PROJECT (1)

a) required algorithm needed to sort a sequence of numbers using Heapsort Algorithms.

1.Heapsort algorithm

```
HEAPSORT(array)
BUILD_MAX_HEAP(array)
for end from length(array) - 1 to 1
SWAP(array[0], array[end])
HEAPIFY(array, 0, end)
```

2.Build Max Heap Algorithm

```
BUILD_MAX_HEAP(array)
startIndex = (length(array) / 2) - 1
for i from startIndex to 0
HEAPIFY(array, i, length(array))
```

3.Heapify Algorithm

```
HEAPIFY(array, rootIndex, heapSize)
  largest = rootIndex
  leftChild = 2 * rootIndex + 1
  rightChild = 2 * rootIndex + 2
if leftChild < heapSize and array[leftChild] > array[largest]
    largest = leftChild
if rightChild < heapSize and array[rightChild] > array[largest]
    largest = rightChild
if largest != rootIndex
    SWAP(array, rootIndex, largest)
    HEAPIFY(array, largest, heapSize)
```

4.Swap Elements

```
SWAP(array, index1, index2)
temp = array[index1]
array[index1] = array[index2]
array[index2] = temp
```

5.Get User Input

```
GET_USER_INPUT()
print "Enter a sequence of numbers (comma-separated):"
input = read input
return SPLIT input by ',' and PARSE as integers
```

6.Main Method

```
MAIN()
  numbers = GET_USER_INPUT()
  print "Original array: " + JOIN(numbers, ", ")
  HEAPSORT(numbers)
  print "Sorted array: " + JOIN(numbers, ", ")
```

b) Analyze written algorithms in Part (a).

Step 1: Build Max Heap

Time Complexity: O(n).

Step 2: Extract and Sort

Time Complexity: O(n log n)

Step 3: Heapify Subtree

Time Complexity: O(log n)

-> Total Time Complexity: O(n log n)