Task

a. Write all required algorithms needed to sort a sequence of numbers using Heapsort Algorithms.

```
get_parent(child):
   parent = floor(child/2)

heap_sort(arr):
    for i = 0 downTo arr_size - 1 :
        child = i

        temp_arr.append(arr[i])
        while child>0 :
           parent = get_parent(child)
           if n_arr[child] > n_arr[parent]:
                swap child and parent in n_arr
                child = parent
      return n_arr
```

b. Analyze in detail your written algorithms in Part (a).

Code Explanation :

1. `get_parent(child)` function:

- This function calculates the index of the parent node in a binary heap given the index of the child node.
- This line `parent - 1` handles the zero-indexed array in python ,but the general rule of binary trees to find the child's parent is → floor(child/2).

2. `heap_sort(arr)` function:

- This function attempts to perform heap sort on the given array.
- **Build Max-Heap:**
    - The code iterates through the array from index 0 to n-1.
    - In each iteration:
        - `n_arr.append(arr[i])`: Appends the current element to the `n_arr` list.
        - `while child > 0`:
            - Calculates the parent index using `get_parent(child)`.
            - If the current element (`n_arr[child]`) is greater than its parent (`n_arr[parent]`), it swaps the elements.
            - Updates the `child` index to the parent index for the next iteration.

Code summary :

The code creates a new array named `n_arr` to construct the max-heap. It then appends the values of the original array to `n_arr` one by one. Each time a value is appended to `n_arr`, the code compares it to its current parent in the heap. If this value is greater than its parent, they are swapped. This process continues iteratively until the end of the original array is reached.

Time Complexity :

```
get_parent(child):
   parent = floor(child/2)

heap_sort(arr):
    for i = 0 downTo arr_size - 1 : # this loop will all over ar
        child = i

        temp_arr.append(arr[i])
        while child>0 :  # this loop will take O(log(arr_size))
           parent = get_parent(child)
           if n_arr[child] > n_arr[parent]:
                swap child and parent in n_arr
                child = parent
      return n_arr
```

total time complexity = n x logn

n is the size of the array sorted

c. Implement your written algorithms in Part (a).

```python
import math

def get_parent(child):
    """
    Calculates the index of the parent node in a binary heap.

    Args:
        child: Index of the child node.

    Returns:
        Index of the parent node.
    """
    parent = math.floor(child / 2)
    if parent == 0:
        return 0  # Parent of root node is itself
    else:
        return parent - 1
```

```
def heap_sort(arr):
    """
    Performs heap sort on the given array.

    Args
    """
    n = len(arr)
    n_arr = []
    # Build a max-heap
    for i in range(0, n-1):
        child = i
        n_arr.append(arr[i])
        while child > 0:
            parent = get_parent(child)
            if n_arr[child] > n_arr[parent]:
                n_arr[child], n_arr[parent] = n_arr[parent], n_a
                child = parent


    return n_arr





arr = [0, 1, 2, 3, 4]
arr = heap_sort(arr)

print(arr)  # Output: [0, 1, 2, 3, 4]
```