Ahmed Abd-Elsalam Muhammed Afify

Algorithms HW_3

1) Knapsak Problem (Divide_Ahd_Conqure)

In [1]:
```
def karatsuba(x,y):

    if len(str(x)) == 1 or len(str(y)) == 1:
        return x*y

    n = max(len(str(x)), len(str(y)))
    m = n//2

    a = x//(10**m)
    b = x%(10**m)
    c = y//(10**m)
    d = y%(10**m)

    ac = karatsuba(a,c)
    bd = karatsuba(b,d)
    bc_ad = karatsuba(b,c) + karatsuba(a,d)
    result = ac*(10**(2*m)) + bc_ad*(10**m) + bd
    return result
```

In [2]:
```
karatsuba(1234, 56789)
```

Out[2]:
```
70077626
```

---

2) fractional Knapsak

In [3]:

```
def fractional_knabsak_dic(dic, w=0):
    # ----------------------------------------------
    new_dic = {}
    for element in dic:
        new_dic[element] = element/dic[element]
    # ----------------------------------------------
    total_values = 0
    fraction = 0
  # ----------------------------------------------
    for i in range(len(dic)):
        max_key = max(new_dic, key=new_dic.get)
        # ----------------------------------------------
        if w - dic[max_key]>0:
            w -= dic[max_key]
            total_values += max_key
        else:
            fraction = w / dic[max_key]
            total_values += max_key*fraction
        # ----------------------------------------------
        del new_dic[max_key]
    # ----------------------------------------------
    return total_values
```

In [4]:

```
dic = {60: 10, 100: 20, 120: 30}
frac_knap_dic = fractional_knabsak_dic(dic, 50)
print(frac_knap_dic)
```

```
240.0
```

Fractional Knapsak with input as two lists

In [5]:

```
def fractional_knabsak_lst(val, wt, w=0):
    # ----------------------------------------------
    val_by_wt = []
    for i in range(len(val)):
        val_by_wt.append(val[i]/wt[i])
    # ----------------------------------------------
    total_values = 0
    fraction = 0
  # ----------------------------------------------
    for i in range(len(val)):
        max_i = max(val_by_wt)
        index_i = val_by_wt.index(max_i)
        # ----------------------------------------------
        if w - wt[index_i]>=0:
            w -= wt[index_i]
            total_values += val[index_i]
        else:
            fraction = w / wt[index_i]
            total_values += val[index_i]*fraction
        # ----------------------------------------------
        val_by_wt[index_i] = -1
    # ----------------------------------------------
    return total_values
```

In [6]:

```
frac_knap_lst = fractional_knabsak_lst([60, 100, 120],[10, 20, 30], 50)
print(frac_knap_lst)
```

```
240.0
```

Dynamic Programming:

1) Coin Change Problem:

In [7]:

```
def coin_change(coins, money):
    import numpy as np
    min_coins = np.ones(money+1)*(money+1)
    min_coins[0] = 0
    coins = sorted(coins)
    for i in range(1,len(min_coins)):
        for j in coins:
            if (i - j) >= 0:
                min_coins[i] = min(min_coins[i-j]+1, min_coins[i])
            else:
                continue
        if min_coins[i] in [0,money+1]:
            min_coins[i] = -1
    return min_coins[money]
```

In [8]:

```
trial1 = coin_change([1,3,5,6,9],90)
trial2 = coin_change([1,2,3],10)
trial3 = coin_change([2],5)
print(trial1)
print(trial2)
print(trial3)
```

```
10.0
4.0
-1.0
```

2) Edit Distance (levenshtein distsnce):

In [9]:

```
def edit_distance(x, y):
    import numpy as np
    x_dim = len(x)+1   # No. of columns
    y_dim = len(y)+1   # No. of rows
    min_distance = np.zeros((y_dim, x_dim))

    for i in range(1, len(x)+1):
        min_distance[0, i] = i
    for i in range(1, len(y)+1):
        min_distance[i, 0] = i

    for i in range(1, len(x)+1):
        for j in range(1, len(y)+1):
            if x[i-1] == y[j-1]:
                min_distance[j, i] = min_distance[j-1, i-1]
            else:
                min_distance[j, i] = min(min_distance[j-1, i-1], min_distance[j,

    #print(min_distance)
    return min_distance[y_dim-1, x_dim-1]
```

In [10]:

```
edit2 = edit_distance('short', 'ports')
print(edit2)
```

```
3.0
```

3) Longest Common Subsequence:

In [11]:

```
def longest_com_subs(x, y):
    import numpy as np
    x_dim = len(x)+1    # No. of columns
    y_dim = len(y)+1    # No. of rows
    max_common = np.zeros((y_dim, x_dim))

    for i in range(1, len(x)+1):
        for j in range(1, len(y)+1):
            if x[i-1] == y[j-1]:
                max_common[j, i] = max_common[j-1, i-1]+1
            else:
                max_common[j, i] = max(max_common[j, i-1], max_common[j-1, i])

    #print(max_common)
    return max_common[y_dim-1, x_dim-1]
```

In [12]:

```
test = longest_com_subs('AGGTAB', 'GXTXAYB')
print(test)
```

```
4.0
```

In [13]:

```
t = edit_distance('ab','ab')
print(t)
```

```
0.0
```

4) 0_1 Knapsak Problem:

In [14]:

```
def zero_one_knapsak(values, weights, W):
    import numpy as np
    n = len(values)
    optimal_weight = np.zeros((n, W+1))
    for i in range(n):
        for j in range(1,W+1):
            if j >= weights[i]:
                 optimal_weight[i, j] = max(optimal_weight[i-1, j], optimal_weigh
            else:
                optimal_weight[i, j] = optimal_weight[i-1, j]
    return optimal_weight[n-1,W]
```

In [15]:

```
test_kanpsak = zero_one_knapsak([60,100,120], [10,20,30], 50)
print(test_kanpsak)
```

```
220.0
```

In [ ]: