Question 1.21: Find the pair of nearest points in squared Euclidean distance

Give a list of 200 2-dimensional points

In [1]:
```
points = [[1.6917766692855842, -5.755629277630385], [7.675827069862372, 7.301699114949773], [-3.2909129520196334, 9.6
```

In [2]:
```
min_dist = float('inf')
min_i, min_j = 0, 0
n = len(points)
for i in range(n):
    for j in range(i+1, n):
        d = (points[i][0] - points[j][0]) ** 2 + (points[i][1] - points[j][1]) ** 2
        if d < min_dist:
            min_dist, min_i, min_j = d, i, j
print('minimum squared Euclidean distance:', min_dist, 'between point', min_i, 'and point', min_j)
```

```
minimum squared Euclidean distance: 0.008302746110664843 between point 66 and point 113
```

Question 1.22: Find longest palindrome

if two palindromes are of same length, return the one appears first

optional part: if two palindromes are of same length, return the one appears first in alphabetical order

In [3]:
```
s = 'abbabcdefghijihgfedcbaaabbbbaaaabbaaaabbbbaaabc'
```

In [4]:
```
n = len(s)
palindrome = {}
for i in range(n):
    j = 1
    while j <= i and j < n - i:                 # consider strings center at [j], from index i-j to index i+j
        if s[i-j] != s[i+j]:
            break                               # break when the characters do not match
        j += 1
    if 2 * j - 1 not in palindrome:             # optional part add 'or palindrome[2*j-1] > s[i-j+1:i+j]'
         palindrome[2*j-1] = s[i-j+1:i+j]
for i in range(n-1):
    if s[i] == s[i+1]:                          # consider strings center at [j] and [j+1]
        j = 1
        while j <= i and j < n - i - 1:         # from index i-j to index i+1+j
            if s[i-j] != s[i+1+j]:
                break
            j += 1
        if 2 * j not in palindrome:             # optional part add 'or palindrome[2*j] > s[i-j+1:i+1+j]'
            palindrome[2*j] = s[i-j+1:i+1+j]
print(palindrome)
```

```
{1: 'a', 3: 'bab', 19: 'abcdefghijihgfedcba', 5: 'baaab', 4: 'abba', 2: 'aa', 10: 'aaabbbbaaa', 8: 'bbaaaabb', 28: 'c
baaabbbbaaaabbaaaabbbbaaabc'}
```

Question 1.23 : Largest number in a string matching -- in a single scan

e.g. string = 'abc123def456.789' => return 789

In [5]:
```
s = 'abc123def456.789.012345.6789-12345678.9012311111.456$#@'
#s = 'abc123def456.789'
#s = 'abc123def789...456.'
s = 'abc123.132abc......a13.'
#s = 'a1b2c3d1'
s = '.....12.12.ab.11..23.'
```

In [6]:
```
n = len(s)
max_float = 0.0
first, second = '0', '0'
for i in range(n):
    if s[i].isdigit():                          # orange case: waiting for more digits to come
        first += s[i]
        second += s[i]
    elif s[i] == '.':                           # blue case:
        max_float = max(max_float, float(second)) #     -- 1. settle the second buffer which may contain a decimal wi
        second = first + '.'                    #     -- 2. make a new standby buffer for a decimal with '.'
        first = '0'                             #     -- 3. make a new buffer for an integer without '.'
    elif len(first) > 1 or len(second) > 1:     # green case: settle the digits in the buffers
        max_float = max(max_float, max(float(first), float(second)))
        first, second = '0', '0'
    else:                                       # red case: do nothing
        pass
max_float = max(max_float, max(float(first), float(second)))
print(max_float)
```

```
23.0
```

Question 1.24: Write the naive primality test algorithm

In [7]:
```
prime = []
for i in range(2, 200):
    is_prime = True              # first assume i is a prime
    for f in range(2, i):        # for factors from 2 to i-1
        if i % f == 0:
            is_prime = False     # set is_prime to False if divisible
    if is_prime:
        prime.append(i)
print(prime)
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109,
113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199]
```

