```python
# Sorting Algorthims
A1 = [7, 8, 5, 4, 9, 2]


# Insertion Sort
def insertion_sort(A):
    for i in range(1, len(A)):
        j = i - 1
        while A[j] > A[j + 1] and j >= 0:
            A[j], A[j + 1] = A[j + 1], A[j]
            j -= 1


# Selection Sort
def selection_sort(A):
    for i in range(0, len(A) - 1):
        min_index = i
        for j in range(i + 1, len(A)):
            if A[j] < A[min_index]:
                min_index = j
        if min_index != i:
            A[i], A[min_index] = A[min_index], A[i]
    return A


print(selection_sort(A1))


# Bubble Sort
def bubbleSort(A):
    for i in range(0, len(A) - 1):
        for j in range(0, len(A) - 1 - i):
            if A[j] > A[j + 1]:
                A[j], A[j + 1] = A[j + 1], A[j]
    return A


print(bubbleSort(A1))


# Merge Sort
def merge(A, first, middle, last):
    L = A[first:middle]
    R = A[middle:last + 1]
    L.append(999999999)
    R.append(999999999)
    i = j = 0
    for k in range(first, last + 1):
        if L[i] <= R[j]:
            A[k] = L[i]
            i += 1
        else:
            A[k] = R[j]
            j += 1


def merge_sort2(A, first, last):
    if first < last:
        middle = (first + last) // 2
        merge_sort2(A, first, middle)
        merge_sort2(A, middle + 1, last)
        merge(A, first, middle, last)


def merge_sort(A):
    merge_sort2(A, 0, len(A) - 1)
    return A
```

```
print(merge_sort(A1))


# Quick Sort Algorithm
def get_pivot(A, low, hi):
    mid = (hi + low) // 2
    pivot = hi

    if A[low] < A[mid]:
        if A[mid] < A[hi]:
            pivot = mid
    elif A[low] < A[hi]:
        pivot = low
    return pivot


def partition(A, low, hi):
    pivotIndex = get_pivot(A, low, hi)
    pivotValue = A[pivotIndex]
    A[pivotIndex], A[low] = A[low], A[pivotIndex]
    border = low

    for i in range(low, hi + 1):
        if A[i] < pivotValue:
            border += 1
            A[i], A[border] = A[border], A[i]
    A[low], A[border] = A[border], A[low]
    return border


def quick_sort2(A, low, hi):
    if low < hi:
        p = partition(A, low, hi)
        quick_sort2(A, low, p - 1)
        quick_sort2(A, p + 1, hi)


def quick_sort(A):
    quick_sort2(A, 0, len(A) - 1)
    return A


# Shell Sort
numbers = [54, 26, 93, 17, 77, 31, 44, 55, 20]


def shell_sort(A):
    subListCount = len(numbers) // 2
    while subListCount > 0:
        for i in range(subListCount):
            SubListInsertionSort(numbers, i, subListCount)

        print("After increments of size ", subListCount, "The list is ", numbers)
        subListCount = subListCount // 2


def SubListInsertionSort(numbers, start, gapSize):
    for i in range(start + gapSize, len(numbers), gapSize):
        currentValue = numbers[i]
        index = i

        while index >= gapSize and numbers[index - gapSize] > currentValue:
            numbers[index] = numbers[index - gapSize]
            index -= gapSize
        numbers[index] = currentValue
```

```
# Heap Sort
class MaxHeap:
    def __init__(self, items=[]):
        super().__init__()
        self.heap = [0]
        for i in items:
            self.heap.append(i)
            self.__floatUp(len(self.heap) - 1)

    def push(self, data):
        self.heap.append(data)
        self.__floatUp(len(self.heap) - 1)

    def peek(self):
        if self.heap[1]:
            return self.heap[1]
        else:
            return False

    def pop(self):
        if len(self.heap) > 2:
            self.__swap(1, len(self.heap) - 1)
            max = self.heap.pop()
            self.__bubbleDown(1)
        elif len(self.heap) == 2:
            max = self.heap.pop()
        else:
            max = False
        return max

    def __swap(self, i, j):
        self.heap[i], self.heap[j] = self.heap[j], self.heap[i]

    def __floatUp(self, index):
        parent = index // 2
        if index <= 1:
            return
        elif self.heap[index] > self.heap[parent]:
            self.__swap(index, parent)
            self.__floatUp(parent)

    def __bubbleDown(self, index):
        left = index * 2
        right = index * 2 + 1
        largest = index
        if len(self.heap) > left and self.heap[largest] < self.heap[left]:
            largest = left
        if len(self.heap) > right and self.heap[largest] < self.heap[right]:
            largest = right
        if largest != index:
            self.__swap(index, largest)
            self.__bubbleDown(largest)


m = MaxHeap([95, 3, 21])
m.push(10)
print(str(m.heap[0:len(m.heap)]))
print(str(m.pop()))
print(str(m.heap[0:len(m.heap)]))
```