

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 $\log n$ 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:`

```
Time taken: 0.03425 ms
Memory used: 983 KB
Number of comparisons: 651
Number of swaps: 488
Number of basic operations: 781
Sorted array:
1 5 5 6 6 7 7 8 9 10 11 12 13 14 15 16 16 18 19 20 21 22 25 25 26 28 28 30 31 33 33 34 35 35 36 3
7 37 37 40 40 41 45 46 50 51 52 54 54 54 55 57 57 59 59 61 61 62 62 63 63 65 66 67 67 69 70 70 71
72 74 74 75 76 77 77 79 82 82 83 84 84 84 84 87 88 88 88 89 89 89 89 90 91 91 93 93 95 95 95 97 99
```

Figure 1: Description of the image

`% Code Section 1.1.2`

```
Time taken: 0.125084 ms
Memory used: 983 KB
Number of comparisons: 2406
Number of swaps: 1301
Number of basic operations: 2818
Sorted array:
1 2 3 6 7 8 8 9 10 10 13 14 14 15 15 19 20 21 21 22 30 31 31 32 33 34 35 35 36 36 37 37 39 39
40 40 41 42 43 44 47 48 48 50 50 50 50 51 54 57 58 58 58 59 60 62 64 64 65 65 67 67 67 67 68
69 69 70 71 71 71 76 78 79 86 88 89 93 93 94 96 97 98 99 99 100 100 101 101 103 105 105 106
106 107 107 107 107 109 110 110 111 112 115 115 115 116 116 117 117 117 117 117 118 121 121 1
22 123 125 125 126 126 129 130 131 133 134 137 137 137 139 141 142 146 149 151 151 153 153 15
4 157 157 158 159 159 160 160 161 161 161 161 162 162 163 164 165 165 166 168 169 169 173 173
174 174 175 176 176 176 177 179 180 180 180 181 181 182 184 184 185 185 185 187 187 188
189 191 192 192 192 192 192 193 197 197 197 198 198 198 200 200 200 200 202 206 207 208 208 2
09 209 211 211 212 213 213 214 215 216 216 218 219 221 221 223 223 225 226 227 229 231 231 23
3 233 233 235 236 236 236 237 238 238 239 239 241 242 243 243 245 245 247 247 248 249 249
250 250 252 254 254 254 257 258 259 260 261 263 266 266 267 268 269 269 271 272 272 272 272
273 276 276 279 280 281 282 282 283 284 285 286 289 289 291 292 293 295 297 299 300
```

