A3Q1

a)
1. (d1, h1), (d2, h2), (d3, h3): Unstable. h3 prefers d2 to d3 and d2 prefers h3 to h2.
2. (d1, h1), (d2, h3), (d3, h2): Unstable. h1 prefers d2 to d1 and d2 prefers h1 to h3.
3. (d1, h2), (d2, h1), (d3, h3): Stable
4. (d1, h2), (d2, h3), (d3, h1): Unstable. h1 prefers d2 to d3 and d2 prefers h1 to h3.
5. (d1, h3), (d2, h2), (d3, h1): Unstable. h1 prefers d2 to d3 and d2 prefers h1 to h2.
6. (d1, h3), (d2, h1), (d3, h2): Stable

stable matchings: 2

b) Interns propose to hospitals:

d1 proposes to h1, h1 accepts.	(d1, h1)
d2 proposes to h1. h1 prefers d2 to d1, h1 accepts, cancelling on d1.	(d2, h1)
d3 proposes to h3, h3 accepts.	(d2, h1), (d3, h3)
d1 proposes to h2. h2 accepts.	(d1, h2), (d2, h1), (d3, h3)

c) Hospitals propose to interns:

h1 proposes to d2, d2 accepts.	(h1, d2)
h2 proposes to d2. d2 prefers h1 to h2, d2 rejects.	(h1, d2)
h2 proposes to d3, d3 accepts.	(h1, d2), (h2, d3)
h3 proposes to d1, d1 accepts	(h1, d2), (h2, d3), (h3, d1)

A3Q2

a)

Preprocess: Sort L, P for increasing order of locations from start of highway.
Base case: i = j. S[i][j] contains element L[i]. T[i][j] contains profit P[i].
Subproblem: S[i][j] contains consecutive elements starting at L[i], the following at least D apart from the last one. P[i][j] contains profit from these elements.
Pseudocode analysis: We have two nested for loops, one from i = 1 to n. The nested one being from j = i to n. We do this twice, first to calculate the sub-problems. Next to filter through our results to only include results that have k elements and return the locations with the maximum profit. Therefore this is $O(n^2)$.

```
L: n restaurant locations
P: Profits associated with n locations
D: Each location in solution at least D distance apart
k: # restaurants in solution
return the locations from which maximum profit is possible,
at least D apart
FastFoodRestaurant(L, P, D, k, n):
  # build tuples, sort by first element of tuple,
  # assign back to arrays
  L, P = sorted([(L[i], P[i]) for i in range(1, len(L)])
  S[n][n] # array of elements included in L[i..j]
  T[n][n] # profit from S[i][j] elements
  x, y, maxProfit = -1, -1, 0

  # Within L[i..j], fill S[i][j] with locations at least D
apart starting with L[i]. P[i][j] holds the profit from such
locations
  for i = 1 to n:
    S[i] = [] * n
    T[i] = [0] * n
    for j = i to n:
      if i = j:
        S[i][j] = [j]
        T[i][j] = P[j]
      else:
        S[i][j] = S[j-1]
        T[i][j] = T[j-1]
```

```
        if ((L[j] - L[S[i][j][-1]]) >= D):
          S[i][j] += [i]
          T[i][j] += P[i]

 for i = 1 to n:
   for j = i to n:
     if len(S[i][j]) = k and T[i][j] > maxProfit:
        x, y = i, j
        maxProfit = T[i][j]

 if x = -1 or y = -1:
   return -1
 else:
   return S[x][y]
```

b)

S[i..j] for elements included
T[i..j] for profit from given elements
Excl (for excluded) indicates # elements < k
Bold cell indicates optimal solution.

i \ j	1	2	3	4	5	6	7	8	9
1	[1] 4 Excl	[1, ~~2~~] 10 Excl	[1, 3] 13 Excl	[1, 3, ~~4~~] 13 Excl	[1, 3, 5] 25 Excl	[1, 3, 5, ~~6~~] 25 Excl	[1, 3, 5, 7] 33	[1, 3, 5, 7, ~~8~~] 33	[1, 3, 5, 7, 9] 40 Excl
2		[2] 10 Excl	[2, 3] 19 Excl	[2, 3, ~~4~~] 19 Excl	[2, 3, 5] 31 Excl	[2, 3, 5, ~~6~~] 31 Excl	**[2, 3, 5, 7] 39**	[2, 3, 5, 7, ~~8~~] 39	[2, 3, 5, 7, 9] 46
3			[3] 9 Excl	[3, ~~4~~] 9 Excl	[3, 5] 21 Excl	[3, 5, ~~6~~] 21 Excl	[3, 5, 7] 29 Excl	[3, 5, 7, ~~8~~] 29 Excl	[3, 5, 7, 9] 36
4				[4] 6 Excl	[4, ~~5~~] 6 Excl	[4, 6] 11 Excl	[4, 6, 7] 19 Excl	[4, 6, 7, ~~8~~] 19 Excl	[4, 6, 7, 9] 26

