Lab 8 Quick sort

Weichen Chai

Fall 2022

Introduction

This lab report seeks to answer the questions presented in ”Quick sort an array and a linked list” for ID1021. In this assignment, the goal is to analyse the advantage of disadvantages of the quick sort technique, and to implement quick sort for both an array and a linked list.

Background

We have worked with sorting techniques in previous assignments, the one we will focus on for this assignment is called quick sort, which works in a similar manner to merge sort, as it divides data into two parts, and sort the parts separately. The main difference quick sort has in comparison to merge sort is that it is able to be more efficient when joining the two sets of sorted data, however, will be less efficient when dividing the data.

Tasks

Quick sort Array

In a quick sort array, we must first select a pivot, through comparing with the pivot, we move the smaller elements to the left, and bigger elements to the right of the pivot.

```
 static void Qsort(int[] arr, int low, int high)
    {
        if (low < high) {
   // find the pivot point
            int pivot = partition(arr, low, high);

//lower partition sorting
            Qsort(arr, low, pivot - 1);
//larger partition sorting
```

```
            Qsort(arr, pivot + 1, high);
        }
    }
```

Partition selects and places the pivot element into the position that will partition the array. The element at the start of the array is selected to be a pivot, and the numbers after the pivot elements will be sorted so that the elements smaller than the pivot will be placed to the right side, switched by identifying an i and j, where i starts at the start of the sequence and j starts at the end of the sequence. When numbers are smaller or equal to the pivot, i is incremented, and when items are bigger than the pivot, j is decremented. Finally, when i is positioned at a larger element than j, then proceed by swapping the pivot element to position j.

```
    static int partition(int[] a, int low, int high) {
        int i = low;
        int j = high;
        int pivot = a[low];
        while (i < j) {

            while (a[i] <= pivot && i < j) {
                i++;
            }
            while (a[j] > pivot) {
                j--;
            }

            if (i < j) {

                int temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
        int temp = a[low];
        a[low] = a[j];
        a[j] = temp;
        return j;
    }
}
```

Quick sort Linked List

In the second task, we are asked to implement quick sort in a linked list format, this can be achieved through selecting and placing a pivot in the partition function, and traverse through the nodes to sort them into two different lists, containing smaller or larger nodes in comparison to the pivot. Where then they will be sorted in each partition by changing their references, after the sorting occurs, the pivot will be appended to the tail position of the linked list containing smaller nodes, and at last the list containing values larger than the pivot will be appended on top of that. This is the method we will be going with to receive fully sorted list. In order to keep a tail, we can make a public static node, which indicates the tail.

First we begin by identifying two lists that will contain nodes smaller or larger than the pivot.

```
        Quicklist low = new Quicklist();
        //linkedlist that will be larger than pivot element
        Quicklist high = new Quicklist();
```

Then we identify the first element of a linked list to be the pivot, traverse through the entire list and place values into lower or higher partition in relation to the pivot.

```
public static Node Qlast;
...
            Node pivot = a.head;
            Node current = pivot.next;  // a current pointer that is one after the
            Node right = a.tail
            while(current != right.next) { // iterates through the entire list
                Node a = current.next;
                current= null;
                if(current.val < pivot.val) {
                    low.add(current);
                }else{
                    high.add(current);
.
.
.
```

Benchmark

The time complexity for quick sort is estimated to be O(nlogn), as it follows the basic fundamental principle as merge sort, where we first divide the elements into parts, sort them, and place them back together, which we have dealt with before. The worst case scenario in this case, is when the pivot point is either the largest or smallest element, making the time complexity O(n2), these cases remain true for both array and linked list quick sorts. The results are shown below, they are run 10000 times and averaged to receive the final result.

Size	**Linked List**	**Arrays**
100	1710	1512
200	4082	2583
300	6518	5821
400	9143	7823
500	13021	9832
1000	33927	25892
1500	56281	45209

As presented above, the time complexity of the two variations of quick sort does indeed share similarities with the hypothesis stated before, supporting the claim that both their time complexities are O(nlogn). However, it is noticeable that the time taken for arrays is quick a bit less than that of the linked list. This can be assumed to be due to the fact that linked lists requires to work with the change of reference and append, whilst arrays will mainly take time to change the actual values inside, making it much faster.