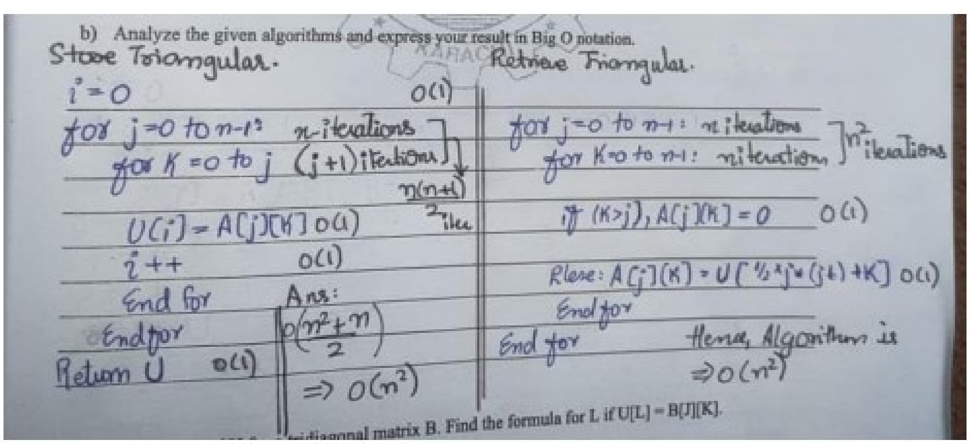
Exercise (a)

```python
## A5.1 - StoreTriangular(A)
def storeTriangular(A):
    n = len(A)
    U_size = int(0.5*n*(n+1))
    U = [0 for i in range(U_size)]
    i = 0
    for j in range(n):
        for k in range(j+1):
            U[i] = A[j][k]
            i+=1
    return U

## A5.2 - RetrieveTriangular(U)
def retrieveTriangular(U):
    Usize = len(U)
    n = int((-1+(1+8*Usize)**0.5)//2)

    A = [[0 for col in range(n)] for row in range(n)]

    for j in range(n):
        for k in range(n):
            if (k>j):
                A[j][k] = 0
            else:
                L = int((0.5*j*(j+1))+k)
                A[j][k] = U[L]
    return A

A = [
 [4,0,0,0],
 [3,-5,0,0],
 [1,6,2,0],
 [8,0,5,9]]

U = storeTriangular(A)
print("Stored U: ", U)
retrieved = retrieveTriangular(U)
print("Retrieved Matrix: \n", retrieved)
```

```
Stored U:  [4, 3, -5, 1, 6, 2, 8, 0, 5, 9]
Retrieved Matrix:
 [[4, 0, 0, 0], [3, -5, 0, 0], [1, 6, 2, 0], [8, 0, 5, 9]]
```

Exercise (b)



b) Analyze the given algorithms and express your result in Big O notation.

Store Triangular.

i = 0 — O(1)

for j = 0 to n-1: n-iterations

for K = 0 to j: (j+1) iterations — $\frac{n(n+1)}{2}$ iter

U[i] = A[j][K] O(1)

i++ O(1)

End for

End for

Return U O(1)

Ans: $O\left(\frac{n^2+n}{2}\right)$

$\Rightarrow O(n^2)$

Retrieve Triangular.

for j = 0 to n-1: n iterations

for K = 0 to n-1: n iterations — n^2 iterations

if (K>j), A[j][K] = 0 O(1)

Else: A[j][K] = U[½ * j * (j+1) + K] O(1)

End for

End for

Hence, Algorithm is $\Rightarrow O(n^2)$

...diagonal matrix B. Find the formula for L if U[L] = B[J][K].

Exercise (d) & (e)

```python
def store_tridiagonal(B):
    n = len(B)
    Usize = 3*n-2
    U = [0 for i in range(Usize)]
    i = 0
    for j in range(n):
        for k in range(j-1,j+2):
            if 0 <= k < n:
                U[i] = B[j][k]
                i += 1
    return U

def retrieve_tridiagonal(U):
    Usize = len(U)
    n = (Usize+2)//3
    B = [[0 for i in range(n)] for j in range(n)]

    for j in range(n):
        for k in range(n):
            if j+1 >= k >= j-1:
                x = 2*j + k
                B[j][k] = U[x]
            else:
                B[j][k] = 0
    return B


B = [
    [5, -7, 0, 0],
    [1, 4, 3, 0],
    [0, 9, -3, 6],
    [0, 0, 2, 4]
]
result2 = store_tridiagonal(B)
print("U: ", result2)
retrieved2 = retrieve_tridiagonal(result2)
print("Retrieved Matrix:")
for row in retrieved2:
    print(row)
```

```
U:  [5, -7, 1, 4, 3, 9, -3, 6, 2, 4]
Retrieved Matrix:
[5, -7, 0, 0]
[1, 4, 3, 0]
[0, 9, -3, 6]
[0, 0, 2, 4]
```

Exercise (f)

```
import numpy as np
from scipy.sparse import csr_matrix

dense = np.array([
    [1, 0, 0, 0, 2, 0],
    [0, 0, 3, 0, 0, 4],
    [5, 0, 0, 6, 0, 0]
])

print("Dense Matrix:\n", dense)

#Converting to CSR format
sparse_csr = csr_matrix(dense)
print("\nCSR Representation:\n", sparse_csr)

#Converting back to dense
dense_back = sparse_csr.toarray()
print("\nBack to Dense:\n", dense_back)
```

```
Dense Matrix:
 [[1 0 0 0 2 0]
 [0 0 3 0 0 4]
 [5 0 0 6 0 0]]

CSR Representation:
 <Compressed Sparse Row sparse matrix of dtype 'int64'
	with 6 stored elements and shape (3, 6)>
  Coords	Values
  (0, 0)	1
  (0, 4)	2
  (1, 2)	3
  (1, 5)	4
  (2, 0)	5
  (2, 3)	6

Back to Dense:
 [[1 0 0 0 2 0]
 [0 0 3 0 0 4]
 [5 0 0 6 0 0]]
```