Design and Analysis of Algorithm

Lecture-6:

Contents

Complexity

$$T(n) = \begin{cases} b & if\ n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & otherwise \end{cases}$$

$$T(n) = O(nlog_2n)$$

Note:

1. Merge sort is good for large sized array
2. It is not in-place sorting.
3. It is the stable sorting algorithm

Quick sort

In merge sort, the array $a[1:n]$ was divided at its midpoint into sub arrays which were independently sorted and later merged.
In quick sort, the division into two sub arrays is made so that the sorted sub arrays do not need to be merged later.

This is accomplished by rearranging the elements in $a[1:n]$ such that $a[i] < a[j]$

for all i between 1 and m-1

and all j between $m+1$ and n for some m, $1 < m < n$.

Thus, the elements in $a[1:m-1]$ and $a[m+1:n]$ can be independently sorted. No merge is needed

Quick sort

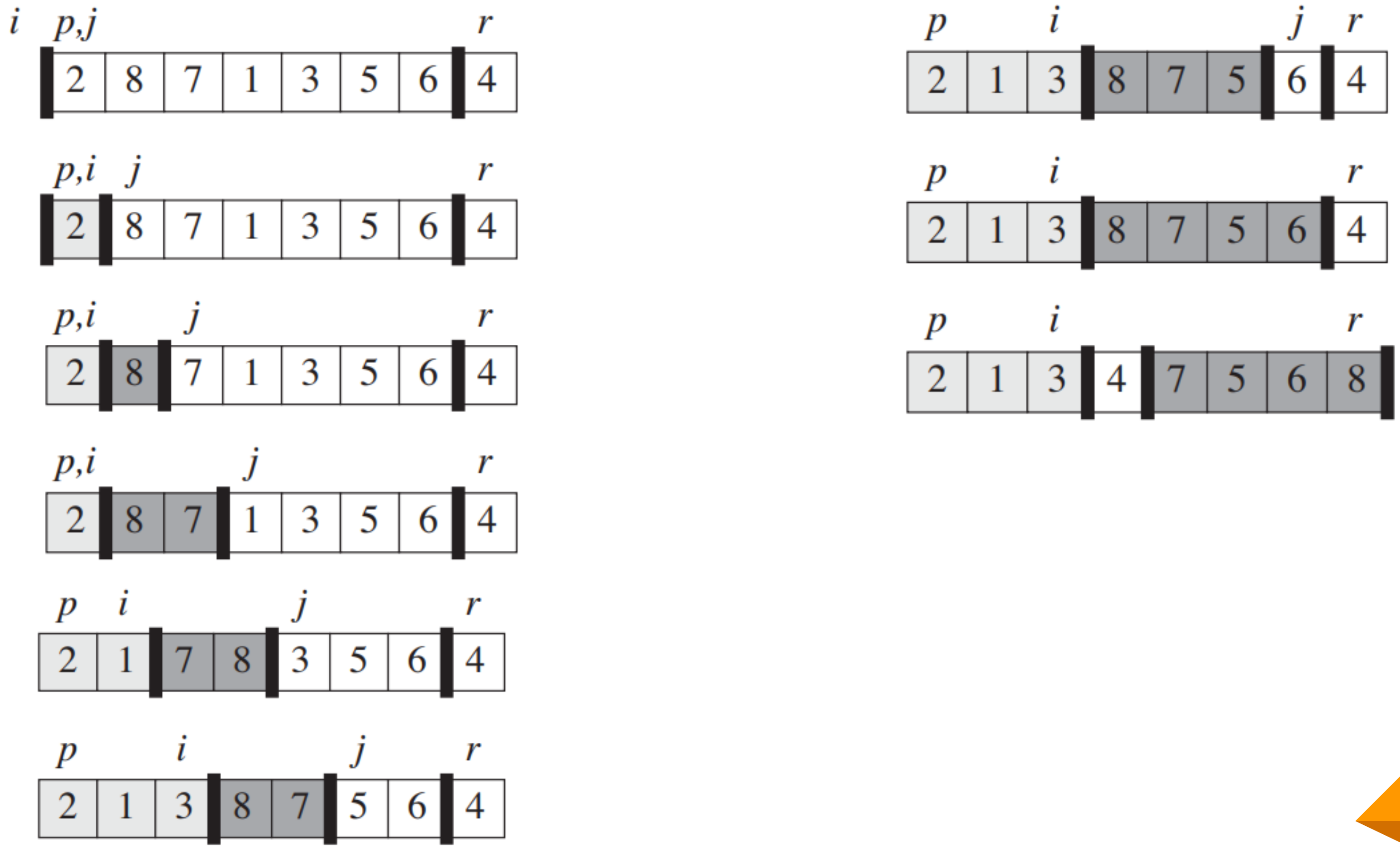
```
Algorithm QUICKSORT(A, p, r)
// Sorts the elements a[p], ..., a[q] which reside in the global
// array a[1 : n] into ascending order; a[n + 1] is considered to
// be defined and must be ≥ all the elements in a[1 : n].
{
    if (p < q) then  // If there are more than one element
    {
        // divide P into two subproblems.
            q = PARTITION(A, p, r)
                // j is the position of the partitioning element.
        // Solve the subproblems.
            QUICKSORT(A, p, q − 1)
            QUICKSORT(A, q + 1, r)
        // There is no need for combining solutions.
    }
}
```

