Homework 5 : Algorithms and Data Structures

Digdarshan Kunwar

March 2019

Problem 5.1

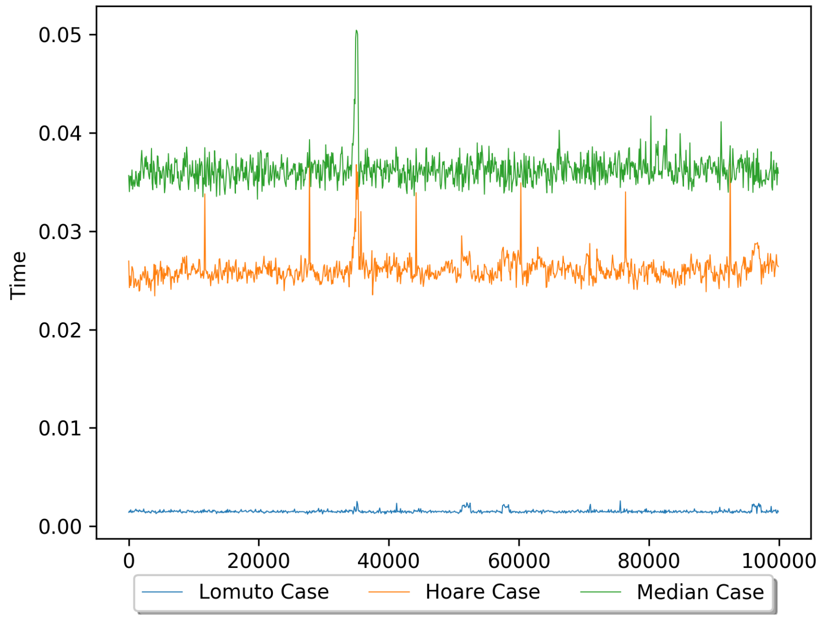
Quick Sort and its Variants

(a) It is implemented in lomuto.py.

(b) It is implemented in hoare.py.

(c) It is implemented in median.py.

(d) It is implemented in Comparison.py.



Average Timings
Lomuto Case : 0.0015162210464477754
Hoare Case : 0.026086989402770995
Median Case : 0.03623667860031128

Here from the result we see that Lomuto takes the least amount of time.
This is because lomutu has the least cost for the runtime. As for the median of the three it has the highest average time in my case, this is because the program takes extra cost during the extra Median of Three function.Thus it results in extra cost and shift of the extra time.As for the Hoare Case we have the average timing between the two other cases.Also Lomuto's partition and Hoare's partitioning both would cause runtime of any sorted input for the Quicksort to degrade to $O(n^2)$, if the pivot chosen is among the first or the last element.

Problem 5.2

Quick Sort Variant

> **<u>!! NOTE !!</u>**
> Run file named QuicksortVariant.py and RandomQuickSort.py from the zip file for QuicksortVariant and RandomQuickSort.

(a) It is implemented in QuicksortVariant.py.

(b) Here,
Best Case:
We will have best case when there is equal partition in all the recursive calls.As the partition iterates through the element only once,so have its complexity as O(n).For every recursive call we will have the recursive call that divides the array into 3 intervals.So We can write for the best cases :

$$T(n) = 3T(\frac{n}{3}) + O(n)$$

Using the master method we have :

$$T(n) = O(nlg(n))$$

The base of lg is 3.

Worst Case:
The worst case occurs when the array is sorted in ascending or descending order.
In out implementation we took the first two elements as the pivots.So if they are in some kind of order then they wont have any elements in between them so the pivots necessarily don't act as pivots as there are no elements in between.
So if the pivot1 and pivot2 are in order then the pivot2 in the end of the array has to come after the first element so it would be coming n-2 places back from the end.
So,

$$T(n) = T(n-2) + T(1) + T(1) + O(n)$$

After solving the recurrence we have:

$$T(n) = O(n^2)$$

(c) It is implemented in RandomQuickSort.py

Problem 5.3

Decision Tree

Write a different proof for $lg\ (n!) = \Theta(n\ lgn)$ (Lecture 9) without having to use Stirling's formula.
Here we know that,

$$\begin{aligned} lg\ (n!) &= lg(1*2*3*4*5*......(n-2)*(n-1)*n) \\ &= lg(1) + lg(2) + lg(3) + lg(4) + ... + lg(n-2) + lg(n-1) + lg(n) \end{aligned}$$

$$\begin{aligned} lg(1) + lg(2) + ... + lg(n-1) + lg(n) &\leqslant lg(n) + lg(n) + lg(n)... + lg(n) \\ &\leqslant n\ lg(n) \end{aligned}$$

Therefore, it implies $\implies \quad lg\ (n!) = O(n\ lg(n))$

Now for $\Omega(n\ lg(n))$

$$\begin{aligned} lg(1) + lg(2) + ... + lg(n-1) + lg(n) &\geqslant lg\left(\frac{n}{2}\right) ... + lg(n) \\ &\geqslant \sum_{i=n/2}^{n} \lg \tfrac{n}{2} \\ &\geqslant \tfrac{n}{2} \lg \tfrac{n}{2} \\ &= \tfrac{n}{2}\big(\lg n - \lg 2) \\ &= \Omega(n \lg n)\,. \end{aligned}$$

Therefore, it implies $\implies \quad lg\ (n!) = \Omega(n\ lg(n))$

Therefore form above as:
$lg\ (n!) = O(n\ lg(n))$ and $lg\ (n!) = \Omega(n\ lg(n))$
We have:
$lg\ (n!) = \Theta(n\ lg(n))$