Figure 2: Description of the image

`% Code Section 1.1.3`

`% Code Section 1.1.4`

1.2. Subdivision 2

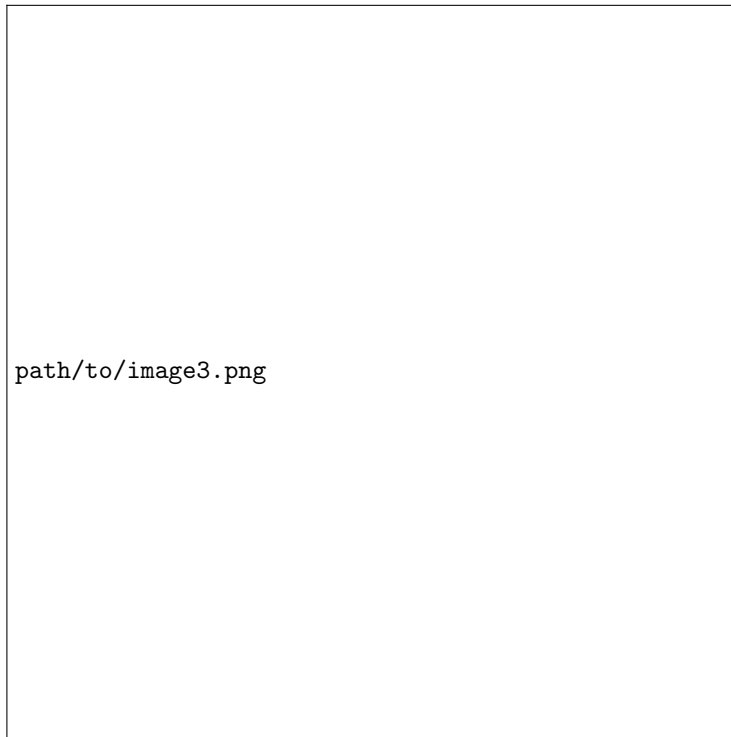


Figure 3: Description of the image

`% Code Section 1.2.1`

`% Code Section 1.2.2`

`% Code Section 1.2.3`

`% Code Section 1.2.4`

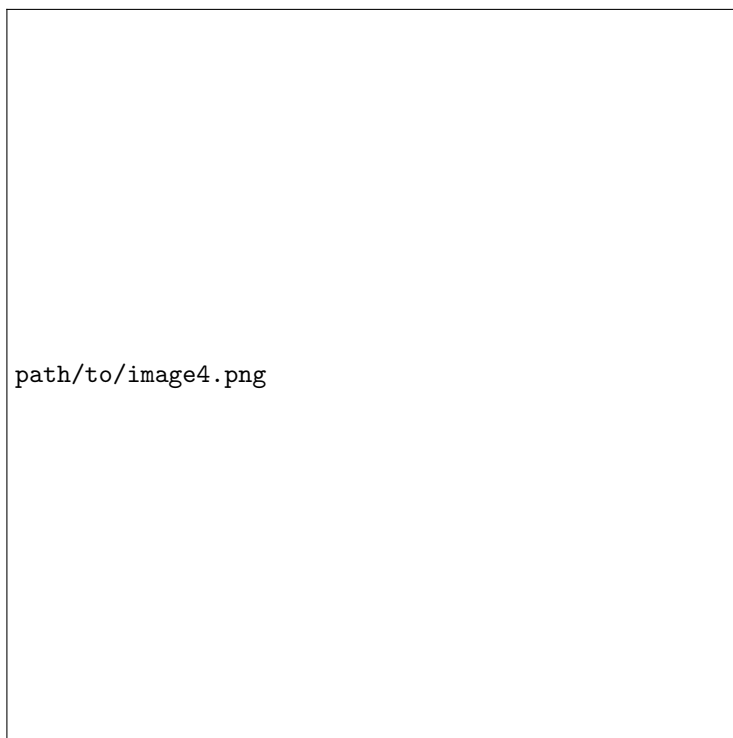


