# Problem Set 1

## Problem 1.1

**Solution:**
(a) $f(n) = 3n$ $g(n) = n^3$
$f \notin \Theta(g)$ because there is no lower bound $c_1$
$f \in O(g)$ because there is an upper bound when $n > 1$ and $c = 1$.
$f \in o(g)$ because $lim_{x\to\infty} 3n/n^3 = 0$
$f \notin \Omega(g)$ because there is no lower bound constant $c$.
$f \notin \omega(g)$ because $lim_{x\to\infty} 3n/n^3 \neq \infty$

$g \notin \Theta(f)$ because there is no upper bound $c_2$
$g \notin O(f)$ because there is no upper bound constant $c$
$g \notin o(g)$ because $lim_{x\to\infty} n^3/3n \neq 0$
$g \in \Omega(g)$ because there is a lower bound $c = 1$ when $n > 1$
$g \notin \omega(f)$ because $f \in O(g)$

(b) $f(n)$ grows faster than $g(n)$ because $lim_{x\to\infty} f(n)/g(n) = \infty$.
$f(n) \geq c.g(n)$ when $c > 0$ and $n > 1$.
$g \in o(f)$
$g \in O(f)$
$f \in \Omega(g)$
$f \in o(g)$

(c)$f(n)$ grows faster than $g(n)$ because $lim_{x\to\infty} f(n)/g(n) = lim_{x\to\infty} n/(\log(n)^2) = \infty$
$f(n) \geq c.g(n)$ when $c > 0$ and $n > 1$.
$f \in \Omega(g)$
$f \in \omega(g)$
$g \in o(f)$
$g \in O(f)$

(d)When we plot graphs for $f$ and $g$, we conclude that:
$f \in \Omega(g)$
$f \in \omega(g)$
$g \in o(f)$
$g \in O(f)$

## Problem 1.2

**Solution:**
(a) See the file selection_sort.py for implementation

```
#Selection sort
def selectionSort(self,A):
  # Traverse through all array elements
  for i in range(len(A)):
  # Find the minimum element in remaining
  # unsorted array
    min_pos = i
    for j in range(i+1, len(A)):
      if A[min_pos] > A[j]:
        min_pos = j
    # Swap the found minimum element with
    # the element at position i
    A[i], A[min_pos] = A[min_pos], A[i]
```

(b) Loop invariant: At the start of each for loop, elements in the array until and including position $i - 1$ are sorted.
Initialization: When $i = 0$, there are no elements before it to sort. When $i = 1$, there is only one element until $i - 1$, therefore, it is sorted.
Maintenance: We assume that every element in the array before $i$ is sorted. Now, in every iteration of the first for loop, we find the smallest element in the array after $i$ (by storing the position of the smallest element in $min\_pos$ and updating it upon comparison). We then swap that element with the element in position $i$. Now, all elements before and including element $i$ are sorted, so

we can continue into the next iteration.
Termination: After the last iteration, all elements upto and including $n$ are sorted. Therefore, the algorithm is correct.

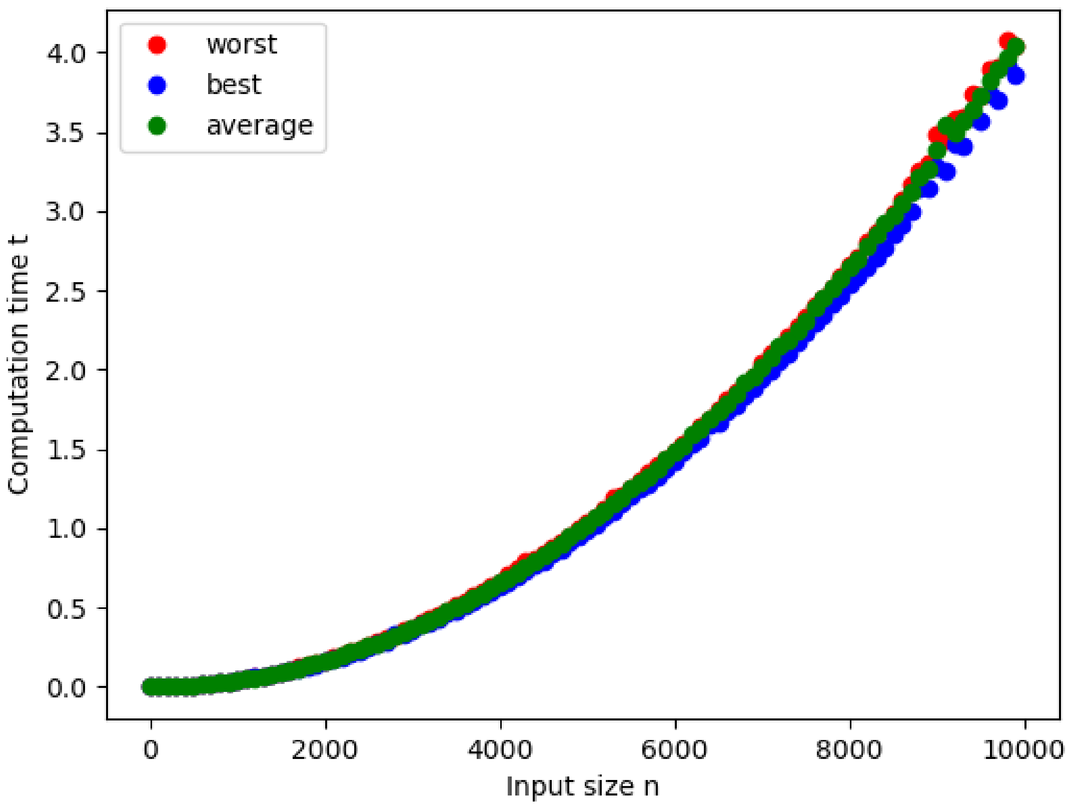
(c) See the file selection_sort.py for implementation

```
import random as rd
#Generates a reverse sorted list n...0
def genReverseSorted(self,n):
  A=list(range(n))
  A=list(reversed(A))
  return A

#Generates a sorted list 0...n
def genSorted(self,n):
  A=list(range(n))
  return A

#Generates a random list of size n with range a,b:
def genRandom(self,a,b,n):
  A=rd.sample(range(a,b),n)
  return A
```

(d)See the file selection_sort.py for implementation




(e)The time complexity for all best, worst and average cases for selection sort is $O(n^2)$.