Homework 6

Hamza Bouhelal

March 2021

Problem 6.1
a)

```
Bubble_sort(lst, n)
    for j= 0 to n-1
        for i= 0 to n-j-1
            if lst[i]>lst[i+1]
                temp = lst[i]
                lst[i] = lst[i+1]
                lst[i+1] lst[i]
```

b)

```
def best_case(n):
    #   O(n)
    lst = []
    for _ in range(n):
        lst.append(randrange(50))
    return sorted(lst)

def Average_case(n):
    #   O(n^2)
    lst = []
    for _ in range(n):
        lst.append(randrange(50))
    return lst

def worst_case(n):
    #   O(n^2)
    lst = []
    for _ in range(n):
        lst.append(randrange(50))
    return sorted(lst, reverse=True)
```

After implementation of all of best, average and worst case, we find for different value of n that:

```
Best Case:
n = 0 in 0.005200 *10^-3 second
n = 1 in 0.001800 *10^-3 second
n = 2 in 0.003800 *10^-3 second
n = 3 in 0.004300 *10^-3 second
n = 4 in 0.005500 *10^-3 second
n = 5 in 0.007100 *10^-3 second
n = 6 in 0.008400 *10^-3 second
n = 8 in 0.012900 *10^-3 second
n = 10 in 0.017300 *10^-3 second
n = 13 in 0.026300 *10^-3 second
n = 16 in 0.036300 *10^-3 second
n = 20 in 0.053400 *10^-3 second
n = 25 in 0.079000 *10^-3 second
n = 30 in 0.109300 *10^-3 second
n = 38 in 0.485500 *10^-3 second
n = 48 in 0.271700 *10^-3 second
n = 60 in 0.405300 *10^-3 second
n = 75 in 0.620600 *10^-3 second
n = 80 in 0.702000 *10^-3 second
n = 100 in 1.067200 *10^-3 second
```

```
Average Case:
n = 0 in 0.001900 *10^-3 second
n = 1 in 0.001400 *10^-3 second
n = 2 in 0.002800 *10^-3 second
n = 3 in 0.005300 *10^-3 second
n = 4 in 0.006400 *10^-3 second
n = 5 in 0.008300 *10^-3 second
n = 6 in 0.012500 *10^-3 second
n = 8 in 0.018200 *10^-3 second
n = 10 in 0.023800 *10^-3 second
n = 13 in 0.043800 *10^-3 second
n = 16 in 0.057900 *10^-3 second
n = 20 in 0.095200 *10^-3 second
n = 25 in 0.127200 *10^-3 second
n = 30 in 0.186000 *10^-3 second
n = 38 in 0.555200 *10^-3 second
n = 48 in 0.475700 *10^-3 second
n = 60 in 0.726200 *10^-3 second
n = 75 in 1.058200 *10^-3 second
n = 80 in 1.327500 *10^-3 second
n = 100 in 1.884500 *10^-3 second
```

```
Worst Case:
n = 0 in 0.001600 *10^-3 second
n = 1 in 0.001400 *10^-3 second
n = 2 in 0.003800 *10^-3 second
n = 3 in 0.005100 *10^-3 second
n = 4 in 0.007500 *10^-3 second
n = 5 in 0.010400 *10^-3 second
n = 6 in 0.013800 *10^-3 second
n = 8 in 0.022500 *10^-3 second
n = 10 in 0.033300 *10^-3 second
n = 13 in 0.054900 *10^-3 second
n = 16 in 0.077800 *10^-3 second
n = 20 in 0.118700 *10^-3 second
n = 25 in 0.185600 *10^-3 second
n = 30 in 0.734600 *10^-3 second
n = 38 in 0.428000 *10^-3 second
n = 48 in 0.674000 *10^-3 second
n = 60 in 1.033600 *10^-3 second
n = 75 in 1.620200 *10^-3 second
n = 80 in 1.840900 *10^-3 second
n = 100 in 4.381100 *10^-3 second
```

c) Insertion Sorts iterates through our array and swap the element we're in to the place it belongs to to its right, and therefore it will not swap two keys with same value, it is therefore stable.

Merge Sort is stable, since it divides our problem to sub-problems, then it merge back our sub-problems, by comparing the first element of each array and swapping when needed, it will therefore never swap two keys with the same value.

Heap Sort is unstable, since for an array with R and S the same value and - any other value : -RS— $\Longrightarrow$ RS—-$\Longrightarrow$ S–R– $\Longrightarrow$ –SR–

Bubble sort is stable, since it repeatedly iterates through the string comparing 2 elements and swapping them if needed it will therefore never swap two keys of same value.

d) Insertion Sorts iterates through our array and swap the element we're in to the place it belongs to to its right, it is taking advantage of existing order, adding to it the elements missing, it is therefore adaptive.

adaptive Merge Sort is adaptive, since it divides our problem to sub-problems, in all of our sub-problems, some might already be sorted, which means no swap will be needed.
Others variant of the Merge sort are not adaptive since the problem gets totally divided and the order therefore doesn't matter anymore.

Heap Sort is not adaptive since any ordered combination placed at the beginning of the array is will react like this:
ABC—$\Longrightarrow$ BC–A-$\Longrightarrow$ C–AB-$\Longrightarrow$ –ABC- with A < B < C

Bubble sort is not adaptive, since it repeatedly iterates through the string comparing 2 elements and swapping them if needed it will therefore not use existing order:
DABC$\Longrightarrow$ ADBC$\Longrightarrow$ ABDC$\Longrightarrow$ ABCD with A < B < C < D

Problem 6.2
a) - b) Heap sort.py

c)
After running both Heap sort and Heap sort bottom up for different values of n, we find that:

```
Heap sort:
n = 0 in 0.008000 *10^-3 second
n = 1 in 0.002900 *10^-3 second
n = 2 in 0.008200 *10^-3 second
n = 3 in 0.006600 *10^-3 second
n = 4 in 0.008700 *10^-3 second
n = 5 in 0.009700 *10^-3 second
n = 6 in 0.014700 *10^-3 second
n = 8 in 0.021000 *10^-3 second
n = 10 in 0.027900 *10^-3 second
n = 13 in 0.040000 *10^-3 second
n = 16 in 0.050100 *10^-3 second
n = 20 in 0.070700 *10^-3 second
n = 25 in 0.092600 *10^-3 second
n = 30 in 0.114200 *10^-3 second
n = 38 in 0.150200 *10^-3 second
n = 48 in 0.202000 *10^-3 second
n = 60 in 0.277000 *10^-3 second
n = 75 in 0.361700 *10^-3 second
n = 80 in 0.397500 *10^-3 second
n = 100 in 0.518600 *10^-3 second
n = 200 in 1.565500 *10^-3 second
n = 500 in 2.770600 *10^-3 second
n = 1000 in 9.867900 *10^-3 second
```

```
Bottom up Heap sort:
n = 0 in 0.004000 *10^-3 second
n = 1 in 0.002700 *10^-3 second
n = 2 in 0.007000 *10^-3 second
n = 3 in 0.007800 *10^-3 second
n = 4 in 0.011600 *10^-3 second
n = 5 in 0.013100 *10^-3 second
n = 6 in 0.016300 *10^-3 second
n = 8 in 0.023200 *10^-3 second
n = 10 in 0.031800 *10^-3 second
n = 13 in 0.040800 *10^-3 second
n = 16 in 0.054000 *10^-3 second
n = 20 in 0.072700 *10^-3 second
n = 25 in 0.097500 *10^-3 second
n = 30 in 0.119700 *10^-3 second
n = 38 in 0.160700 *10^-3 second
n = 48 in 0.216500 *10^-3 second
n = 60 in 0.280900 *10^-3 second
n = 75 in 0.369500 *10^-3 second
n = 80 in 0.400800 *10^-3 second
n = 100 in 0.523700 *10^-3 second
n = 200 in 1.555400 *10^-3 second
n = 500 in 2.559200 *10^-3 second
n = 1000 in 4.702700 *10^-3 second
```

We can therefore conclude that The Heap sort Algorithm is more time efficient for small value of n, and that the Bottom up Heap sort is more time efficient for big value of n.