MCAC 302: Design and Analysis of Algorithms

Neelima Gupta

ngupta@cs.du.ac.in

September 3, 2020

Analysis of Partition

We will no longer refer to the pseudo-code for analysis. We will do it at abstract level. We know that we don't need to count the statements executed constant number of times. And we don't count the number of times loop control variable changes its value.

Analysis: After every comparison at least one key is in the right place.

Analysis of Partition

We will no longer refer to the pseudo-code for analysis. We will do it at abstract level. We know that we don't need to count the statements executed constant number of times. And we don't count the number of times loop control variable changes its value.

Analysis: After every comparison at least one key is in the right place.
After at most how many comparisons, all the keys will be in their right place?

Analysis of Partition

We will no longer refer to the pseudo-code for analysis. We will do it at abstract level. We know that we don't need to count the statements executed constant number of times. And we don't count the number of times loop control variable changes its value.

Analysis: After every comparison at least one key is in the right place.

After at most how many comparisons, all the keys will be in their right place?

n......Worst Case

Analysis of Partition

We will no longer refer to the pseudo-code for analysis. We will do it at abstract level. We know that we don't need to count the statements executed constant number of times. And we don't count the number of times loop control variable changes its value.

Analysis: After every comparison at least one key is in the right place.
After at most how many comparisons, all the keys will be in their right place?
n......Worst Case
What is the best case for Partition?

Analysis of Partition

We will no longer refer to the pseudo-code for analysis. We will do it at abstract level. We know that we don't need to count the statements executed constant number of times. And we don't count the number of times loop control variable changes its value.

Analysis: After every comparison at least one key is in the right place.
After at most how many comparisons, all the keys will be in their right place?
n......Worst Case
What is the best case for Partition?
It does exactly n (plus minus 1) comparisons in every case.

Analysis of Quick Sort

input : Array: $p, r, A[p \dots r]$
output: Sorted Array: $A[p] \leq A[2] \leq .... \leq A[r]$

QuickSort(A,p,r)
/* Performs sorting on the input array */
if $p < r$ **then**
    q=Partition(A,p,r)
    QuickSort(A,p,q-1)
    QuickSort(A,q+1,r)
end

Let $T(n)$ be the time taken by QuickSort to sort an array containing n elements. Then,

$$T(n) = ?$$

Analysis of Quick Sort

input : Array: $p, r, A[p \dots r]$
output: Sorted Array: $A[p] \leq A[2] \leq .... \leq A[r]$

```
QuickSort(A,p,r)
/* Performs sorting on the input array */
if p < r then
    q=Partition(A,p,r)
    QuickSort(A,p,q-1)
    QuickSort(A,q+1,r)
end
```

Let $T(n)$ be the time taken by QuickSort to sort an array containing n elements. Then,

$$T(n) = T(q-p) + T(r-q) + n$$

Analysis of Quick Sort

input : Array: $p, r, A[p \ldots r]$
output: Sorted Array: $A[p] \leq A[2] \leq .... \leq A[r]$

QuickSort(A,p,r)
/* Performs sorting on the input array */
if $p < r$ **then**
    q=Partition(A,p,r)
    QuickSort(A,p,q-1)
    QuickSort(A,q+1,r)
end

Let $T(n)$ be the time taken by QuickSort to sort an array containing n elements. Then,

$$T(n) = T(n/2) + T(n/2) + n$$

in best case when the partition is almost perfectly balanced

Analysis of Quick Sort

input : Array: $p, r, A[p \dots r]$
output: Sorted Array: $A[p] \leq A[2] \leq .... \leq A[r]$

```
QuickSort(A,p,r)
/* Performs sorting on the input array */
if p < r then
    q=Partition(A,p,r)
    QuickSort(A,p,q-1)
    QuickSort(A,q+1,r)
end
```

Let $T(n)$ be the time taken by QuickSort to sort an array containing n elements. Then,

$$T(n) = T(n/2) + T(n/2) + n = 2T(n/2) + n$$

in best case when the partition is almost perfectly balanced

Analysis of Quick Sort

input : Array: $p, r, A[p \dots r]$
output: Sorted Array: $A[p] \leq A[2] \leq .... \leq A[r]$

```
QuickSort(A,p,r)
/* Performs sorting on the input array */
if p < r then
    q=Partition(A,p,r)
    QuickSort(A,p,q-1)
    QuickSort(A,q+1,r)
end
```

Let $T(n)$ be the time taken by QuickSort to sort an array containing n elements. Then,

$$T(n) = T(n/2) + T(n/2) + n = 2T(n/2) + n = n \log n$$

in best case when the partition is almost perfectly balanced

Analysis of Quick Sort

input : Array: $p, r, A[p \dots r]$
output: Sorted Array: $A[p] \leq A[2] \leq .... \leq A[r]$

```
QuickSort(A,p,r)
/* Performs sorting on the input array */
if p < r then
    q=Partition(A,p,r)
    QuickSort(A,p,q-1)
    QuickSort(A,q+1,r)
end
```

Let $T(n)$ be the time taken by QuickSort to sort an array containing n elements. Then,

$$T(n) = T(n/2) + T(n/2) + n = 2T(n/2) + n = n \log n$$

in best case when the partition is almost perfectly balanced
Note of Caution: This is not a proof that this is the best case...It can be proved using Substitution method...We will not do that here....Self Study from CSLR

Analysis of QuickSort Contd..

And, in the worst case

$$T(n) = T(n-1) + T(0) + n$$

when the partition is completely imbalanced

Analysis of QuickSort Contd..

And, in the worst case

$$T(n) = T(n-1) + T(0) + n$$

$$T(n) = T(n-1) + T(0) + n = \theta(n^2)$$

when the partition is completely imbalanced

Analysis of QuickSort Contd..

And, in the worst case

$$T(n) = T(n-1) + T(0) + n$$

$$T(n) = T(n-1) + T(0) + n = \theta(n^2)$$

when the partition is completely imbalanced
Again this is not a proof....Self Study from CSLR