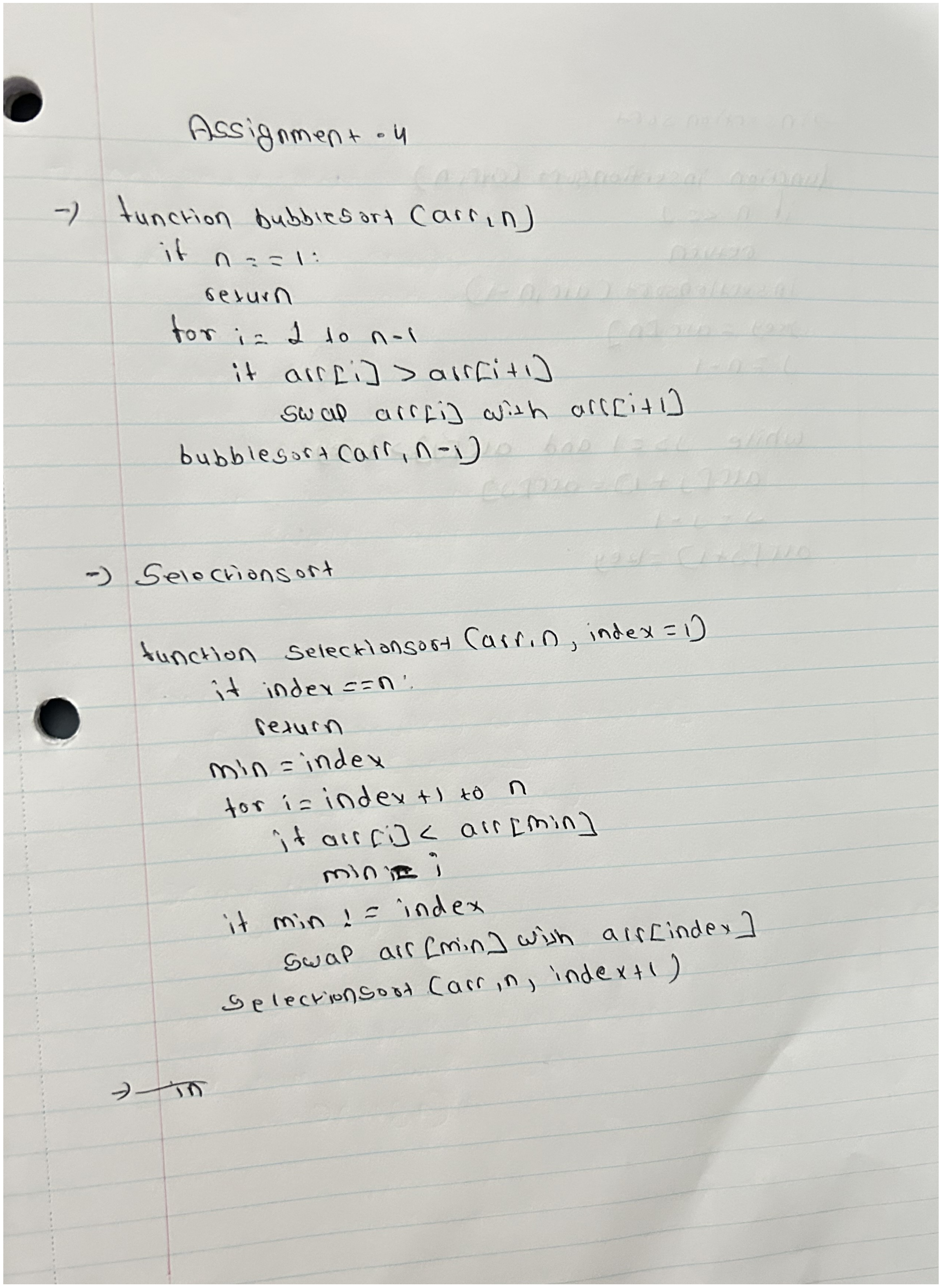
Assignment 4

Manan Patel

September 2024

1: . To sort an array A of integers with n as length, write recursive algorithms for (a) Bubble Sort, (b) Selection Sort, and (c) Insertion Sort, respectively.



