### Problem 1-1

<table>
<thead>
<tr>
<th></th>
<th>1 sec</th>
<th>1 minute</th>
<th>1 hour</th>
<th>1 day</th>
<th>1 month</th>
<th>1 year</th>
<th>1 century</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \log n )</td>
<td>( 2^{10^6} )</td>
<td>( 26.10^6 )</td>
<td>( 2^{36.10^8} )</td>
<td>( 2^{2864.10^8} )</td>
<td>( 2^{2592.10^9} )</td>
<td>( 2^{94608.10^{10}} )</td>
<td>( 2^{94608.10^{12}} )</td>
</tr>
<tr>
<td>( \sqrt{n} )</td>
<td>( 10^{12} )</td>
<td>( 36.10^{14} )</td>
<td>( 1296.10^{16} )</td>
<td>( 746496.10^{18} )</td>
<td>( 6718464.10^{18} )</td>
<td>( 8950673664.10^{20} )</td>
<td>( 8950673664.10^{24} )</td>
</tr>
<tr>
<td>( n )</td>
<td>( 10^6 )</td>
<td>( 6.10^7 )</td>
<td>( 36.10^8 )</td>
<td>( 864.10^9 )</td>
<td>( 2592.10^9 )</td>
<td>( 94608.10^{10} )</td>
<td>( 94608.10^{12} )</td>
</tr>
<tr>
<td>( n \log n )</td>
<td>( 62746 )</td>
<td>( 2801417 )</td>
<td>( 13.10^7 )</td>
<td>( 27.10^8 )</td>
<td>( 67.10^9 )</td>
<td>( 8.10^{10} )</td>
<td>( 69.10^{12} )</td>
</tr>
<tr>
<td>( n^2 )</td>
<td>( 1000 )</td>
<td>( 7445 )</td>
<td>( 60000 )</td>
<td>( 293938 )</td>
<td>( 1609968 )</td>
<td>( 30758413 )</td>
<td>( 307584134 )</td>
</tr>
<tr>
<td>( n^3 )</td>
<td>( 100 )</td>
<td>( 391 )</td>
<td>( 1532 )</td>
<td>( 4420 )</td>
<td>( 13736 )</td>
<td>( 98169 )</td>
<td>( 455661 )</td>
</tr>
<tr>
<td>( 2^n )</td>
<td>( 19 )</td>
<td>( 25 )</td>
<td>( 31 )</td>
<td>( 36 )</td>
<td>( 41 )</td>
<td>( 49 )</td>
<td>( 56 )</td>
</tr>
<tr>
<td>( n! )</td>
<td>( 9 )</td>
<td>( 11 )</td>
<td>( 12 )</td>
<td>( 13 )</td>
<td>( 15 )</td>
<td>( 17 )</td>
<td>( 18 )</td>
</tr>
</tbody>
</table>
PROBLEM 2-4

a. (8, 6), (2, 1), (3, 1), (8, 1), (6, 1)

b. The array with the descending order has the most inversions. Since all pairs are inversions the number of inversions is \( \binom{n}{2} \).

c. Let \( A \) be an array with \( n \) elements.

Let \( inv_i \) be the number of inversions of \( A \) with the second element \( A_i \).

Let \( INV \) be the number of all inversions in the array. Then, \( INV = \sum_{i=2}^{n} inv_i \)

We can see that the number of iterations of the while loop in the pseudocode of insertion sort is \( inv_i \). Then we can denote the running time of the insertion sort as:

\[
T(n) = \sum_{i=2}^{n} (\theta(1) + \theta(inv_i)) = \sum_{i=2}^{n} \theta(1) + \sum_{i=2}^{n} \theta(inv_i) = \theta(n) + \theta(INV) = \theta(n + INV)
\]
d. The following algorithm finds the number of inversions with the parameters (A, 1, n)

```
Find-Inversions (A, left, right)
    if left<right
        middle = (left+right)/2
        inversions = Find-Inversions(A, left, middle)
        inversions = inversions + Find-Inversions(A, middle+1, right)
        inversions = inversions + Merge(A, left, middle, right)
    return inversions

Merge (A, left, middle, right)
    n1 = middle-left+1
    n2 = right-middle
    Copy the elements A[left..middle] to L[1..n1]
    Copy the elements A[middle+1..right] to R[1..n2]
    inversions = 0
    i = 1
    j = 1
    k = left
    while i<=n1 and j<=n2
        if L[i] <= R[j]
            A[k] = L[i]
            i = i+1
            inversions = inversions + (j - 1)
        else
            A[k] = R[j]
            j = j+1
        k = k+1
    if i>n1
        Copy R[j..n2] to A[middle+j..right]
    else
        Copy L[i..n1] to A[right-n1+i..right]
    inversions = inversions + (j-1)*(n1-i+1)
    return inversions
```
Problem 3-4

a. False. Counter-example: \( f(n) = n \) and \( g(n) = n^2 \).

b. False. Counter-example: \( f(n) = n \) and \( g(n) = n^2 \).

c. True.
\[ f(n) = O(g(n)) \Rightarrow \exists c \text{ s.t. } f(n) \leq c \cdot g(n) \text{ for sufficiently large } n. \]
If we get the log of both sides, we have \( \log(f(n)) \leq \log(c \cdot g(n)) = \log(c) + \log(g(n)) \)
Since \( \log(g(n)) \) is a non-convergent monotonically increasing function, and \( \log(c) \) is constant, there exists an \( n_0 \), such that \( \log(c) \leq \log(g(n)) \) for \( n > n_0 \).