In [8]:
```
prime = [2]
for i in range(3, 200, 2):       # test for odd numbers only
    is_prime = True              # first assume i is a prime
    for f in prime:              # test prime factors only
        if i % f == 0:
            is_prime = False     # set is_prime to False if divisible
    if is_prime:
        prime.append(i)
print(prime)
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109,
113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199]
```

In [9]:
```
prime = [2]
for i in range(3, 200, 2):       # test for odd numbers only
    is_prime = True              # first assume i is a prime
    for f in prime:              # test prime factors only
        if f * f > i:            # factors are in pairs except perfect squares, test stops when f * f > i
            break
        if i % f == 0:
            is_prime = False     # set is_prime to False if divisible
    if is_prime:
        prime.append(i)
print(prime)
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109,
113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199]
```

Question 1.25: Sudoku checker -- return cells which you are certain about its value

```
    7       1
  5 9       2 3
8 2 3       4 5 6
5 1           7 8
9     1   3     4
    4 2   8 9
  7 5 3   6 8 9
  8 6       7 1
  9           4
```

In [10]:
```
sudoku1 = ['  7   1  ', ' 59   23 ', '823   456', '51     78', '9  1 3  4', '  42 89  ', ' 753 689 ', ' 86   71 ', '
sudoku2 = ['         ', ' 528 614 ', ' 98   27 ', '  12 39  ', '735   682', ' 695 741 ', ' 1 3 8 9 ', '42 7 1 65', '
sudoku3 = ['    76 3 ', ' 13 8  92', ' 4 2 31  ', '  86  5 9', '26     78', '4 9  82  ', '  17 5 2 ', '95  3 41 ', '
```

In [11]:

```
value = [[-1] * 9 for _ in range(9)]
possible = [[[True] * 10 for _ in range(9)] for _ in range(9)]

i = 0
for s in sudoku1:
    if len(s) != 9:
        print('length should be 9:', s)
    for j in range(9):
        if s[j] != ' ':
            v = int(s[j])
            value[i][j] = v
            for vv in range(1, 10):
                if vv != v:
                    possible[i][j][vv] = False # mark this cell false on all other values
            for jj in range(9):
                if jj != j:
                    possible[i][jj][v] = False # mark other cells in the same row false on the same value
            for ii in range(9):
                if ii != i:
                    possible[ii][j][v] = False # mark other cells in the same column false on the same value
            gi, gj = i // 3 * 3, j // 3 * 3
            for ii in range(3):
                for jj in range(3):
                    if gi + ii != i or gj + jj != j:
                        possible[gi + ii][gj + jj][v] = False # mark other cells in the same 3x3 grid false on the sa
    i += 1
for i in range(9):
    for j in range(9):
        if value[i][j] == -1:
            print('(%d, %d): %s' % (i + 1, j + 1, ' '.join([str(v) for v in range(1, 10) if possible[i][j][v]])))
```

```
(1, 1): 4 6
(1, 2): 4 6
(1, 4): 4 5 6 8 9
(1, 5): 2 3 4 5 6 8 9
(1, 6): 2 4 5 9
(1, 8): 8
(1, 9): 9
(2, 1): 1 4 6
(2, 4): 4 6 7 8
(2, 5): 1 4 6 7 8
(2, 6): 1 4 7
(2, 9): 7
(3, 4): 7 9
(3, 5): 1 7 9
(3, 6): 1 7 9
(4, 3): 2
(4, 4): 4 6 9
(4, 5): 4 6 9
(4, 6): 4 9
(4, 7): 3 6
(5, 2): 6
(5, 3): 2 8
(5, 5): 5 6 7
(5, 7): 5 6
(5, 8): 2 6
(6, 1): 3 6 7
(6, 2): 3 6
(6, 5): 5 6 7
(6, 8): 6
(6, 9): 1 3 5
(7, 1): 1 2 4
(7, 5): 1 2 4
(7, 9): 2
(8, 1): 2 3 4
(8, 4): 4 5 9
(8, 5): 2 4 5 9
(8, 6): 2 4 5 9
(8, 9): 2 3 5
(9, 1): 1 2 3
(9, 3): 1 2
(9, 4): 5 7 8
(9, 5): 1 2 5 7 8
(9, 6): 1 2 5 7
(9, 7): 3 5 6
(9, 9): 2 3 5
```