11 \le 17x + x = 18x$. Choosing witness $C = 18$ and $k = 11$, we have $f(x) = 17x + 11$ is $O(x^2)$.

b) For all $x > \sqrt{1000}$, $f(x) = x^2 + 1000 \le x^2 + x^2 = 2x^2$. Choosing witness $C = 2$ and $k = \sqrt{1000}$, we have $f(x) = x^2 + 1000$ is $O(x^2)$.

c) For all $x > 0$, $f(x) = x \log x \le x \cdot x = x^2$. Choosing witness $C = 1$, $k = 0$, we have $f(x) = x \log x$ is $O(x^2)$.

d) If there were a constant C such that $x^4/2 \le Cx^2$ for sufficiently large x, then we would have $C \ge x^2/2$. This clearly impossible because C is a constant. Hence, $f(x) = x^4/2$ is not $O(x^2)$.

e) If there were a constant C such that $2^x \le Cx^2$ for sufficiently large x, then we would have $C \ge 2^x/x^2$. We know that when $x > 10$, $2^x > x^3$. We obtain $C \ge 2^x/x^2 > x^3/x^2 = x$. Clearly, it is impossible because C is a constant. Hence, $f(x) = 2^x$ is not $O(x^2)$.

f) For all $x > 1$, $f(x) = \lfloor x \rfloor \cdot \lceil x \rceil \le x(x + 1) \le x \cdot 2x = 2x^2$. Choosing witness $C = 2$ and $k = 1$, we have $f(x) = \lfloor x \rfloor \cdot \lceil x \rceil$ is $O(x^2)$.

11. Show that $(x^3 + 2x)/(2x + 1)$ is $O(x^2)$.

For all $x > 1$, $\frac{x^3+2x}{2x+1} \le \frac{x^3+2x^3}{2x} = \frac{3}{2}x^2$. Choosing witness $C = 3/2$ and $k = 1$, we have $(x^3 + 2x)/(2x + 1)$ is $O(x^2)$.

12. Let k be a positive integer. Show that $1^k+2^k+\ldots+n^k$ is $O(n^{k+1})$.
For all $n > 1$, $1^k + 2^k + \cdots + n^k \leq n^k + n^k + \cdots + n^k = n \times n^k = n^{k+1}$.
Choosing witness C =1, K = 1, we have $1^k+2^k+\ldots+n^k$ is $O(n^{k+1})$.

13. Give a big-O estimate for each of these functions. For the function g in your estimate that $f(x)$ is $O(g(x))$, use a simple function g of the smallest order.
a) $n \log(n^2 + 1) + n^2 \log n$
b) $(n \log n + 1)^2 + (\log n + 1)(n^2 + 1)$
c) $n^{2^n} + n^{n^2}$
The underline part is the final solution.
a) Since $n^2 + 1$ is $O(n^2)$ by Theorem 1, $n \log(n^2 + 1)$ is $O(n \log(n^2))$. It is the fact that $n \log(n^2) < n^2 \log n$. By Theorem 2, we obtain $n \log(n^2 + 1)+n^2 \log n$ is $\underline{O(n^2 \log n)}$.
b) By the fact that $n \log n + 1$ is $O(n \log n)$, we have $(n \log n + 1)^2$ is $O(n^2 \log^2 n)$. Also, $(\log n + 1)(n^2 + 1)$ is $O(n^2 \log n)$. By Theorem 2, we obtain $(n \log n + 1)^2 + (\log n + 1)(n^2 + 1)$ is $\underline{O(n^2 \log^2 n)}$.
c) For all $n > 4$, we have $2^n > n^2$. Hence, $n^{2^n} > n^{n^2}$, for all $n > 1$, and $n^{2^n} + n^{n^2} < 2n^{2^n}$. The estimation of $n^{2^n} + n^{n^2}$ is $\underline{O(n^{2^n})}$.

14. Give a big-O estimate of the product of the first n odd positive integers.
The n-th odd positive integer is $2n$–1. For all n, $\prod_{i=1}^{n}(2i - 1) = \prod_{i=1}^{n} 2n = (2n)^n$. Therefore, The big-$O$ estimate of the product of the first n odd positive integers is $O((2n)^n)$.

