Heaps and Heapsort

.add() Method

The `.add()` method is used in the `MaxHeap` class to add new values to a max-heap while maintaining the max-heap property that a parent must have a larger value than its children.

```
def add(self, element):
  self.count += 1
  print("Adding: {0} to {1}".format(element,
self.heap_list))
  self.heap_list.append(element)
  self.heapify_up()
```

.heapify_up() Method

The `.heapify_up()` method is used in the `MaxHeap` class to rebalance the heap data structure after an element is added to it.
Starting at the end of the heap where the new element is placed, the new element is compared to its parent value. If the parent has a smaller value than the child, the values swap places. This process repeats itself until an element has no parent value.

```
def heapify_up(self):
  print("Heapifying up")
  idx = self.count
  while self.parent_idx(idx) > 0:
    child = self.heap_list[idx]
    parent = self.heap_list[self.parent_idx(idx)]
    if parent < child:
      print("swapping {0} with {1}".format(parent, child))
      self.heap_list[idx] = parent
      self.heap_list[self.parent_idx(idx)] = child
    idx = self.parent_idx(idx)
  print("Heap Restored {0}".format(self.heap_list))
```

Heapsort

Heapsort is a sorting algorithm that utilizes the heap data structure to sort an unordered list of data.

To implement a heapsort algorithm, take the following steps:

- Add items of an unsorted list into a max-heap.
- While there is at least one element in the heap, remove the root of the heap and place it at the beginning of a list that will hold the sorted values. Whenever the root is extracted, the heap must be rebalanced.
- Once the heap is empty, return the sorted list.

```
def heapsort(lst):
  sort = []
  max_heap = MaxHeap()
  # Add items of an unsorted list into a max-heap.
  for idx in lst:
    max_heap.add(idx)
  # While there is at least one element in the heap, remove the root of the heap and place it at the beginning of a list that will hold the sorted values. Whenever the root is extracted, the heap must be rebalanced.
  while max_heap.count > 0:
    max_value = max_heap.retrieve_max()
    sort.insert(0, max_value)
  # Return the sorted list
  return sort

my_list = [99, 22, 61, 10, 21, 13, 23]
sorted_list = heapsort(my_list)
print(sorted_list) # Prints: [10, 13, 21, 22, 23, 61, 99]
```

.retrieve_max() Method

The `.retrieve_max()` method is used in the heapsort algorithm to return the largest value in a heap.
In this method, the root of the heap is extracted and replaced by the last element in the heap. Then, the method rebalances the heap data structure using `.heapify_down()`. Finally, the method returns the largest value.

```
def retrieve_max(self):
  if self.count == 0:
    print("No items in heap")
    return None
  # Store the largest value in a variable
  max_value = self.heap_list[1]
  print("Removing: {0} from {1}".format(max_value,
self.heap_list))
  # Replace the root of the heap with the last element in
the list
  self.heap_list[1] = self.heap_list[self.count]
  # Decrease the count
  self.count -= 1
  # Remove the last element in the list
  self.heap_list.pop()
  print("Last element moved to first:
{0}".format(self.heap_list))
  # Rebalance the heap
  self.heapify_down()
  # Return the largest value
  return max_value
```

.heapify_down() Method

The `.heapify_down()` method is used in the heapsort algorithm to rebalance the heap data structure after the root is removed and replaced with the last element in the heap.

While an element contains a child value, the parent value is compared with the value of its largest child. The larger child is determined using the `.get_larger_child_idx()` method. If the parent has a smaller value than the child, the two elements are swapped. Once an element has no children, the heap is restored.

```
def heapify_down(self):
  idx = 1
  # This while loop will execute as long as a child element is present
  while self.child_present(idx):
    print("Heapifying down!")
    # Get the index of the child element with the larger value
    larger_child_idx = self.get_larger_child_idx(idx)
    child = self.heap_list[larger_child_idx]
    parent = self.heap_list[idx]
    # If the parent value is less than the child value, swap the values
    if parent < child:
      self.heap_list[idx] = child
      self.heap_list[larger_child_idx] = parent
    idx = larger_child_idx
  print("HEAP RESTORED! {0}".format(self.heap_list))
```

.get_larger_child_idx() Method

The `.get_larger_child_idx()` method, which is used in the heapsort algorithm, compares the values of an element's children and returns the index of the child with the larger value.

```
def get_larger_child_idx(self, idx):
  # Check if a right child exists
  if self.right_child_idx(idx) > self.count:
    print("There is only a left child")
    return self.left_child_idx(idx)
  else:
    left_child = self.heap_list[self.left_child_idx(idx)]
    right_child = self.heap_list[self.right_child_idx(idx)]
    # Compare the left and right child values and return the
index of the larger child
    if left_child > right_child:
      print("Left child "+ str(left_child) + " is larger
than right child " + str(right_child))
      return self.left_child_idx(idx)
    else:
      print("Right child " + str(right_child) + " is larger
than left child " + str(left_child))
      return self.right_child_idx(idx)
```