Solutions to Homework 1

CSE 551: Foundations of Algorithms

1. For $O(n^2)$, use brute force to calculate the absolute sum of all possible pairs of numbers in the array and output the sum with the minimum absolute value.

   A better approach would be to sort the array at first and traverse from the left and right till you find a pair that has a sum closest to zero. Note that since we are considering the absolute sum, we cannot have a sum that is lower than zero.

```
ABSOLUTE_SUM (A, beg, end)

begin
        MERGE_SORT (A, beg, end)

        left ← 1
        right ← length
        min ← ∞

        while ( left < right )
        begin
                sum ← A_left + A_right
                if min > sum
                then min ← sum
                if sum ≥ 0
                then r ← r - 1
                else l ← l + 1
        end while
        return min
end
```

   The time complexity of the algorithm is $O(n \log n)$ for **MERGE_SORT** and $O(n)$ for the traversal, which gives us a combined running time of $O(n \log n)$.

2. $f_2(n) < f_3(n) < f_6(n) < f_1(n) < f_4(n) < f_5(n)$
3. $g_1(n) < g_3(n) < g_4(n) < g_5(n) < g_2(n) < g_7(n) < g_6(n)$

4. **i**. **False**.

   Disprove by counterexample.

Assumption: $f(n)$ is $O(g(n))$, i.e. there exist constants c and N such that $|f(n)| \leq c\ |g(n)|$ for $n > N$.

We need to check if $log_2 f(n)$ is $O(log_2(g(n))$, i.e. $log_2 f(n) \leq c\ log_2(g(n))$.
Take $f(n) = 2$ and $g(n) = 1$. So we get:

$$log_2 2 \ \leq\ c\ log_2 1$$
$$\Rightarrow \quad 1 \leq c\ .0$$
$\Rightarrow \quad 1 \leq\ 0$, which fails for all values of $c > 0$.

ii. **False**.

Disprove by counterexample.

Assumption: $f(n)$ is $O(g(n))$, i.e. there exist constants c and N such that $|f(n)| \leq c\ |g(n)|$ for $n > N$.

Take $f(n) = 3n$ and $g(n) = n$. This holds under the assumption that $f(n)$ is $O(g(n))$ for all n, $3n \leq Cn$ for any

constant $C > 4$.

We need to check if :

$$2^{f(n)} \text{ is } O(2^{g(n)})$$
$\Rightarrow \quad 2^{3n} \leq c.2^n$, which fails for all values of $c > 0$.

iii. **True.**

Assumption: $f(n)$ is $O(g(n))$, i.e. there exist constants c and N such that $|f(n)| \leq c\ |g(n)|$ for $n > N$.

For $n > 0$, $f(n)^2 < (c.\ g(n))^2$ as $f(n)^2 \leq c^2.g(n)^2$ for all $c > 0$.

5. **i.** Since the first two for loops each iterate n times, we have n^2 for both the **for** loops. Now for the inner loop which computes the sum from $A[i]$ to $A[j]$, we can assume that it can make at most n iterations.

So, $n \times n \times n = n^3$. All the other operations take constant time.

So, we can choose $f(n) = O(n^3)$. We can justify this by saying that the algorithm will never have a running time worse than $O(n^3)$.

Not that we can find a function which is always exceed the running time of the algorithm, but the art lies in providing a better bound.

ii. We can exactly count the number of additions taking place.

Suppose $i = 1$.

When $j = 2$, we will have 1 addition.
When $j = 3$, we will have 2 additions.
.
.

.
When j = n, we will have n-1 additions.

So number of addition operations = $1 + 2 + \ldots + (n\text{-}1) = n(n\text{-}1)/2$.
But we have to iterate over i for n times.
So number of operations = $\{ n\ (n\text{-}1)/2 \} * n = (n^3 - n^2)/2 = f(n)$.
Now prove that $f(n)$ is $\Omega(n^3)$ using the definition.

iii. We can't really change the two outer loops as all they do is traverse the 2-D array. But what we can do is improve on the portion where we are traversing $A[i]$ to $A[j]$ which adds to the overall time-complexity.

A simple approach would be to pre-compute the values in the outer loops so that we do not have to include an inner loop. Consider the following algorithm:

```
MATRIX_SUM (A, n)

begin

        do B[i][j] ← 0 for all i, j ≤ n

        for i ← 1 to n
        begin
                sum ← A[i]
                for j ← i+1 to n
                begin
                        sum ← sum + A[j]
                        B[i][j] ← sum
                end for
        end for
end
```

Clearly, this omits the repeated traversal of the 1-D array and the precomputations help in improving the running time to $O(n^2)$.