Quick Sort

Fredrik Berzins

Fall 2023

Introduction

Quick sort is as it's name suggests a quick sorting algorithm. In this report differences for quick sorting an array and linked list will be compared, this will be done with the help of some benchmarks. Quick sort is a unstable sorting method that is good for parallelization, with Big o of O($n * logn$) as best and average case but worst case of O(n^2).

Quick sort explanation

Quick sort is a recursive sorting algorithm, witch in principle is simple, it has some complexities when programming. The quick explanation for how it works is:

1. the an element in the list, this is the pivot.
2. swap elements starting from each end of the list until you get to the center, only if the larger end up on the right side.
3. move the pivot to the center, this means all element on the left are smaller and right are larger.
4. Now do this over again for each unsorted half.

Quick sort is reliant on recursive functions this makes it some times harder to bug fix and harder to grasp witch is why its simple in practice but harder to implement. It is also not stable witch might be a down side in some circumstances.

Implementation

Quick sort for array implementation

Quick sort for arrays was implemented with 2 methods one to select the pivot, find what to swap and move pivot to the correct spot, this method will be shown below. The second method was a swap function that just swapped two values.

```java
public static void QuickSortArr(int[] arr, int loIndx, int hiIndx) {
    if (loIndx < hiIndx) {
        int pivotValue = arr[hiIndx];
        int pivot = loIndx - 1;
        // mian
        for (int j = loIndx; j < hiIndx; j++) {
            if (arr[j] <= pivotValue) {
                pivot++;
                swap(arr, pivot, j);
            }
        }
        // end of main
        swap(arr, pivot, hiIndx);
        QuickSortArr(arr, loIndx, (pivot-1));
        QuickSortArr(arr, (pivot+1), hiIndx);
    }
}

public static void swap(int[] arr, int a, int b) {
    int temp = arr[a];
    arr[a] = arr[b];
    arr[b] = temp;
}
```

Quick sort for linked list implementation

The implementation for linked list is a bit different since it makes new linked list for upper and lower half for ever recursive step. This makes it slightly more complicated and slower since it moves nodes into an linked list. This is shown in the code below witch is the equivalent of the code block for arrays just for linked list.

```java
while (curr != null) {
            Quick_Node nxt = curr.next;
            if (curr.value <= pivot.value) {
                smaller.add(curr);
            }
            else {
                larger.add(curr);
            }
            curr = nxt;
        }
```

Compare Array and linked list implementation

The two code blocks above clearly show how arrays are simpler to work with since swap only swaps the values instead of add witch adds a node to an existing linked list, this involves re-referencing the lists first pointer and re-referencing the nodes next pointer. In the code its also clear how the pointers makes it more complicated to work with compared to index of an array since you remove the original pointer when adding it to a new list.

Benchmarks

The testing was done with different array/list length between 100 and 6400 values(doubling each time, 100,200,400...). every 1'000 time tests was redone 1'000 time, this gives a low time showing avg from the group with the 1'000 quickest runs.(Average of 1'000 minimum times.)

Result

As the results in the table below shows both Quick sort for array and linked list have a time complexity of O($n * logn$). It also shows that it's about half the time to quick sort the array compared to the linked list, this is as predicted from the implementation. If the implementation for linked list would be more like the array where the values are swapped, the time is comparable with minor difference since it at the point would be used like a complex array.

n	**Array**	**Linked List**
100	6873	13110
200	15245	29623
400	35049	66399
800	77614	146489
1600	170018	325816
3200	369579	731225
3200	798941	1627355

Table 1: time to sort array/list of length n with quick sort in ns

Conclusion

In conclusion Quick sort is a can be a useful sorting algorithm but it has some competition from merge sort witch doesn't have a case worse then O($n * logn$) compared to quick sorts O(n^2). It's also not stable compared to merge so the only real reason to use quick sort over merge sort is the memory complexity of O(n) for merge sort, since Quick sorts is O$(logn)$. If memory was a concerns both heap and block sort would be better picks both also have better time complexity.