Section 3.3

1. Give a big-O estimate for the number of operations (where an operation is an addition or a multiplication) used in this segment of an algorithm.

$t := 0$
for $i := 1$ **to** 3 → 3 loops
  for $j := 1$ **to** 4 → 4 loops
    $t := t + ij$ → 2 operations

Total operations: $3 \cdot 4 \cdot 2 = 24$

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

$t := 0$
for $i := 1$ **to** n → n loops
  for $j := 1$ **to** n → n loops
    $t := t + i + j$ → 2 operation

total operations

$n \cdot n \cdot 2 = 2n^2$.

3. Give a big-O estimate for the number of operations, where an operation is a comparison or a multiplication, used in this segment of an algorithm (ignoring comparisons used to test the conditions in the **for** loops, where $a_1, a_2, \ldots, a_n$ are positive real numbers).

$m := 0$
for $i := 1$ **to** n
  for $j := i + 1$ **to** n
    $m := \max(a_i a_j, m)$ → 2 operations

\# of operations

$i=1$. $j=2$ to n → $n-1$ loops with 2 operations ⇒ $2(n-1)$
$i=2$, $j=3$ to n → $n-2$ loops with 2 operation ⇒ $2(n-2)$
⋮
2 operation ⇒ 2
$i=n-1$. $j=n$ to n → 1 loops with 2 operation
$i=n$. $j=n+1$ to n ✗ No operation

$\Rightarrow$ Total operations $= 2(n-1)+2(n-2)+\cdots+2\cdot 1$

$$= 2[(n-1)+(n-2)+\cdots+1] = 2\,\frac{(n-1)n}{2} = (n-1)n$$

$f(n) = (n-1)\cdot n = n^2 - n \Rightarrow |f(n)| < 2|n^2|$

$\Rightarrow f(n)$ is $O(n^2)$.

4. Give a big-O estimate for the number of operations, where an operation is an addition or a multiplication, used in this segment of an algorithm (ignoring comparisons used to test the conditions in the **while** loop).

$i := 1$
$t := 0$
while $i \le n$
  $t := t + i$ → 1 operation
  $i := 2i$ → 1 operation

Since i will double itself in each while loop, that is

$i = 1, 2, 4, 8, \cdots, 2^k$ when there are k steps.

Then once $2^k > n$, the while loop will stop, it implies

that $k > \log_2 n$ is the worse case we will have

Thus, total operations = # of steps time # of operations

$$= \log_2 n \cdot 2 = 2\log_2 n$$

$\Rightarrow f(n) = 2\log_2 n$ and $|f(n)| < 2|\log_2 n|$

$\Rightarrow f(n)$ is $O(\log_2 n)$.

5. How many comparisons are used by the algorithm given in Exercise 16 of Section 3.1 to find the smallest natural number in a sequence of n natural numbers?

```
procedure min(a_1, a_2, ..., a_n: a list of n numbers)
n := the length of {a_i}
temp.min := a_1
for i := 2 to n
    if tempmin > a_i then tempmin := a_i
return tempmin {tempmin is the smallest element}
```

In the worse case scenario, we need $n-1$ comparisons

20. What is the effect in the time required to solve a problem when you double the size of the input from n to $2n$, assuming that the number of milliseconds the algorithm uses to solve the problem with input size n is each of these functions? [Express your answer in the simplest form possible, either as a ratio or a difference. Your answer may be a function of n or a constant.]

a) $\log \log n$ **b)** $\log n$ **c)** $100n$
d) $n \log n$ **e)** n^2 **f)** n^3
g) 2^n

Sol: a) For a "$\log\log n$" algorithm, if input size increases from n to $2n$

we have $\log\log(2n) = \log(\log 2 + \log n)$

$= \log\log 2 + \log\log n$

and it means it keeps the same order because it only increases a constant when we double the input size.

b) For a "log n" algorithm, if input size increase from n to $2n$.

We have $\log(2n) = \log 2 + \log n$

and it means it keeps the same order because it only increases a constant when we double the input size.

c) For a "100n" algorithm, if input size increase from n to $2n$.

We have $100(2n) = 200n = 2\cdot(100n)$

and it means that we need to double the time if we double the input size

d) For a " n logn" algorithm, if input size increase from n to $2n$

We have $(2n)\cdot\log(2n) = 2n\cdot(\log 2 + \log n) = 2n\log 2 + 2n\log n$

and it means that we need more than twice of time we used for input size n but it still keeps the same order.

e) For a "n^2" algorithm, if input size increase from n to $2n$.

We have $(2n)^2 = 4n^2 = \underline{\underline{4}}(n^2)$

and it means that we need to quadruple the time we used for input size n.

f) For a "n^3" algorithm, if input size increase from n to $2n$.

We have $(2n)^3 = 8n^3 = \underline{\underline{8}}(n^3)$

and it means that we need to octuple the time we used for input size n.

g) For a "2^n" algorithm, if input size increase from n to $2n$.

We have $2^{2n} = (2^2)^n = 4^n$ and $\frac{4^n}{2^n} = 2^n$

and it means that we need 2^n times of the time we need for input size n.

22. Determine the least number of comparisons, or best-case performance,

a) required to find the maximum of a sequence of n integers, using Algorithm 1 of Section 3.1.

b) used to locate an element in a list of n terms with a linear search.

c) used to locate an element in a list of n terms using a binary search.

please check our classwork 33, 34, 35

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

please check Quiz 8, Question 2 for answer.