10	11
[1, 3, 5, 7, 9, ~~10~~] 40 Excl	[1, 3, 5, 7, 9, 11] 42 Excl
[2, 3, 5, 7, 9, ~~10~~] 46	[2, 3, 5, 7, 9, 11] 48
[3, 5, 7, 9, ~~10~~] 36	[3, 5, 7, 9, 11] 38 Excl
[4, 6, 7, 9, ~~10~~] 26	[4, 6, 7, 9, 11] 28 Excl

5					[5] 12 Excl	[5, ~~6~~] 12 Excl	[5, 7] 20 Excl	[5, 7, ~~8~~] 20 Excl	[5, 7, 9] 25 Excl
6						[6] 5 Excl	[6, 7] 13 Excl	[6, 7, ~~8~~] 13 Excl	[6, 7, 9] 20 Excl
7							[7] 8 Excl	[7, ~~8~~] 8 Excl	[7, 9] 15 Excl
8								[8] 10 Excl	[8, ~~9~~] 10 Excl
9									[9] 7 Excl
10									
11									

[5, 7, 9, ~~10~~] 25 Excl	[5, 7, 9, 11] 29
[6, 7, 9, ~~10~~] 20 Excl	[6, 7, 9, 11] 22
[7, 9, ~~10~~] 15 Excl	[7, 9, 11] 17
[8, ~~10~~] 10 Excl	[8, 11] 12 Excl
[9, ~~10~~] 7 Excl	[9, 11] 9 Excl
[10] 11 Excl	[10, ~~11~~] 11 Excl
	[11] 2 Excl

A3Q3

a) Let our problem instance, I, be:

l	1	2	3	4
a_l	1	4	9	10
b_l	-3	2	6	15

Optimal solution: $max\ i = 1\ to\ n\ \left|a_i - b_j\right| = 5\ for\ i = j = 4.$

i	1	2	3	4
Strategy X / j	2	3	1	4
Strategy X / $\lvert a_i - b_j \rvert$	1	2	**12**	5
Strategy Y / j	1	2	3	4
Strategy Y / $\lvert a_i - b_j \rvert$	4	2	3	**5**
Strategy Z / j	4	3	2	1
Strategy Z / $\lvert a_i - b_j \rvert$	**14**	2	7	13

Therefore strategies X and Z are incorrect, counter-example given above.

Proof for Strategy Y by contradiction. Let's say we have a pair $(a_i,\ b_j)$ such that $|a_i - b_j| < |a_i - b_i|$ where $i \neq j$.

Case 1. $i > j$. We would also need a matching for at least one pair from $a_{i..n}$ and $b_{0..j}$. As $b_{0..j} < b_j$ and $a_{i..n} > a_i$, difference of the elements would be more than that of (a_i, b_i).
Case 2. $i < j$. Without loss of generality, we can show the same result from case 1.

From the two cases, we have a contradiction. Therefore Strategy Y is most optimal.

b)

Base case: i = j. C[i][i] contains abs(A[i] - B[i]). Runs in O(n).
Pseudocode analysis: We have two nested for loops, one from i = 1 to n. The nested one being from j = i+1 to n. Runs in $O(n^2)$.

```
A, B: array of n elements where element i < element j for i
< j.
Returns an array of tuples for optimal pairing
MinMaxPairwise(A, B, n):
  C[n][n] # memory
  for i = 1 to n:
    C[i][i] = abs(A[i] - B[i])

  for i = 2 to n:
     for j = i+1 to n:
       C[i][j] = min(C[i-1][j-1] + abs(A[i] - B[j]), C[i][j-1])

  return [(C[i], C[i]) for i = 1 to n]
```

A3Q4

a)

Base case: j = k, which is a palindrome of 1 character.

```
findLongestPalindromeSubSequence(text, len):
  memory[len][len] = {[]} # store indices of palindrome
characters
  result = ""

  // Base case
  for i = 0 to len:
    memory[i][i] = 1

  // Table
  for i = 2 to len+1:
    for j = 0 to (len - i + 1):
      k = j + i - 1;
      if (i = 2 and text[j] = [k]):
        memory[j][k] = [j, k];
      else if (text[j] = text[k]:
        memory[j][k] = [j] + memory[j + 1][k - 1] + [j];
      else:
        if (memory[j][k - 1] > memory[j + 1][k]):
          memory[j][k] = memory[j][k - 1];
        else:
          memory[j][k] = memory[j + 1][k];

  for i = 0 to memory[0][len-1]:
    result += text[i]

  return result
```