Homework 3 : Algorithms and Data Structures

Digdarshan Kunwar

March 2019

Problem 4.1

Bubble Sort Stable and Adaptive Sorting

(a) The pseudo code for bubblesort is given below:

Algorithm 1 BubbleSort Pseudo code

```
1: procedure BUBBLESORT(A)                          ▷ Procedure Declaration
2:     for i = 0 to A.length do
3:         swapflag=False
4:         for j= 0 to A.length-i-1 do
5:             if A[j] > A[j + 1] then
6:                 Swap (A[j] with A[j+1])
7:                 swapflag=True
8:             if swapflag==False then
9:                 break
10:
```

Implementation in Python:

```python
def bubbleSort(arr):
    size = len(arr)
    # Go through all the array elements
    for i in range(size):
        swapflag = False
        # The last i elements are in their sorted place.
        # So no need to go through the last i elements
        for j in range(0, size-i-1):
            # Go through the array from 0 to size-i-1
            # If the element found is greater then swap
            if arr[j] > arr[j+1] :
                arr[j], arr[j+1] = arr[j+1], arr[j]
                swapflag = True
        # This is a swap flag to check if there were any swaps in inner loop
        if swapflag == False:
            break
```

(b) Here,finding out the asymptotic worst-case, average-case, and best-case time complexity of Bubble Sort.

Worst Case:
In the worst case the array is sorted in reverse order.So for every iteration there would be decreasing number of comparisons in the inner loop.
So it means that initially for the first iteration there would be (n-1) comparisons and (n-1) swaps.So as the outer loop iteration increases the inner loop the iterations in inner-loop decreases by 1.

So we have:

$$\begin{aligned} T(n) &= O((n-1)+(n-2)+(n-3)+.....+2+1) \\ &= O\left(\frac{n(n-1)}{2}\right) \\ &= O(n^2) \end{aligned}$$

Average Case:
As for the average case there is the half probability of swaps in the algorithm.
So we can say:

We have : $\frac{n(n-1)}{2*2}$ swaps $\implies$ $\frac{n(n-1)}{4}$ swaps

Therefore the time complexity is :$\Theta(n^2)$
Best Case:
In the best case the array is sorted .So for the first iteration for outer loop there would be n number of comparisons in the inner loop which do not satisfy and if the swapflag inside the if condition is not changed then the array is already sorted.And therefore the the program breaks the outer loop. As a result the innerloop runs for only one time for n-1 iterations.
S0, the time complexity is $\Omega(n)$

(c) **Stability of an Algorithms**
A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appear in the input unsorted array.

Insertion Sort
Initially we have an unsorted random array:

4	6	8	2_a	2_b	3	9

⇒ **Insertion Sort** ⇒

2_a	2_b	3	4	6	8	9

Here as the two objects with equally key appear in the same order as before sort and after sort.Therefore **Insertion sort is stable sorting algorithm.**

Merge Sort
Initially we have an unsorted random array:

4	6	8	2_a	2_b	3	9

⇒ **Merge Sort** ⇒

2_a	2_b	3	4	6	8	9

Here as the two objects with equally key appear in the same order as before sort and after sort.Therefore,**Merge sort is a stable sorting algorithm.**

Heap Sort
Initially we have an unsorted random array:

4	6	8	2_a	2_b	3	9

⇒ **Heap Sort** ⇒

2_b	2_a	3	4	6	8	9

Here as the two objects with equally key don't appear in the same order as before sort and after sort.Therefore,**Heap sort is an unstable sorting algorithm.**

Bubble Sort
Initially we have an unsorted random array:

4	6	8	2_a	2_b	3	9

⇒**Bubble Sort**⇒

2_a	2_b	3	4	6	8	9

Here as the two objects with equally key appear in the same order as before sort and after sort.Therefore, **Bubble sort is a stable sorting algorithm.**

(d) **Adaptivity of an Algorithms**

An adaptive algorithm is an algorithm that changes its behavior at the time it is run, based on information available and on a priori defined reward mechanism (or criterion).

Insertion Sort

Insertion sort is adaptive sorting algorithm.

Best Case Complexity : $O(n)$
Worst Case Complexity : $O(n^2)$

Here there is change in the behaviour of time complexity in best and worst cases. Therefore it is **adaptive** sorting algorithm.

Merge Sort

Merge sort is a adaptive sorting algorithm.

Best Case Complexity :$O(n \log n)$
Worst Case Complexity :$O(n \log n)$

Here there is no change in the behaviour of time complexity in best and worst cases. Therefore it is **non-adaptive** sorting algorithm.

Heap Sort

Heap sort is an non-adaptive sorting algorithm.

Best Case Complexity : $O(n \log n)$
Worst Case Complexity : $O(n \log n)$

Here there is no change in the behaviour of time complexity in best and worst cases. Therefore it is **non-adaptive** sorting algorithm.

Bubble Sort

Bubble sort is a adaptive sorting algorithm.

Best Case Complexity : $O(n)$
Worst Case Complexity : $O(n^2)$

Here there is change in the behaviour of time complexity in best and worst cases. Therefore it is **adaptive** sorting algorithm.

Problem 4.2

Heap Sort

!! NOTE !!
Run file named comparison.py from the zip file.
In the zip file within the folder **Plots** are the testcases graph of algorithms.

(a) The implementation of heapsort is attached in the file heapsor.py

(b) The implementation of heapsort variant is attached in the file heapsor_variant.py

(c) The comparisons of the two variants of heapsort in done in comparison.py.
A graph is generated comparing two variants.
The graph is included in the zip file.
The variant of heap-sort seems to take less time for implementation for n number of elements in the an array.This is because for every parent node where there is a swap we had a max heapify function executing but now, the element that was swapped and was in the top of the heap goes to the leaf of the tree and goes up the nodes if needed which save the time.This reduces the time by a certain factor.
Check the comparision graphs in the Plots Folder in the Zip File