Assignment - 4

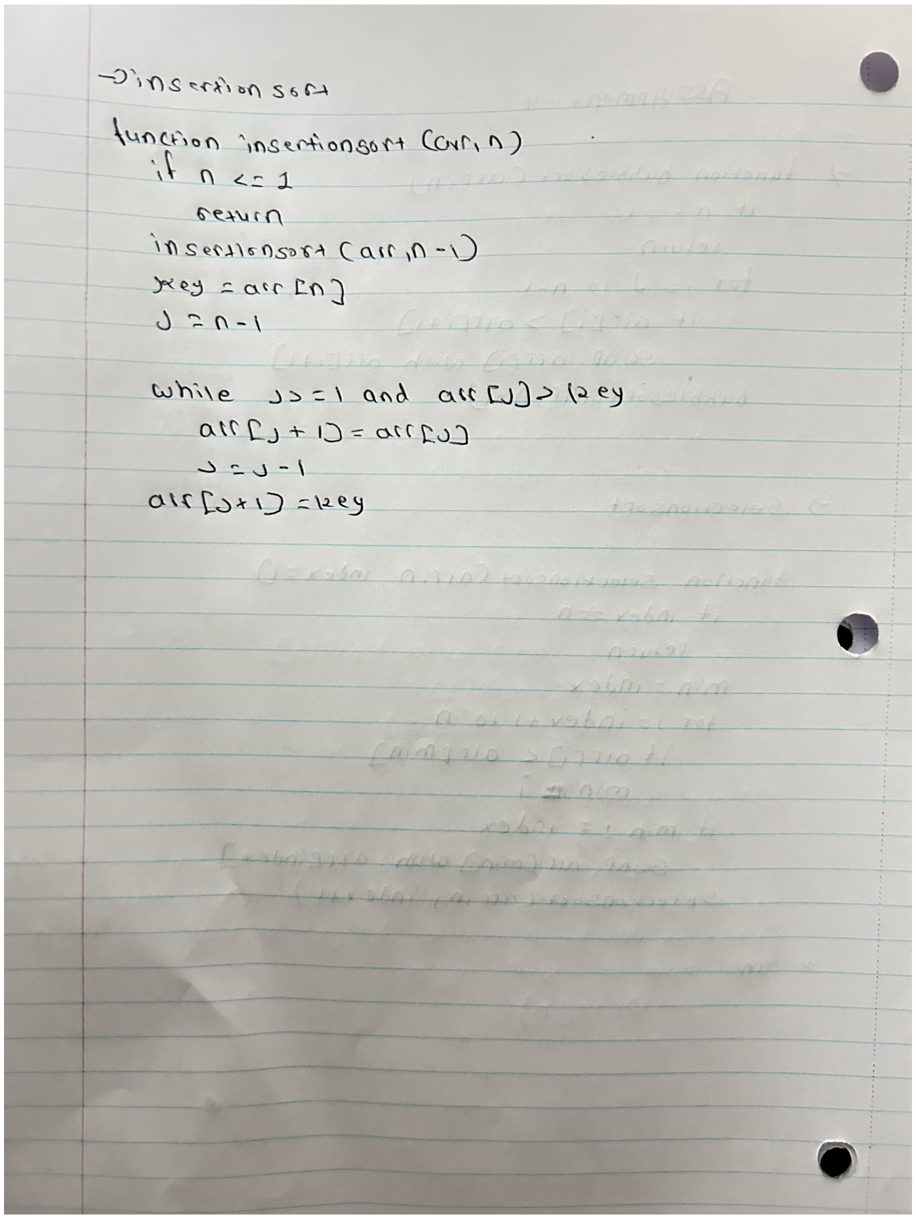
```
-> function bubblesort (arr, n)
     if n == 1:
       return
     for i = 1 to n-1
         if arr[i] > arr[i+1]
             swap arr[i] with arr[i+1]
     bubblesort(arr, n-1)
```

```
-> Selectionsort

   function Selectionsort (arr, n, index = 1)
       if index == n:
         return
       min = index
       for i = index+1 to n
           if arr[i] < arr[min]
               min = i
       if min != index
           swap arr[min] with arr[index]
       Selectionsort (arr, n, index+1)
```