d. False. Counter-example: \( f(n) = n \) and \( g(n) = n/2 \)

e. False. Counter-example: \( f(n) = 1/n \)

f. True.
\[ f(n) = O(g(n)) \Rightarrow \exists c \text{ s.t. } f(n) \leq c \cdot g(n) \Rightarrow 1/c \cdot f(n) \leq g(n) \]

g. False. Counter-example: \( f(n) = 2^n \)

h. True.
\[ g(n) = o(f(n)) \Rightarrow \exists c \text{ s.t. } g(n) \leq c \cdot f(n) \text{ for sufficiently large } n. \]
Therefore \( f(n) \leq g(n) + f(n) \leq (1+c) \cdot f(n) \)
PROBLEM 4-6

a. Necessity proof:
Let \( k = i+1 \) and \( l = j+1 \).
Then we have \( A[i, j] + A[i+1, j+1] \leq A[i+1, j] + A[i, j+1] \)

Sufficiency proof:
**Induction on columns**
**Basis:**

**Induction hypothesis:**

From the basis we have:

From the induction hypothesis, we have

When we add (1) and (2) side by side we yield:

**Induction on rows**
**Basis:**

**Induction hypothesis:**
Suppose \( A[i, j] + A[i+r, l] \leq A[i+r, j] + A[i, l] \) is correct where \( 0 < i+r < k \)

From the basis we have:

From the induction hypothesis, we have

When we add (3) and (4) side by side we yield:

QED

b. There are several different solutions one of which is incrementing \( A[1, 3] \) by three.
c. Suppose that the expression is not correct. Then there exists an \( f(i) \) s.t. \( f(i) > f(i+1) \). This yields \( A[i, f(i+1)] + A[i+1, f(i)] > A[i, f(i)] + A[i+1, f(i+1)] \), which is a contradiction. QED

d. Since we find all the even-numbered rows results recursively, we just need to search over the odd numbered rows. We can continue this step until \( m \) becomes 1. Let we assign \( f(0) = 0 \) and \( f(m+1) = n \). Then for an odd-numbered row \( 2*i-1 \) where \( 1 \leq 2*i-1 \leq n \), we can just find the minimum element in \( A[2*i-1, f(2*i-2), f(2*i)] \). The running time of one iteration is:

\[
O\left( \sum_{i=1}^{[m/2]} (f(2i) - f(2i - 2) + 1) \right) = O(m + n)
\]

e. 

\[
T(m, n) = T\left( \frac{m}{2}, n \right) + O(m + n) = \sum_{i=1}^{\log m} O\left( \frac{m}{2^i} + n \right) = O(n \log m) + O\left( \sum_{i=1}^{\log m} \frac{m}{2^i} \right) = O(n \log m) + O(2m) = O(m + n \log m)
\]
PROBLEM 7-2

a. When all elements are equal, randomized partition divides the elements into two with \( n-1 \) and 0 elements. So \( T(n) = T(n-1) + \Theta(n) = \Theta(n^2) \).

b. 
\[
\text{Partition'}(A, p, r) \\
x = A[r] \\
i = p-1 \\
k = i \\
\text{for } j=p \text{ to } r-1 \\
\quad \text{if } A[j] < x \\
\quad \quad k = k+1 \\
\quad \quad \text{exchange } A[j] \text{ with } A[k] \\
\quad \quad i = i+1 \\
\quad \quad \text{exchange } A[i] \text{ with } A[k] \\
\quad \text{else if } A[j] == x \\
\quad \quad k = k+1 \\
\quad \quad \text{exchange } A[j] \text{ with } A[k] \\
\text{exchange } A[k+1] \text{ with } A[r] \\
\text{return } (i+1, k+1)
\]

c. Randomized-Partition's code is the same except it returns two values.

\[
\text{Randomized-Quicksort'}(A, p, r) \\
\text{if } p<r \\
\quad (q, t) = \text{Randomized-Partition'}(A, p, r) \\
\quad \text{Randomized-Quicksort'}(A, p, q-1) \\
\quad \text{Randomized-Quicksort'}(A, t+1, r)
\]

d. Let \( n' \) be the number of distinct elements in the array. Let \( z_i', z_2', z_3' \ldots \) be the sorted array of distinct numbers in the array. Let \( n_i \) be the number of occurrences of \( z_i' \). Let \( m_i \) be the number of elements of the array that are lower than \( z_i' \).

Since an element which is equal to the pivot finds its final place just after the partition, it cannot be chosen as a pivot later. Therefore two values can be compared at most in one partition method. The probability of comparing \( z_i' \) with \( z_j' \) is \( \frac{n_i}{m_j - m_i + n_j} \) if \( z_i' \) is chosen as the pivot and the number of comparisons will be \( n' \). A record with a value is compared with the other records with the same value \( n_i - 1 \) times. Hence the expected value is:

\[
E[X] = \sum_{i=1}^{n'-1} \sum_{j=i+1}^{n'} \frac{2n_in_j}{m_j - m_i + n_j} + \sum_{i=1}^{n'} (n_i - 1) = n - n' + 2 \sum_{i=1}^{n'-1} \sum_{j=i+1}^{n'} \frac{n_in_j}{m_j - m_i + n_j}
\]