19CSE302 - Design Analysis Of Algorithms
Lab Evaluation 1

Praneeth V - CB.EN.U4CSE22244
Sai Krishna - CB.EN.U4CSE22246

August 23, 2024

Lab Evaluation Questions

1. First Question Analysis of Sorting algorithms

   1.1. In-Place Quick Sort (with last element as the pivot)

   Quick sort applies the Divide-And-Conquer strategy to sort a subarray A[p..n]:

   Divide : The array is rearranged into two subarrays such that A[p .. q-1] is less than that of A[q] and the other subarray A[q+1 .. n] greater than A[q]. Here A[q] is the pivot element.

   Conquer : We sort the subarrays A[p .. q-1] and A[q+1 .. n]

   Combine: Now, we combine the sorted arrays together.

   The time complexity of the In-Place Quick Sort depends on whether the partitioning is balanced or not, and depends on which elements are used for partitioning.

   If the partitioning is balanced: It's the best case where the algorithm runs as fast as merge sort. Since the partitioning is balanced, we get two even halves where one is of the size $\lfloor n/2 \rfloor$ and the other one of the size $\lfloor n/2 \rfloor - 1$. So the recurrence relation for this becomes: $T(n) = 2T(n/2) + \Theta(n)$ and by Master's Theorem, it becomes $T(n) = \Theta(n \log n)$.

   If the partitioning is unbalanced: Its the worst case where we have one subarray with size of $n-1$ and the other subarray with the size of 0. The time complexity of the partitioning costs $\Theta(n)$ time and the recurrence relation for this case becomes: $T(n) = T(n-1) + \Theta(n)$ and making the overall time complexity $T(n) = \Theta(n^2)$ and this case mostly happens when the array is already sorted.

   If the partitioning is based on proportionality: Its the average case where the partitioning is done based on some proportions where the partitioning is like x-to-10$-x$, then in this case the recurrence relation becomes like: $T(n) = T(x \cdot n/10) + T(n/10) + c \cdot n$ in which just

reaching the depth of the recursion tree for partition takes $\log_{10/x} n$ which is $\Theta(\log n)$ and the cost at each level is n making the time complexity: $T(n) = O(n \log n)$

The Space complexity for In-Place Quicksort is $O(\log n)$ for best and average case partitioning where *logn* represents the size of the recursion tree and in the worst case partitioning the space complexity becomes $O(n)$ as the recursion tree becomes skewed leading to the size of the tree to be equal to the number of elements in an array.

```
% Test Case 1: For size 100:
```

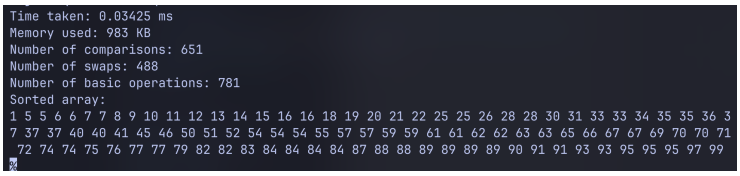

Figure 1: Description of the image

```
% Code Section 1.1.2
```

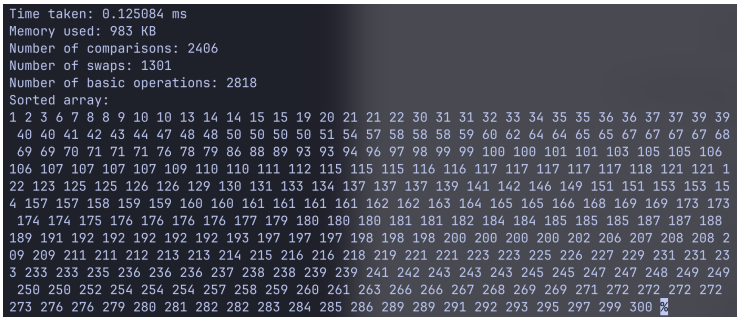

Figure 2: Description of the image

```
% Code Section 1.1.3
```

```
% Code Section 1.1.4
```

