FATIH SULTAN MEHMET VAKIF UNIVERSITY

2021-2022 ACADEMIC YEAR FALL SEMESTER

BLM19307E ALGORITHM ANALYSIS & DESIGN ASSIGNMENT I

Asst. Prof. Berna Kiraz

Res. Asst. Zeki Kuş

Hoare's Partitioning and Lomuto's Partitioning in Quicksort Algorithm.

Kevser Büşra YILDIRIM

1821221029 – Computer Engineering

November 2020

Introduction

In this programming assignment, I designed an experimental study for the comparison between Hoare’s partitioning and Lomuto’s partitioning in Quicksort algorithm. I designed the experiments for the comparison of two algorithms both theoretically and empirically.

Implementation

I implemented parts of Hoare' and Lomuto’s partitioning as recursive functions in Java programming language. Implementations are in Java source code file.

```java
public static int LomutoPartition(int[] b, int low, int high) {
    int pivot = b[high];
    int i = low;
    for (int j = low; j < high; j++) {
        if (b[j] <= pivot) {
            swap(b, i, j);
            i++;
        }
    }
    swap(b, i, high);

    return i;
}

public static void LomutosQuickSort(int[] b, int low, int high) {

    if (low < high) {
        int p = LomutoPartition(b, low, high);
        LomutosQuickSort(b, 0, p - 1);
        LomutosQuickSort(b, p + 1, high);
    }

}
```

```java
public static int HoarePartition(int[] a, int low, int high) {
    int pivot = a[low];
    int i = low - 1;
    int j = high + 1;

    while (true) {
        do {
            i++;
        } while (a[i] < pivot);

        do {
            j--;
        } while (a[j] > pivot);

        if (i >= j) {
            return j;
        }

        swap(a, i, j);
    }
}

public static void quicksortHoares(int[] a, int low, int high) {
    if (low >= high) {
        return;
    }
    int pivot = HoarePartition(a, low, high);
    //elements less than the pivot
    quicksortHoares(a, low, pivot);
    //elements more than the pivot
    quicksortHoares(a, pivot + 1, high);
}
```

The pseudocode of the codes is as follows:

- **Pseudocode of Hoare’s partitioning:**

  HoarePartition (a[], low, high)

  pivot = a[low]
  i = low - 1
  j = high + 1

  repat i = i + 1 until a[i] < pivot

  repat j-- until (a[j] > pivot);

  if i >= j then
    return j

  swap a[i] with a[j]

```
quicksortHoares(int[] a, int low, int high)
    if low >= high then
        return;
    pivot = HoarePartition(a, low, high)
    //elements less than the pivot
    quicksortHoares(a, low, pivot)
    //elements more than the pivot
    quicksortHoares(a, pivot + 1, high)
```

- **Pseudocode of Lomuto's partitioning:**

```
LomutoPartition (b[], low, high)
  pivot = b[high];
  i =low;

  for j=low; j<high; j++

    if b[j] <= pivot

        swap b[i] with b[j]
        i++;

  swap b[i] with b[high]

  return i;

LomutosQuickSort(int[] b, int low, int high)

    if low < high then
        p = LomutoPartition(b, low, high)
        LomutosQuickSort(b, 0, p - 1)
        LomutosQuickSort(b, p + 1, high)
```

Time Complexity of Hoare's and Lomuto's Partitions

The runtime of quicksort depends on whether the shredding is balanced or unbalanced. If partitioning is balanced, the algorithm runs fast by asymptotically sorting. However, if partitioning is unbalanced, it runs asymptotically as slow as insertion sorting algorithm.

- **Hoare's Partition Time Complexity:**

Basic Operation: Comparison
Input Size: n
Time Complexity: $3n+4 \in O(n)$
Each n gives us the number of unembedded loops, i.e., do-while loops, in the algorithm. Constants are obtained from simple mathematical operations.

- **quicksortHoares method's time complexity:**

  Best Case:

  The best case is when split input array in the middle.