15. Show that if $f_1(x)$ and $f_2(x)$ are functions from the set of positive integers to the set of real numbers and $f_1(x)$ is $\Theta(g_1(x))$ and $f_2(x)$ is $\Theta(g_2(x))$, then $(f_1 f_2)(x)$ is $\Theta(g_1g_2(x))$.
From the definition of big-Θ notation, there are constants C_{11}, C_{12}, C_{21}, C_{22}, k_1, and k_2 such that
$C_{11}|g_1(x)|\leq|f_1(x)|\leq C_{12}|g_1(x)|$ when $x>k_1$ and
$C_{21}|g_2(x)|\leq|f_2(x)|\leq C_{22}|g_2(x)|$ when $x>k_2$.
When x is greater than $\max(k_1, k_2)$, it follows that
$|(f_1f_2)(x)| = |f_1(x)|\,|f_2(x)| \geq C_{11}|g_1(x)|\,C_{21}|g_2(x)| \geq C_{11}C_{21}\,|g_1g_2(x)| = C_1\,|g_1g_2(x)|$,
$|(f_1f_2)(x)| = |f_1(x)|\,|f_2(x)| \leq C_{12}|g_1(x)|\,C_{22}|g_2(x)| \leq C_{12}C_{22}\,|g_1g_2(x)| = C_2\,|g_1g_2(x)|$,
where $C_1=C_{11}C_{21}$ and $C_2=C_{12}C_{22}$. From this inequality, it follows that $f_1(x)\,f_2(x)$ is $\Theta(g_1(x)\,g_2(x))$.

16. Give a big-O estimate for the number of additions used in this segment of an algorithm.

```
t := 0;
for i:=1 to n
  for j:=1 to n
     t := t + i + j
```

The loop has n^2 iterations and each iteration contains two additions. Therefore, its big-O estimate is n^2.

17. Given a real number x and a positive integer k, determine the number of multiplications used to find x^{2^k} starting with x and successively squaring (to find x^2, x^4, and so on). Is this a more efficient way to find x^{2^k} than by multiplying x by itself the appropriate number of times?

If we successively square k times, then we have computed x^{2^k}. Since each square operation requires a multiplication, we need only k multiplications. If we compute x^{2^k} by multiplying x by itself, it would require 2^k-1 multiplications. Therefore, the squaring method is more efficient.

18. Horner's method is an algorithm for evaluating polynomial $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ at $x=c$. The pseudocode of Horner's method is give as the following:

```
procedure Horner(c, a0, a1, a2, …, an: real numbers)
    y := an
    for i :=1 to n
        y := y * c + an-i
    return y  [y = anc^n + an-1c^n-1 + ⋯ + a1c + a0]
```

a) Evaluate $3x^2+x+1$ at $x=2$ by working through each step of the algorithm showing the values assigned at each assignment step.

b) Exactly how many multiplications and additions are used by this algorithm to evaluate a polynomial of degree n at $x=c$?

a) Set y to 3 initially. For $i=1$, y is modified to 3*2+1=7. Then, for $i=2$, y is modified to 7*2+1=15. The returned valued is 15.

b) The **for** loop has n iterations and each iteration contains one multiplication and one addition. Hence, there are n multiplications and n additions.

19. Analyze the worst case of the algorithm you devised in Exercise 28 of Section 3.1 (Question 6) for finding the first term of a sequence less than the immediately preceding term.

The worst case is that the first term of a sequence less than the immediately preceding term is a_n. In this case, the **while** loop is executed n-1 iteration and each iteration performs three comparisons, including loop condition, and an assignment. Hence, the time complexity of procedure *last_minimum* is $O(n)$.

20. Show that the greedy algorithm for making changes for n cents using quarters, dimes, nickels, and pennies has $O(n)$ complexity measured in terms of comparisons needed.

An iteration of the greedy algorithm for making changes performs a comparison and two arithmetic operations, one addition and one subtraction. The number of iterations in this algorithm is a faction of n. Hence, the complexity is $O(n)$.