Quick sort

```
Partition (A,p,r)
    x = A[r]
    i=p-1
    for j = p to r-1
    {
      if A[j] ≤ x
      {
          i = i+1
          exchange A[i] with A[j]
      }
    }
    exchange A[i+1 with A[r]
    return i+1
```

i	p,j							r
	2	8	7	1	3	5	6	4

Quick sort Example



Quick sort Example

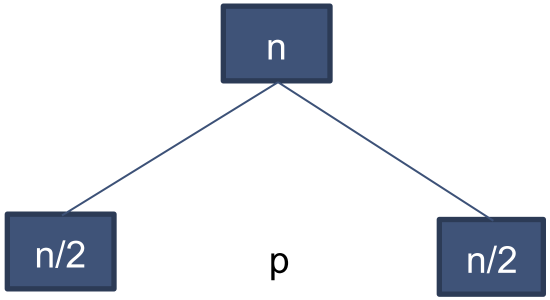
9	7	5	11	12	2	14	3	10	6

5	2	3	6	12	7	14	9	10	11

Complexity of Quick sort

Let $t(n)$ be the time required to sort the given array using quick sort

Best Case



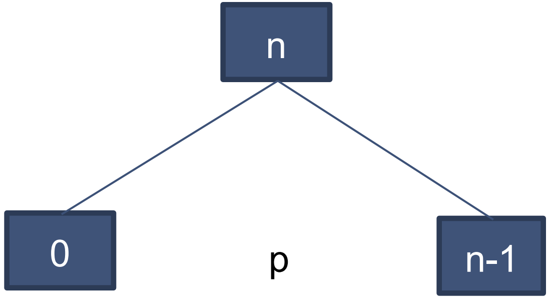
$$T(n) = \begin{cases} c & if\ n = 1 \\ 2T\left(\frac{n}{2}\right) + n & otherwise \end{cases}$$

$$T(n) = nlog_2 n$$

Complexity of Quick sort

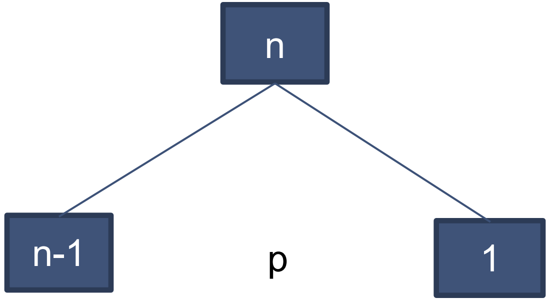
Let $t(n)$ be the time required to sort the given array using quick sort

<u>Worst Case</u>



$$T(n) = \begin{cases} c & if\ n = 1 \\ T(n-1) + n & otherwise \end{cases}$$

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



Average Case

This case occurs when the partitioning of the array is mixture of best case and worst case.

Consider for the average case the partitioning is alternatively in best case and worst case like:

$$B, W, B, W, B, W, B, W, B, W, B, W \ldots\ldots.$$

$$B(n) = W\left(\frac{n}{2}\right) + W\left(\frac{n}{2}\right) + n \qquad \Rightarrow B(n) = 2W\left(\frac{n}{2}\right) + n$$

$$W(n) = B(0) + B(n-1) + n \qquad \Rightarrow W(n) = B(n-1) + n$$

Complexity of Quick sort

Average Case

$$B(n) = 2W\left(\frac{n}{2}\right) + n$$

Since $W(n) = B(n-1) + n$, so $W(n/2) = B((n-1)/2) + n/2$

$$B(n) = 2\left[B\left(\frac{n-1}{2}\right) + \frac{n}{2}\right] + \mathrm{n}$$

$$B(n) = 2B\left(\frac{n-1}{2}\right) + 2\mathrm{n}$$

Solution: $(O(n * log_2 n))$