Figure 4: Description of the image

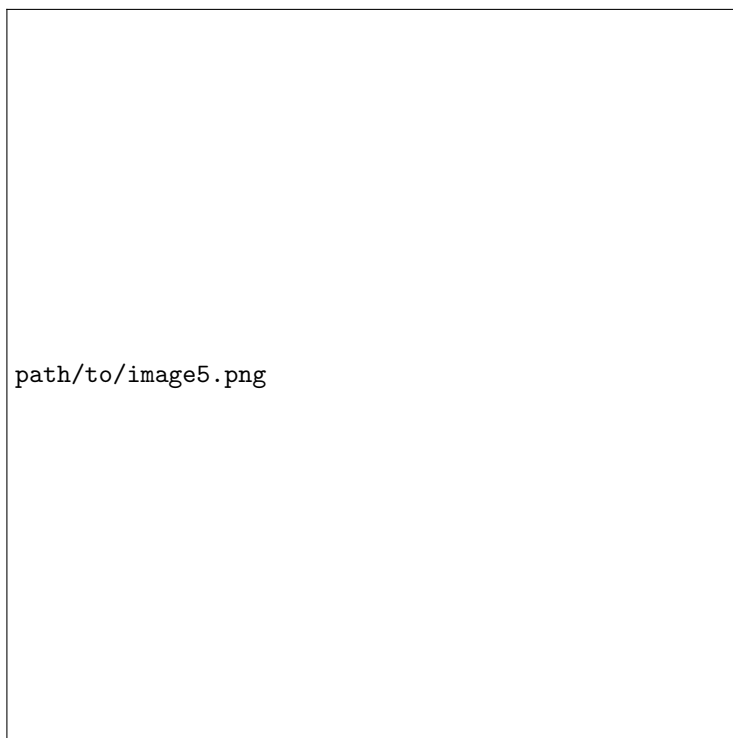


Figure 5: Description of the image

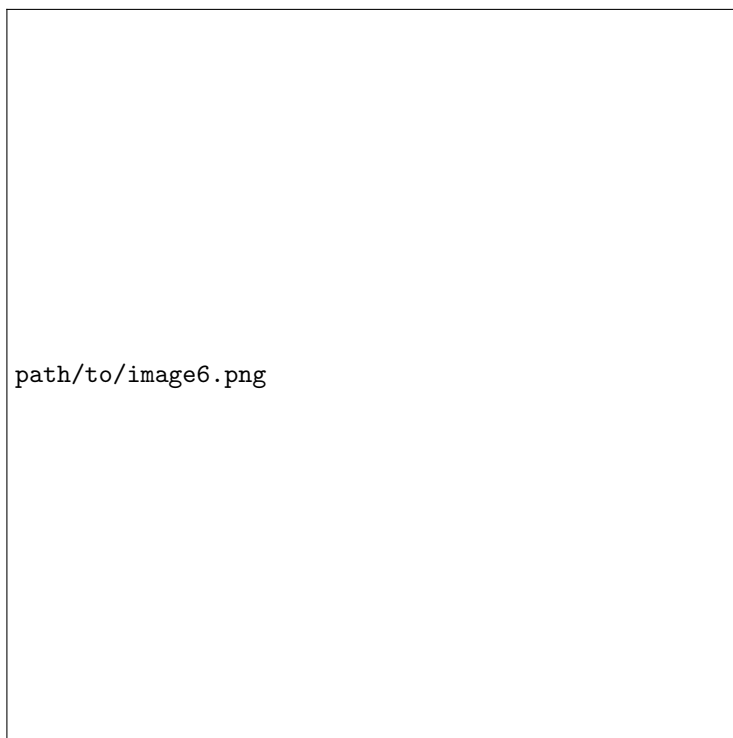


Figure 6: Description of the image

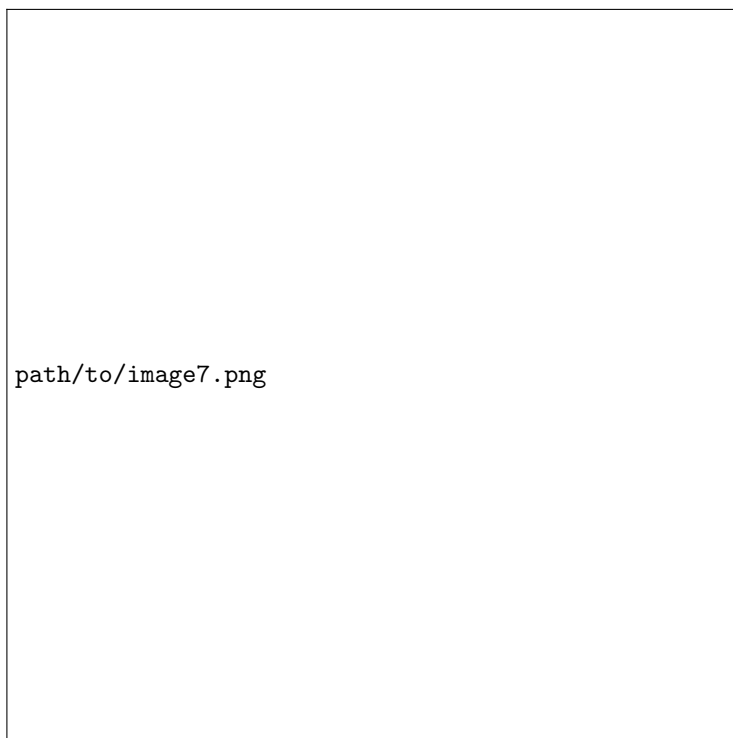


Figure 7: Description of the image

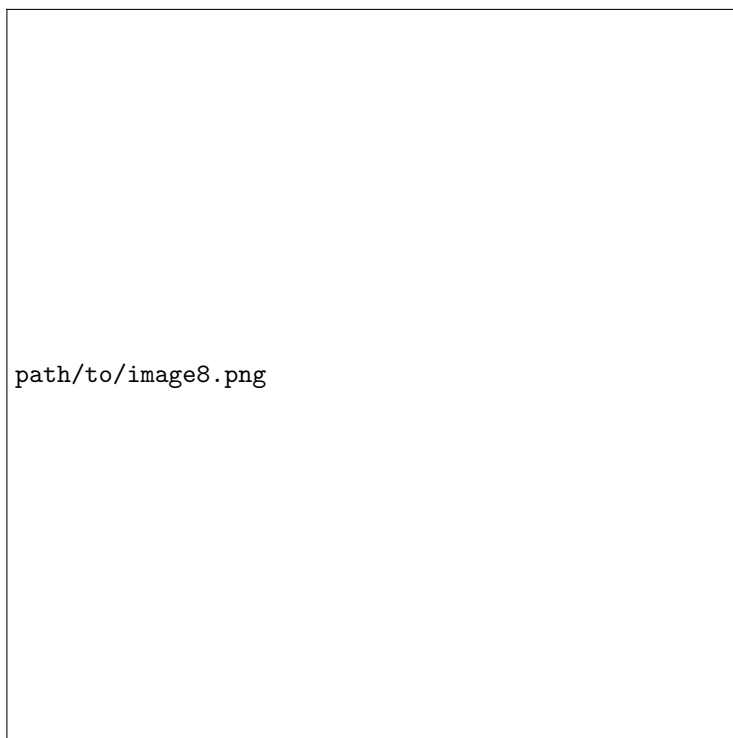


Figure 8: Description of the image