  T(n) = 2.T(n/2) + (n+1)

  a=2, b=2, d=1 => a=b^d according to Master Theorem T(n) ∈ O (n $\log n$)

  Worst Case:

  The worst case is when the input array is already sorted before entering the quicksort algorithm. When the input array is already sorted, time complexity degrades to Θ (n^2) of Hoare's partition scheme.

  T(n) = (n-1) + n

  T(n) ∈ $O(n^2)$

- **Lomuto's Partition Time Complexity:**

  Basic Operation: Comparison

  Input Size: n

  Time Complexity: n+5 ∈ O(n)

- **LomutosQuickSort method's time complexity:**

  Best and Worst Case:

  A[p…r-1] > pivot

  T(n) = T(n-1)+n

  T(n) ∈ $O(n^2)$

Result and Discussion

The results indicate that, Lomuto's partitioning does the work in just one array traversal like as Hoare's partition, but Lomuto partition requires more swaps. Lomuto's partition puts the pivot at the correct position in the array as well as returns the index whereas Hoare's partition only returns the correct index of the pivot. Lomuto partition is more relatively inefficient in time complexity respect. This situation to cause Lomuto's partition slower than Hoare's partition. Both partitions are linear algorithms.

Upon exploring the situation from multiple perspectives, we can say that, while Hoare's partition algorithm is slightly difficult to understand and to implement, Lomuto's partition algorithm easier to understand and implement. based on the results of this study, it seems some factors causes the Hoare's partitioning algorithm to be preferred.

	Faculty of Engineering Computer Engineering Program **BLM19307E Algorithm Analysis & Design Lab Work**	Name Surname: Grade:

Hoare’s Partitioning and Lomuto’s Partitioning in Quicksort Algorithm.

<u>Question 1:</u> Below, Hoare’s Partitioning and Lomuto’s Partitioning (in Quicksort Algorithm) are given.

```
public static int LomutoPartition(int[] b, int low, int high) {
    int pivot = b[high];
    int i = low;
    for (int j = low; j < high; j++) {
        if (b[j] <= pivot) {
            swap(b, i, j);
            i++;
        }
    }
    swap(b, i, high);

    return i;
}

public static void LomutosQuickSort(int[] b, int low, int high) {

    if (low < high) {
        int p = LomutoPartition(b, low, high);
        LomutosQuickSort(b, 0, p - 1);
        LomutosQuickSort(b, p + 1, high);
    }

}
```

```
public static int HoarePartition(int[] a, int low, int high) {
    int pivot = a[low];
    int i = low - 1;
    int j = high + 1;

    while (true) {
        do {
            i++;
        } while (a[i] < pivot);

        do {
            j--;
        } while (a[j] > pivot);

        if (i >= j) {
            return j;
        }

        swap(a, i, j);
    }
}

public static void quicksortHoares(int[] a, int low, int high) {
    if (low >= high) {
        return;
    }
    int pivot = HoarePartition(a, low, high);
    //elements less than the pivot
    quicksortHoares(a, low, pivot);
    //elements more than the pivot
    quicksortHoares(a, pivot + 1, high);
}
```

Explain which algorithm can run faster when we run the algorithms. (15p)

<u>Question 2:</u> Make a comparison between the Hoare’s and Lomuto’s partitioning algorithms. What do they have in common? (10p)

What are their differences? (10p)

Explain which one you would prefer to use. (10p)

<u>Question 3:</u> We know that the runtime of quicksort depends on the elements used for partitioning. According to this situation, what are the factor or factors that change the working speed? Please explain. (20p)

The runtime of quicksort depends on whether the shredding is balanced or unbalanced. If partitioning is balanced, the algorithm runs fast by asymptotically sorting. However, if partitioning is unbalanced, it runs asymptotically as slow as insertion sorting algorithm.

Question 4: If we want to calculate the time complexity of algorithms, which algorithm analysis method should we use? (5p)

Calculate the time complexities for both. (30p)