-> ~~in~~

Figure 1: Answer of Task 1



```
→ insertion sort

function insertionsort (arr, n)
  if n <= 1
     return
  insertionsort (arr, n-1)
  key = arr [n]
  j = n-1

  while  j>=1 and arr [j] > key
     arr [j + 1] = arr [j]
     j = j - 1
  arr [j+1] = key
```

Figure 2: Answer of Task 1

2-Compare the time efficiency of Lomuto Partition and Hoare Partition using the methods we have covered and try to be as accurate as possible

Best Case

The best case occurs when the pivot divides the array into two nearly equal subarrays, ensuring the most balanced recursive calls, minimizing recursion depth. This leads to a time complexity of O(n log n).

Lomuto Partition

Example Array:

$$A = [4, 2, 5, 1, 3]$$

Partition Process:

- **Pivot:** The last element, 3.
- After partitioning, the array becomes:
$$[2, 1, 3, 5, 4]$$
- The pivot 3 divides the array into:
  - Left subarray: $[2, 1]$
  - Right subarray: $[5, 4]$

Hoare Partition

Example Array:

$$A = [3, 1, 2, 5, 4]$$

Partition Process:

- **Pivot:** The first element, 3.
- After partitioning, the array rearranges to:
$$[2, 1, 3, 5, 4]$$
- The pivot 3 divides the array into:
  - Left subarray: $[2, 1]$
  - Right subarray: $[5, 4]$

Average Case

The average case happens when the pivot results in moderately balanced subarrays, neither too skewed nor perfectly balanced. This typically occurs with random input, yielding a time complexity of O(n log n) on average.

Lomuto Partition Average Case

Example Array:

$$A = [7, 2, 1, 6, 8, 5, 3, 4]$$

Partition Process:

- **Pivot:** The last element, 4.
- After partitioning, the array becomes:

$$[2, 1, 3, 4, 8, 5, 6, 7]$$

- The pivot 4 divides the array into:
  - Left subarray: $[2, 1, 3]$
  - Right subarray: $[8, 5, 6, 7]$

Hoare Partition Average Case

Example Array:

$$A = [4, 2, 7, 1, 3, 6, 5, 8]$$

Partition Process:

- **Pivot:** The first element, 4.
- After partitioning, the array rearranges to:

$$[3, 2, 1, 4, 6, 7, 5, 8]$$

- The pivot 4 divides the array into:
  - Left subarray: $[3, 2, 1]$
  - Right subarray: $[6, 7, 5, 8]$

Worst Case

The worst case occurs when the pivot is the smallest or largest element, resulting in highly unbalanced partitions. Although the pivot is placed correctly, all other elements still need to be sorted. This causes n recursive calls, each performing up to n comparisons, leading to a time complexity of O(n^2) due to inefficient sorting.

Lomuto Partition Worst Case

Example Array:

$$A = [1, 2, 3, 4, 5, 6, 7, 8]$$

Partition Process:

- **Pivot:** The last element, 8.
- After partitioning, the array remains unchanged:

$$[1, 2, 3, 4, 5, 6, 7, 8]$$

- The pivot 8 divides the array into:
  - Left subarray: $[1, 2, 3, 4, 5, 6, 7]$
  - Right subarray: [] (empty)

Hoare Partition Worst Case

Example Array:

$$A = [8, 7, 6, 5, 4, 3, 2, 1]$$

Partition Process:

- **Pivot:** The first element, 8.
- After partitioning, the array rearranges to:

$$[1, 7, 6, 5, 4, 3, 2, 8]$$

- The pivot 8 divides the array into:
  - Left subarray: $[1, 7, 6, 5, 4, 3, 2]$
  - Right subarray: [] (empty)

Case	**Lomuto Partition**	**Hoare Partition**
Best Case	$O(n \log n)$	$O(n \log n)$
Average Case	$O(n \log n)$	$O(n \log n)$
Worst Case	$O(n^2)$	$O(n^2)$

Table 1: Time Complexity of Lomuto and Hoare Partitions