1.2. Subdivision 2

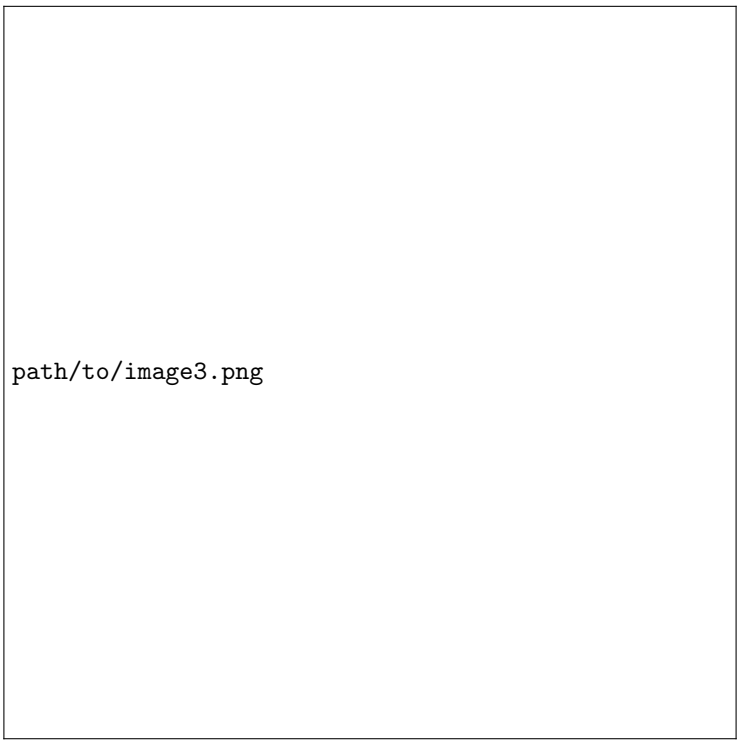

Figure 3: Description of the image

```
% Code Section 1.2.1

% Code Section 1.2.2

% Code Section 1.2.3

% Code Section 1.2.4
```

