1. **Comparison of running times**: For each function $f(n)$ and time t in the following table, determine the largest size n of a problem that can be solved in time t, assuming that the algorithm to solve the problem takes $f(n)$ microseconds.

A	B		

1 second, 1 minute, 1 hour, 1 day, 1 month, 1 year, 1 century.
$\lg n$, $\sqrt{n}$, n, $n \lg n$, n^2, n^3, 2^n, $n!$

For $f(n) = \lg n$. 1 second $= 10^6$ microseconds.

If we can solve a problem of size n, and the operation takes $f(n) = \lg n$, ten 1 second is $\lg n = 10^6$.

$f(n) = \lg n$

n = 2 will be 1 microsecond.

n = 4 will be 2 microseconds.

n = 8 will be 3 microseconds.

n = 256 will be 8 microseconds.

n = 65,536 will be 16 microseconds.

n = 16,777,216 will be 24 microseconds.

Since 1 second has 10^6 microseconds, we can solve a problem of size $n = 2^{10^6}$ in 1 second.

$f(n) = \lg n$

1 second is $n = 2^{10^6}$.

1 minute is $n = 2^{6 \times 10^7}$.

1 hour is $n = 2^{3.6 \times 10^9}$.

1 day is $n = 2^{8.64 \times 10^{10}}$

1 month is $n = 2^{2.59 \times 10^{12}}$

1 year is $n = 2^{3.11 \times 10^{13}}$

1 century is $n = 2^{3.11 \times 10^{15}}$

===

$f(n) = \sqrt{n}$

1 second is $n = 10^{12}$.

1 minute is $n = (60 \times 10^6)^2 = (6 \times 10^7)^2 = (36 \times 10^{14}) = 3.6 \times 10^{15}$.

1 hour is $n = (60 \times 6 \times 10^7)^2 = (3.6 \times 10^9)^2 = 12.96 \times 10^{18} = 1.296 \times 10^{19}$

1 day is $n = (24 \times 3.6 \times 10^9)^2 = (8.64 \times 10^{10})^2 = 74.6496^{20} = 7.465 \times 10^{21}$

1 month is $n = (30 \times 8.64 \times 10^{10})^2 = (259.2 \times 10^{10})^2 = (2.592 \times 10^{12})^2 = 6.7185 \times 10^{24}$

1 year is $n = (12 * 2.592 \times 10^{12})^2 = (31.104 \times 10^{12})^2 = (3.11 \times 10^{13})^2 = 9.672 \times 10^{26}$.

Working.

2. **Insertion sort on small arrays in merge sort**: Although merge sort runs in $\Theta(n \lg n)$ worst-case time and insertion sort runs in $\Theta(n^2)$ worst-case time, the constant factors in insertion sort make it faster for small n. Thus, it makes sense to use insertion sort within merge sort when subproblems become sufficiently small. Consider a modification to merge sort in which n/k sublists of length k are sorted using insertion sort, and then merged using the standard merging mechanism, where k is a value to be determined.

a. Show that the n/k sublists, each of length k, can be sorted by insertion sort in $\Theta(nk)$ worst-case time.

b. Show that the sublists can be merged in $\Theta(n \lg(n/k)$ worst-case time.

c. Given that the modified algorithm runs in $\Theta(nk + n \lg(n/k)$ worst-case time, what is the largest asymptotic (Θ-notation) value of k as a function of n for which the modified algorithm has the same asymptotic running time as standard merge sort?

d. How should k be chosen in practice?

Working.

3. **Inversions**: Let $A[1..n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$ then the pair $(i.j)$ is called an **inversion** of A.

a. List the five inversions of the array $(2, 3, 8, 6, 1)$.

b. What array with elements from the set $\{1, 2, ..., n\}$ has the most inversions? How many does it have?

c. What is the relationship between the running time of insertion sort and the number of inversions? Justify your answer.

d. Give an algorithm that determines the number of inversions in any permutation on n elements in $\Theta(n \lg n)$ wore-case times. (Hint: Modify merge sort).

The five inversions of $A = (2, 3, 8, 6, 1)$ are $(2, 1)$, $(3, 1)$, $(8, 6)$, $(8, 1)$ and $(6, 1)$.

Working.