1) A = {3, 44, 38, 5, 47, 15}

Selection Sort :

Step 1: A = {3, 44, 38, 5, 47, 15}

2: A = {3, 5, 38, 44, 47, 15}

3: A = {3, 5, 15, 44, 47, 38}

4: A = {3, 5, 15, 38, 47, 44}

5: A = {3, 5, 15, 38, 44, 47}

Bubble Sort

Step 1: A = {3, 44, 38, 5, 47, 15}

2: A = {3, 44, 38, 5, 47, 15} // swap

3: A = {3, 38, 44, 5, 47, 15} // swap

4: A = {3, 38, 5, 44, 47, 15} // no swap

5: A = {3, 38, 5, 44, 47, 15} // swap

6: A = {3, 38, 5, 44, 15, 47} // back head

7: A = {3, 38, 5, 44, 15, 47} // no swap

8: A = {3, 38, 5, 44, 15, 47} // swap

9: A = {3, 5, 38, 44, 15, 47} // no swap

10: A = {3, 5, 38, 44, 15, 47} // swap

11: A = {3, 5, 38, 15, 44, 47} // no swap

12: A = {3, 5, 38, 15, 44, 47} // back head

13: A = {3, 5, 38, 15, 44, 47}

14: A = {3, 5, 38, 15, 44, 47} // when index is 2, swap

16: A = {3, 5, 15, 38, 44, 47}

20: A = {3, 5, 15, 38, 44 47} // END

1) Insertion Sort

Setp 1: A = {3, 44, 38, 5, 47, 15}

2: A = {3, 44, 38, 5, 47, 15}

3: A = {3, 44, 38, 5, 47, 15}

4: A = {3, 38, 44, 5, 47, 15}

5: A = {3, 38, 44, 5, 47, 15}

6: A = {3, 38, 5, 44, 47, 15}

7: A = {3, 5, 38, 44, 47, 15}

8: A = {3, 5, 38, 44, 47, 15}

9: A = {3, 5, 38, 44, 15, 47}

10: A = {3, 5, 38, 15, 44, 47}

11: A = {3, 5, 15, 38, 44 47} // END

1) Quick Sort

step 1: {3, 44, 38, 5, 47 15}

step 2: {3, 44, 38, 5, 47 15}

step 3: {3, 15, 38, 5, 47, 44}

step 4: {3, 15, 38, 5, 47, 44}

step 5: {3, 15, 38, 5, 47, 44}

step 6: {3, 5, 38, 15, 47, 44}

step 7: {3 5 15 38, 47, 44}

step 8: {3, 5, 15, 38, 47, 44}

step 9: {3, 5, 15, 38, 44, 47}

2) a) Selection Sort is not stable. Because, during sort, to be swopped some value.

b) Buble Sort is stable. Because, during sort, to be unchange some value.

c) It's possible. Because selection sort travels all array, and to be swopped find the swallest value. This case is valid for linked list. Therefore we make selection sort with linked List

d) if linked list have back pointer as next pointer, it is possible Insertion Sort. Some time, efficiency is $O(n^2)$. But if it don't has back pointer, it is not possible.

3.) Algorithm (Alternating disk)

```
#Function
def alter_sort(A, n)
    moves=0
    for i in xrange(n)
        minn=i
        for j in xrange(i+1,n):
            if A[j] < A[minn]:
                minn=j
        moves += minn-i
        A[i], A[minn] = A[minn], A[i]
```

number of moves is 6 steps

4-) length of text = n # psuedocode
length of pattern = m

```
for i = 0   i < n-m   ++i
    for j = 0  j < m  ++j
        if txt.char(i+j) != pattern.char(j)
            break;
    if j == m return i;
return n;
```

//end

at this case, Exactly number of comparisons is (n-m).m

5) a) # Brute-force Algorithm

```
pattern = [ C A B A A x B Y A ]
countSubstr = 0
IndexA = 0
Index B = 0
flag = False
for i in len(pattern)
    if flag == False :
        if pattern[i] == 'A':
            indexA = i
            flag = True
    if flag == True :
        for j in leng(pattern) - indexA
            if pattern[j+IndexA] == 'B' :
                ++count
                flag = False

return countSubstr
```

6) İlk olarak iki çocuk bota alınıp diğer tarafa geçer. Onlardan biri tekrar geri döner. Bir asker ile diğer tarafa tekrar döner ve bu yolculuk 1'er 1'er devam eder. Toplamda 25 kez yolculuk prosedürü 25 kez tekrar edilir. Bu problemin çözümü 100 yolculuktan sonra çözülür. Bunu genelleştirirsek n askere, 4n yolculuk (sefer) yapılmalıdır.

7.) # Celebrity problem Algorithm

* Prepare test

```
N = 10
R = range(N)
rand_celeb = random.randint(0, N-1)

m = {}
for i in xrange(N):
    if i != rand_celeb:
        m[i] = [random.randint(0, N-1) for j in xrange(10)] +
                                                [rand_celeb]
    else:
        M[i] = []
```

End of prepare test, now ready test and Begin Algorithm

```
def knows(i, j): # i knows j?
    return j in M[i]

def find_celeb(n):
    celeb = n
    for i in xrange(1,n):          # n^2
        if not knows(i,celeb):
            celeb=i

    for i in xrange(n):            # n^2
        if i != celeb and knows(celeb,i):
            return None

    return celeb
```

END

$T(n) = n^2 + n^2 \Rightarrow 2n^2$

worst case $T(n) \in O(n^2)$

8) a) # Flipping Pancakes Algorithm

```
# Sorts Pancakes
def sortPancakes(stack)
    sorted_index = 6  # size of Array in pdf of hw2
    for i in reversed(range(len(stack))):
        stack = flip(stack, findLargestPancake(stack, i))
        stack = flip(stack, i)

    return stack


# All of the pancakes are sorted after index
# Returns the index of largest unsorted pancake
def findLargestPancake(stack, index):
    largest_pancake = stack[index]
    largest_index = index

    for i in range(index):
        if stack[i] > largest_pancake
            largest_pancake = stack[i]
            largest_index = i

    return largest_index

# Slide slice under pancake at index and flip to top
def flip(stack, index):
    newStack = stack[:(index+1)]
    newStack.reverse()
    newStack += stack[(index+1):]
    return newstack

stack = [1, 2, 10, 7 8, 3]
```

8) b) stack = [1, 2, 10, 7, 8, 3] → find largest, put head and reverse

step 2: [10, 2, 1, 7, 8, 3]

step 3: [3, 8, 7, 1, 2, 10]

step 4: [8, 3 7, 1, 2, 10]

step 5: [2, 1, 7, 3, 8, 10]

step 6: [7, 1, 2, 3, 8, 10]

step 7: [3, 2, 1, 7, 8, 10]

step 8: [3, 2, 1, 7, 8, 10]

step 9: [1, 2, 3, 7, 8, 10]

step 10: [2, 1, 3, 7, 8, 10]

step 11: [1, 2, 3, 7, 8, 10]

step 12: [1, 2, 3, 7, 8, 10]

c) $\left[\sum_{i=0}^{n=\text{len(stock)}} \left(\sum_{j=0}^{\text{index}=i} 1\right) + n\right] = c.n^2$

worst case = $O(n^2)$

best case = $O(n^2)$

a) # Algorithm

```
def  pair_of_integer( num)

  if  num % 2 ==0 :
        n1 = num / 2
        n = n1 * n1
        return n .

  else :
        n1 = (num-1) / 2
        n = n1 . (n1 +1)
        return n
```