COMP 3711 Design and Analysis of Algorithms
Solutions to Assignment 3

1. The algorithm: Put the first base station at $x + 4$ where x is the coordinate of the first house. Remove all the houses that are covered and then repeat if there are still houses not covered.

   Correctness: Let X be the solution returned by this greedy algorithm, and let Y be an optimal solution. Consider the first base station where Y is different from X. Suppose the base station in X is located at x and the one in Y is located at y. By the greedy choice, we must have $x > y$. Now move y to x in Y . The resulting Y must still cover all houses. Repeatedly applying this transformation will convert Y into X. Thus X is also an optimal solution.

2. Define $c[i]$ to be the length of the longest increasing subsequence that ends at x_i. Note that the length of the longest increasing subsequence in X is $\max_{1\le i\le n} c[i]$. The longest increasing subsequence that ends with x_i has the form $< Z, x_i >$ where Z is the longest increasing subsequence that ends with x_r for some $r < i$ and $x_r \le x_i$. Thus, we have the following recurrence relation:

$$c[i] = \begin{cases} 1 & \text{if } i = 1 \\ 1 & \text{if } x_r > x_i \text{ for all } 1 \le r < i \\ \max_{\substack{1\le r<i \\ x_r \le x_i}} c[r] + 1 & \text{if } i > 1 \end{cases}$$

   We compute all the $c[i]$'s from $i = 1$ to n. Evaluating the recurrence takes $O(n)$ time, so the total running time is $O(n^2)$.

   In order to report the optimal subsequence we need to store for each i, not only $c[i]$ but also the value of r which achieves the maximum in the recurrence relation. Denote this by $r[i]$. If $c[i]$ is a base case, we set $r[i] = 0$. Then we can trace the solution as follows. Let $c[k] = \max_{1\le i\le n} c[i]$. Then x_k is the last number in the optimal subsequence. Then we update $k \leftarrow r[k]$. Now x_k is the second to last number. Then update $k \leftarrow r[k]$ again and repeat the process until $k = 0$.

3. This problem is similar to the 0-1 Knapsack problem, except that there is no "value" that we want to optimize. We just want to check if it possible to fully pack the knapsack. Thus we use a Boolean array, and define $B[i, w] = true$ if there is a subset of integers in $\{a_1, \ldots, a_i\}$ that adds up to w. The recurrence is thus

$$B[i, w] = B[i - 1, w] \text{ or } B[i - 1, w - a_i].$$

   The base case is $B[0, w] = false$ for any w, $B[i, w] = false$ for any $w < 0$, and $B[i, 0] = true$ for any i. We can compute all the $B[i, w]$'s from $i = 1$ to n, and for each i, we compute each $B[i, w]$ from $w = 1$ to W. The total running time is thus $O(nW)$. Finally, we return $B[n, W]$.

4. Define $L[i, j]$ to be the length of the longest symmetric subsequence for the substring $x[i, ..., j]$. If we look at $x[i, .., j]$, then we can find a symmetric of length at least 2 if $x[i] = x[j]$. If they are not same then we seek the longest symmetric subsequence in $x[i + 1, ..., j]$ or $x[i, ..., j - 1]$. Also every character $x[i]$ is a symmetric in itself of length 1. Therefore, we have the following recurrence:

$$L[i, j] \quad = \quad \begin{cases} L[i + 1, j - 1] + 2 & \text{if } x[i] = x[j] \\ \max\{L[i + 1, j], L[i, j - 1]\} & \text{otherwise} \end{cases}$$

$$
\begin{aligned}
L[i,i] &= 1 \;\; \forall i \in (1,...,n) \\
L[i,i-1] &= 0 \;\; \forall i \in (2,...,n)
\end{aligned}
$$

We compute all $L[i,j]$ from shorter strings to longer strings, similar to the optimal BST problem. More precisely, we first compute all $L[i,j]$ such that $j-i=1$, then all $L[i,j]$ such that $j-i=2$, $\ldots$, until we have $A[1,n]$. It takes $O(1)$ time to compute each $L[i,j]$, so the total running time is $O(n^2)$.

To find the actual longest symmetric subsequence, we keep all the choices for each $L[i,j]$ in an array $r[i,j]$. Note that each $L[i,j]$ has been computed from one of 3 choices. Then we start from $r[1,n]$ and print out the symmetric using the following algorithm:

PRINT(L,r,i,j):
if $L[i,j]=1$
    print $x[i]$
    return
if $r[i,j]=1$
    print $x[i]$
    PRINT$(L,r,i+1,j-1)$
    print $x[j]$
if $r[i,j]=2$
    PRINT$(L,r,i+1,j)$
if $r[i,j]=3$
    PRINT$(L,r,i,j-1)$