

**Analysis of Algorithms 1**

**HW Report**

**HW 2**

**Ahmet Göktuğ SEVİNÇ**

**150140120**

**02.11.2017**

**a)** Quick sort algorithm’s asymptotic upper bounds are: Θ(*n*​2​​) for the worst case, Θ(*n*lg*n*) for the best case and *O*(*n*lg*n*) for the average case.

|  |
| --- |
| For worst case:  T(0) = T(1) = 0 (base case)  T(N) = N + T(N-1)  T(N) = N + T(N-1)  T(N-1) = (N-1) + T(N-2)  T(N-2) = (N-2) + T(N-3)  ...  T(3) = 3 + T(2)  T(2) = 2 + T(1)  T(1) = 0  So, T(N) = N + (N-1) + (N-2) ... + 3 + 2 ≈ N2/2 which is Θ(*n*​2​​) |

|  |
| --- |
| For best case:  T(0) = T(1) = 0 (base case)  T(N) = 2T(N/2) + N  C:\Users\goktu\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Adsız.png  This is exactly same with merge sort which is Θ(*n*lg*n*) |

|  |
| --- |
| For average case:  Lets consider we get always 3-to-1 splits.  T(0) = T(1) = 0 (base case)  T(N) = 3T(N/4) + T(N/4) + N  C:\Users\goktu\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Adsız.png  Which is *O*(*n*lg*n*) |

**b)**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **N** | **1.** | **2.** | **3.** | **4.** | **5.** | **6.** | **7.** | **8.** | **9.** | **10.** | **Avg.** |
| **10** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **100** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1000** | 0.001 | 0.002 | 0.002 | 0.002 | 0.001 | 0 | 0.002 | 0.001 | 0.002 | 0.002 | 0,0013 |
| **10000** | 0.022 | 0.021 | 0.022 | 0.023 | 0.021 | 0.022 | 0.023 | 0.022 | 0.022 | 0.022 | 0.022 |
| **100000** | 0.328 | 0.345 | 0.314 | 0.333 | 0.326 | 0.323 | 0.321 | 0.324 | 0.324 | 0.320 | 0.325 |
| **5000000** | 2.015 | 2.459 | 2.558 | 2.263 | 2.389 | 2.333 | 2.356 | 2.343 | 2.330 | 2.562 | 2.360 |
| **10000000** | 4.233 | 5.202 | 5.656 | 5.520 | 5.130 | 5.327 | 5.544 | 4.567 | 5.016 | 5.452 | 5.164 |

Our program works in average case which is *O*(*n*lg*n*).

**c)** If input is sorted or reverse sorted Quick Sort algorithm works in worst case. In that situation, input list is always partitioned (n-1)-to- 1. To avoid this situation we can use randomized quick sort algorithm. To do this, we choose random pivots in partition function.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **N** | **1.** | **2.** | **3.** | **4.** | **5.** | **6.** | **7.** | **8.** | **9.** | **10.** | **Avg.** |
| **10** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **100** | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| **1000** | 0.083 | 0.081 | 0.082 | 0.084 | 0.083 | 0.084 | 0.081 | 0.083 | 0.082 | 0.080 | 0,082 |
| **10000** | 7.687 | 7.654 | 7.721 | 7.723 | 7.562 | 7.775 | 7.654 | 7.840 | 7.720 | 7.694 | 7.703 |

It works in Θ(*n*​2​​)

**d)** Quick Sort algorithm is not a stable algoritm because in the output file, equaly weighted data might not preserve their positions as in the input file. This situation occurs because of partition function.

For example we have such data in output file;

|  |
| --- |
| 0,75,79,female,00906,8600000US00906 |
| 0,67,69,female,00906,8600000US00906 |
| 0,22,24,female,00906,8600000US00906 |
|  |

Since we are sorting the data according to their population and geoID, these data have equal weights for our program. In the input file these data can be in this order;

|  |
| --- |
| 0,75,79,female,00906,8600000US00906 |
| 0,22,24,female,00906,8600000US00906 |
| 0,67,69,female,00906,8600000US00906 |

So, quick sort algorith does not preserve relative orders hence, it is not a stable algorithm.