Homework 6

Blen Daniel Assefa

Problem 6.1 Bubble Sort & Stable and Adaptive Sorting

(a) Bubble Sort is a sorting algorithm that works by repeatedly iterating through the list to be sorted, comparing each pair of adjacent items, and swapping them if they are in the wrong order. This is repeated until no swaps are needed, which indicates that the list is sorted. Write down the Bubble Sort algorithm in pseudocode, including comments to explain the steps and/or actions.

```
bubbleSort(list)              /*  list[p .. q]                        */
  size = list.size            /*  get size of the list by list.len()  */
  for i = 0 to size-1 do
    swapped = false
    for j = 0 to size-1 do
      /* compare the adjacent elements */
      if list[j] > list[j+1] then
        /* swap them */
        swap( list[j], list[j+1] )
        swapped = true
      end if
    end for
    /*if no number was swapped that means
    array is sorted now, break the loop.*/
    if(not swapped) then
      break
    end if
  end for
  return list
```

(b) Determine and prove the asymptotic worst-case, average-case, and best-case time complexity of Bubble Sort

Worst-case:

Worst case be a reverse order array. It would need both loops to work hence the complexity would be $\Theta(n(n-1))$ *which is* $\Theta(n^2)$complexity.

Average-case:

Average case is when half of the elements need to be swapped so it would be $\Theta(\frac{n(n-1)}{2} * \frac{1}{2})$which would result in $\Theta(n^2)$ complexity.

Best-case:

Best case would be when the array is already sorted. If that is the case, the loop would only take place once so the complexity will be $\Theta(n)$.

(c) Stable sorting algorithms maintain the relative order of records with equal keys (i.e., values). Thus, a sorting algorithm is stable if whenever there are two records R and S with the same key and with R appearing before S in the original list, R will appear before S in the sorted list. Which of the sorting algorithms Insertion Sort, Merge Sort, Heap Sort, and Bubble Sort are stable? Explain your answers

Insertion sort:
Stable sorting algorithm if elements are in order they will not be swapped.

Merge sort:
When we are messing the array at the end we must make sure we use S <= R and if this is the case it is stable, otherwise it is unstable.

Bubble sort:
Stable sorting algorithm because if they are in order they will not be swapped.

Heap sort:
This is an unstable algorithm as even if the elements are in order they must be swapped.

(d) A sorting algorithm is adaptive, if it takes advantage of existing order in its input. Thus, it benefits from the pre-sortedness in the input sequence and sorts faster. Which of the sorting algorithms Insertion Sort, Merge Sort, Heap Sort, and Bubble Sort are adaptive? Explain your answers.

Insertion sort
It is adaptive as a best case $\Theta(n)$ and worst case is $\Theta(n^2)$.

Merge sort
It is non-adaptive as both the best case and worst case come out to be $\Theta(nlogn)$ completes.

Heap sort
It is non-adaptive as both the best case and worst case have $\Theta(nlogn)$ complexity.

Bubble sort
It is adaptive as best case has $\Theta(n)$complexity whereas worst case has $\Theta(n^2)$ complexity.

Problem 6.2 Heap Sort

(a) Implement the Heap Sort algorithm as presented in the lecture.

Implementation is inside heapsort.py

(b) Implement a variant of the Heap Sort that works as follows: In the first step it also builds a max-heap. In the second step, it also proceeds as the Heap Sort does, but instead of calling MAX-HEAP IF Y , it always floats the new root all the way down to a leaf level. Then, it checks whether that was actually correct and if not fixes the max-heap by moving the element up again. This strategy makes sense when considering that the element that was swapped to become the new root is typically small and thus would float down to a leaf level in most cases. Hence, one would save the additional tests when floating down the element. And, the fixing step (moving the element upwards again) would be a rare case. This variant is called Bottom-up Heap Sort.

Implementation is inside heapsortvariant.py

(c) Compare the original Heap Sort and its variant from subpoint (b) for input sequences of different lengths (including larger input sequences). What can you observe?

As you can see from the graph the variant takes in less time after a high number of N has been reached. (according to the data that has been shown in my plot)