Recall Merge Sort

Short comings of Merge sort?

Merge Sort - Shortcomings

- Merging L and R arrays create a new array
- No obvious way to efficiently merge *in place*
- Extra storage is required to merge and can be costly
- Merging happens because elements in left half must move right and vice versa

Motivation – Another Sorting algo / Quick Sort

- Can we divide so that everything to the left is smaller than everything to the right

- No need to Merge

Without Merging

- Suppose the median value in Array A is m

- Move all values <= m to the left half of A

- Move all values > m to the right half of A

- How much time do you need to do this shifting?

Without Merging

- Recursively sort left and right halves
- Array A is now sorted
- No need to merge

Questions

- How do we find the median?
- Sort and pick the middle element
- But, our goal is to sort
- Instead, pick up some value in A i.e pivot
- Split A w.r.t the pivot element

Quick Sort Algorithm

Quick Sort - Introduction

- Tony Hoare - Early 1960’s
- Well Known Computer Scientist
- Turing Award Winner

Quick sort - Idea

- Choose a pivot element
- Typically the first/last value in the array is pivot
- Divide/Partition the array A into lower and upper parts w.r.t pivot
- Move pivot between lower and upper partition
- Recursively sort the two partitions

Quick Sort : Divide and Conquer

- **Divide :**
  - Partition the array A[p..r] to two sub arrays A[p..q-1] and A[q+1..r], where q is computed as part of the *PARTITION* function
  - Each element of A[p..q-1] is less than or equal to A[q]
  - Each element of A[q+1..r] is greater than A[q]
  - The sub arrays can be empty or non-empty

Quick Sort : Divide and Conquer

- **Conquer :**
  - Sort the two sub arrays A[p..q-1] and A[q+1..r]
  - By recursive calls to quick sort

Quick Sort : A Divide and Conquer strategy

- **Combine :**
  - The two sub arrays are already sorted
  - The entire array A[p ..r] is already sorted
  - Nothing particular to do in *combine* step

Pseudo code of Quick Sort

QUICKSORT(A, p, r)

```
if p < r
    then q ← PARTITION(A, p, r)
         QUICKSORT(A, p, q-1)
         QUICKSORT(A, q+1, r)
```

To sort an entire array the initial call is

QUICKSORT(A, 1, A.length)

How do we partition an array ?

- Suppose array A = [9, 7, 5, 11, 12, 2, 14, 3, 10, 6]

- Array entry A[r] is selected as the pivot element, where r is the index of the last element of the array

- A[r] is compared with the array elements until a smaller element *s* (less than or equal to pivot) is obtained

- If *s* is obtained, it is exchanged with the first element of the array initially

- A[r] is again compared with all other elements of A and exchange of smaller element is done further

Working of *PARTITION*

- A = [9, 7, 5, 11, 12, 2, 14, 3, 10, 6]
- Pivot = A[r] = 6
- 6 is compared with the elements of A until a smaller element (than 6) is obtained
- In this case, 6 is compared with 9,7 and 5
- Exchange 9 with 5
- A = [5, 7, 9, 11, 12, 2, 14, 3, 10, 6]

Working of *PARTITION*

- A = [5, 7, 9, 11, 12, 2, 14, 3, 10, 6]
- Again 6 is compared with 11,12 and 2
- 7 and 2 are exchanged
- A= [5, 2, 9, 11, 12, 7, 14, 3, 10, 6]
- Again 6 is compared with 14 and 3
- 9 and 3 are exchanged
- A= [5, 2, 3, 11, 12, 7, 14, 9, 10, 6]

Working of *PARTITION*

- A= [5, 2, 3, 11, 12, 7, 14, 9, 10, 6]
- Again 6 is compared with 10
- We can see that 5, 2 and 3 are lesser or equal to than the pivot and we have compared pivot with all other elements of A
- Exchange pivot with 11 so that all the elements before pivot is less than or equal to pivot and all the elements after pivot is greater than pivot
- A= [5, 2, 3, 6, 12, 7, 14, 9, 10, 11]
- [5, 2, 3] and [12, 7, 14, 9, 10, 11] are two partitions separated by the pivot 6

Detailed working of *PARTITION*

```
  p                                          r
 9, 7, 5, 11, 12, 2, 14, 3, 10, 6
```

```
  p                                          r
 9, 7, 5, 11, 12, 2, 14, 3, 10, 6
i j
```

```
  p                                          r
 9, 7, 5, 11, 12, 2, 14, 3, 10, 6
i    j
```

```
  p                                          r
 9, 7, 5, 11, 12, 2, 14, 3, 10, 6
i        j
```

p r

9, 7, 5, 11, 12, 2, 14, 3, 10, 6

i j

5 , 7, 9, 11, 12, 2, 14, 3, 10, 6

i j

r

5 , 7, 9, 11, 12, 2, 14, 3, 10, 6

i j

r

5 , 7, 9, 11, 12, 2, 14, 3, 10, 6

i j

r

5 , 7, 9, 11, 12, 2, 14, 3, 10, 6

i j

r

5 , 7, 9, 11, 12, 2, 14, 3, 10, 6

i j

r

5 ,2 , 9, 11, 12, 7, 14, 3, 10, 6

i j

r

5 ,2 , 9, 11, 12, 7, 14, 3, 10, 6

i j

r

5 ,2 , 9, 11, 12, 7, 14, 3, 10, 6

i j

r

5 ,2 , 9, 11, 12, 7, 14, 3, 10, 6

i j

r

5 ,2 , 3, 11, 12, 7, 14, 9, 10, 6

i j

r

5 ,2 , 3, 11, 12, 7, 14, 9, 10, 6

i j

r

5 ,2 , 3, 6 , 12, 7, 14, 9, 10, 11

i j

5 ,2 , 3, 6 , 12, 7 , 14, 9, 10, 11

5, 2, 3, 6, 12, 7, 14, 9, 10, 11

- [5, 2, 3] and [12, 7, 14, 9, 10, 11] are two partitions separated by the pivot 6
- All the elements before pivot is less than or equal to pivot and all the elements after pivot is greater than pivot

Design of *PARTITION*

- How many counters/pointers do we need?
  - One counter which goes from p to r-1 : to compare all other elements of A with pivot
  - Another counter which keeps track of the position of the smaller element (less than or equal to pivot)

Pseudo code : *PARTITION*

```
PARTITION (A, p, r)
x ← A[r]
i ← p - 1
for j = p to r - 1
    do if A[j] <= x
        then i=i+1
            Exchange A[i] with A[j]
Exchange A[i +1] with A[r]
return i +1
```

Pseudo code of Quick Sort

```
QUICKSORT(A, p, r)

if p < r
     then q ← PARTITION(A, p, r)
          QUICKSORT(A, p, q-1)
          QUICKSORT(A, q+1, r)
```

Exercise

- Trace the working of *PARTITION* with the sub array [5,2,3]
- Trace the working of *PARTITION* with the sub array [12, 7, 14, 9, 10, 11]
- Trace the working of Quick sort with the array [2 4 6 7 9 23]
- Trace the working of Quick sort with the array [26 24 15 7 3 2]

Thank You