Question 1

a) What kind of input will yield the worst possible running time, for any fixed size n? Why?

Any input array sorted in decreasing order will yield the worst possible running time for that number of elements. This is because the sorting algorithm will execute the swapping process on every comparison in every pass through the array.

b) Let T(n) be the number of primitive operations performed in this worst case. Write, for each line of the pseudocode, the number of primitive operations executed and the type (assignment, condition, arithmetic...) of each. Sum up everything to obtain T(n), assuming all primitive operations take the same time. If T(n) contains summations, replace them by their explicit value. Show your work step by step like we did in class.

Running Time:

Pseudocode	T-operations	
for i←0 to n-1 do	T_{assign} +(n)($T_{compare}$)+(n)(T_{assign})	
for j←0 to n-1-i do	T_{assign} +(n-i)($T_{compare}$) +(n-i)(T_{assign})	Repeated n times
if (A[j+1]<A[j]) then	$T_{compare}$ + T_{index} + T_{index}	In the worst case, this is repeated $\sum_{i=0}^{n-1} n-1-i$ times
tmp←A[j]	T_{assign} + T_{index}	
A[j]←A[j+1]	T_{assign} + T_{index} + T_{index}	
A[j+1]←tmp	T_{assign} + T_{index}	

$$T(n) = (1 + n + n) + \left(n + \sum_{i=1}^{n} 2(n-i+1)\right) + \left(10\left(\sum_{i=0}^{n-1} n-1-i\right)\right)$$

$$T(n) = 1 + 3n + 2\left(n^2 - \sum_{i=1}^{n} i + n\right) + 10\left(n^2 - \sum_{i=1}^{n} i\right)$$

$$T(n) = 1 + 3n + 2\left(n^2 - \left(\frac{n(n+1)}{2}\right) + n\right) + 10\left(n^2 - \left(\frac{n(n+1)}{2}\right)\right)$$

$$T(n) = 1 + 3n + 2n^2 - 2\left(\frac{n(n+1)}{2}\right) + 2n + 10n^2 - 10\left(\frac{n(n+1)}{2}\right)$$

$$T(n) = 1 + 5n + 12n^2 - 12\left(\frac{n(n+1)}{2}\right)$$

c) Give the simplest and most accurate big-Oh representation for T(n).

$$Given: T(n) = 1 + 5n + 12n^2 - 12\left(\frac{n(n+1)}{2}\right), big - O\ representation: O(n^2)$$

Question 2

For each of the algorithms below, indicate the running time using the simplest and most accurate big-Oh notation. Assume that all arithmetic operations can be done in constant time. The first algorithm is an example. No justifications are needed.

Algorithm	Running time in big-O notation
Algorithm Example(n) $x \leftarrow 0$ **for** $i \leftarrow 1$ **to** n **do** . $x \leftarrow x + 1$	$O(n)$
Algorithm algo1(n) $i \leftarrow n$ **while** $(i > 1)$ **do** . $i \leftarrow i/2$	$O(\log n)$
Algorithm algo2(n) $i \leftarrow 1$ **while** $i < n$ **do** . $i \leftarrow i + 100$	$O(n)$
Algorithm algo3(n) $k \leftarrow 1$ **for** $i \leftarrow 1$ **to** 100 . **for** $j \leftarrow 1$ to i . $k \leftarrow (k + i - j) * (2 + i + j)$	$O(1)$
Algorithm algo4(n) $x \leftarrow 0$ **for** $i \leftarrow 1$ **to** n **do** . **for** $j \leftarrow 1$ **to** i **do** . $x \leftarrow x + 1$	$O(n^2)$

Question 3

Prove using only the definition of $O()$ that $5n + 3log(n)$ is $O(n)$. Write your proof using proper mathematical formalism, as done in the lecture notes.

To prove that $f(n) = 5n + 3log(n)$ is $O(n)$, we must show that we can find a constant c and a number n_0 such that $f(n) < c(n)$ for all $n > n_0$.

Premise: $y = log(x)$ grows slower than $y = x$, so for very large numbers of n, the $log(n)$ component of the function $f(n)$ can be ignored.

Premise: $5x + 3log(x) \geq 5x$ for all $x \geq 1$

Premise: $5x + 3log(x) \leq 6x$ for all x

It stands to reason that there must exist some constant c; $\{5 < c < 6\}$ for which the function $g(n) = cn$ is always greater than the function $f(n) = 5n + 3log(n)$ beyond a certain number n_0.

Question 4

Prove using only the definition of $O()$ that $(n+10)^{1.5}+10n+3$ is $O(n^2)$. Write your proof using proper mathematical formalism, as done in the lecture notes.

To prove that $f(n)=(n+10)^{1.5}+10n+3$ is $O(n^2)$, we must show that we can find a constant c and a number n_0 such that $f(n)<c(n^2)$ for all $n>n_0$.

Premise: At very large numbers of n, the 10 in the $(n+10)^{1.5}$ component of $f(n)$ can be ignored.

Premise: $x^{1.5}\geq c\sqrt{x}$ for all positive constants c beyond a certain point x_0

Premise: $x^{1.5}\leq cx^2$ for all positive constants c beyond a certain point x_0

It stands to reason, that there must exist some (possibly large) constant for which the function $g(n)=cn^2$ is always greater than the function $f(n)=(n+10)^{1.5}+10n+3$ beyond a certain number n_0. And that based on the hierarchy of Big-O classes, this is the tightest possible Big-O notation for $f(n)$.

Question 5

Prove using only the definition of $O()$ that 3^n is not $O(2^n)$. Write your proof using proper mathematical formalism, as done in the lecture notes.

To prove that $f(n)=3^n$ is not $O(2^n)$, we must show that for any value c and a number n_0 we can find a value of n such that $f(n)>c(2^n)$.

We will assume that we have a given constant c and number n_0.

I will choose a value $n=c+1$ so that:

$$3^{c+1}>c2^{c+1}>3^c>c2^n$$

$$3(3^c)>2c(2^c)\ >3^c>c2^n$$

It stands to reason that for all constants c, $f(n)=3^n$ will eventually catch up to $g(n)=c2^n$

Question 6

Find two non-negative functions f and g such that $f(n)$ is not $O(g(n))$ and $g(n)$ is not $O\big(f(n)\big)$.

$f(n)=n\sin n$; $g(n)=n\cos n$ or alternatively $f(n)=n\sin^2 n$; $g(n)=n\cos^2 n$ if you prefer a function that does not return negative values for $x>0$. Essentially, any two functions that oscillate out of phase indefinitely between 0 and whatever value cannot be big-O of each other because they will perpetually cross each other returning to 0.