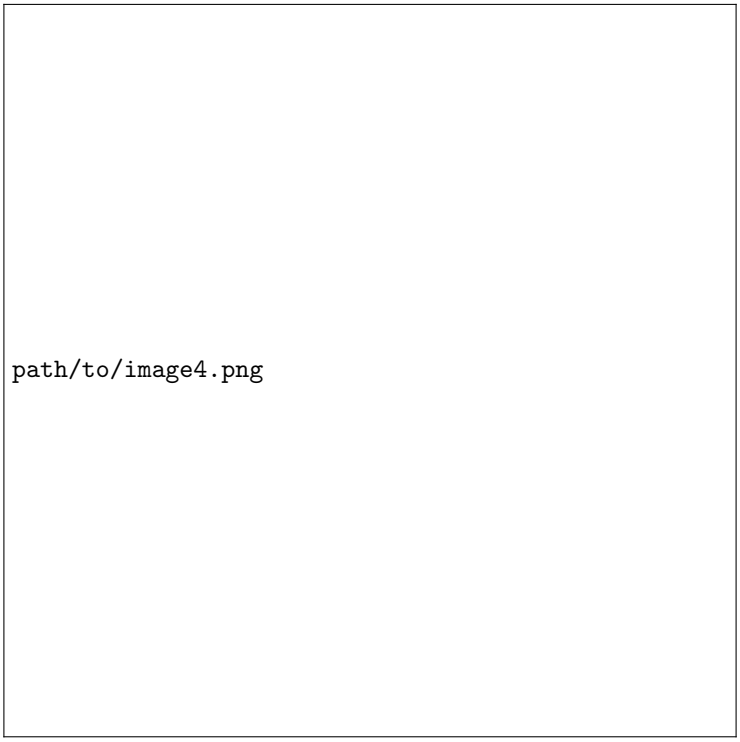

Figure 4: Description of the image

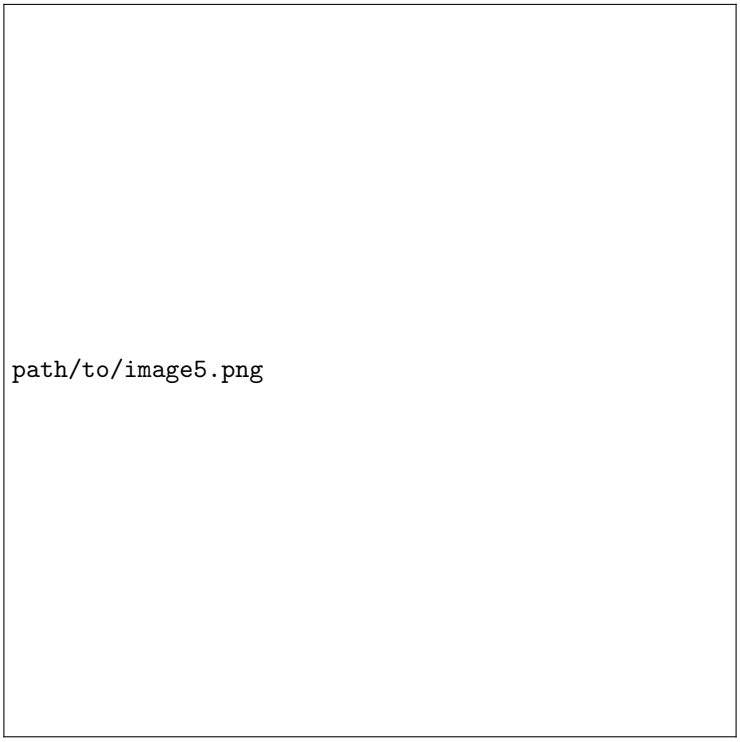

Figure 5: Description of the image

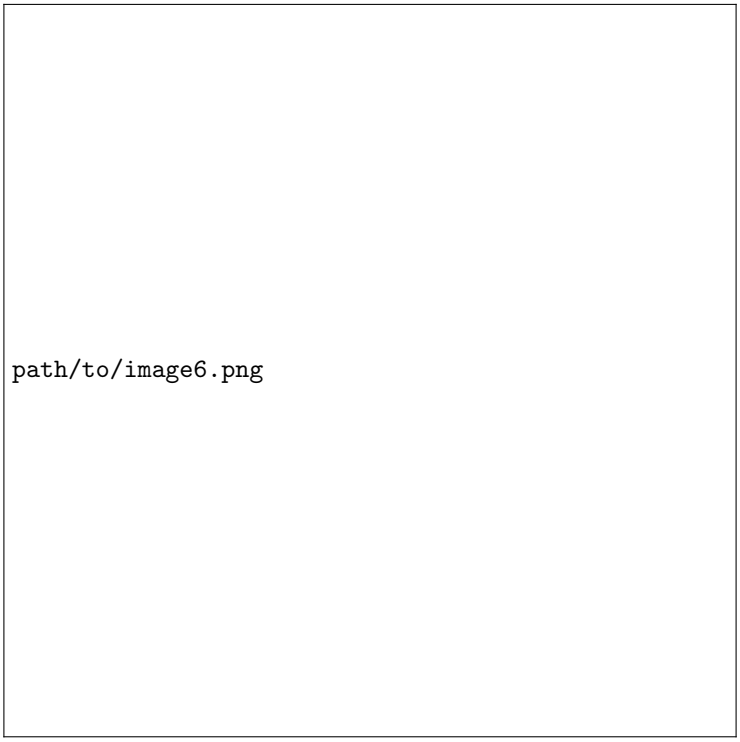

Figure 6: Description of the image

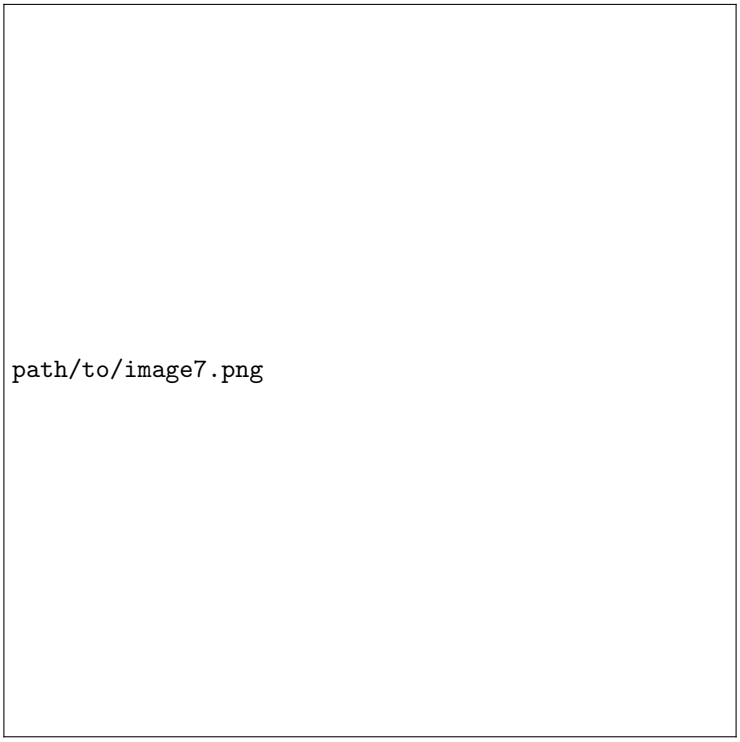

Figure 7: Description of the image

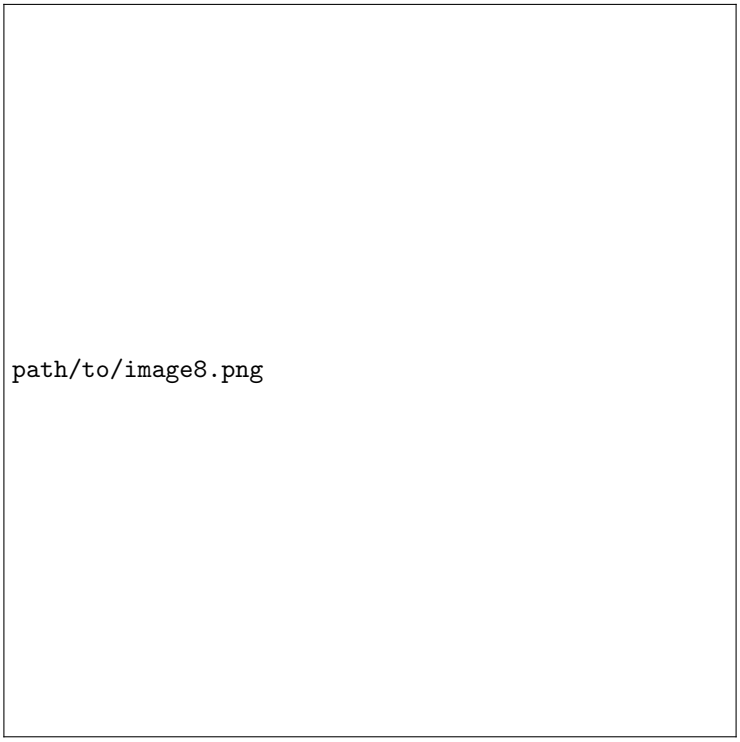

Figure 8: Description of the image