ESc101 : Fundamental of Computing

I Semester 2008-09

Lecture 38

- Analyzing the number of steps in Quick Sort.
- Why the exact values of constants in $an + c$ are ignored
- Time Complexity of algorithms.
- Examples ...

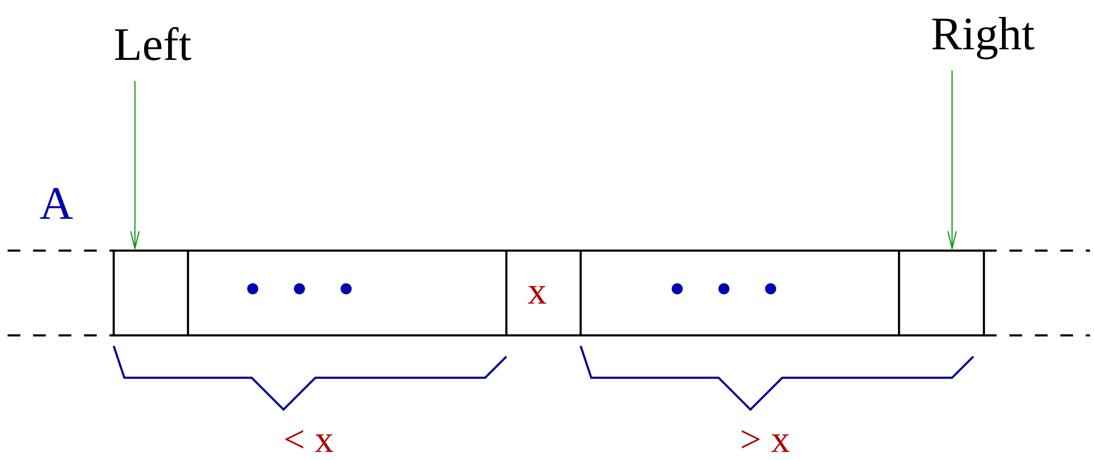
Number of instructions taken in Quick Sort

Quick Sort uses the method *Partition*

```
int Partition(int[] A, int left, right)
```

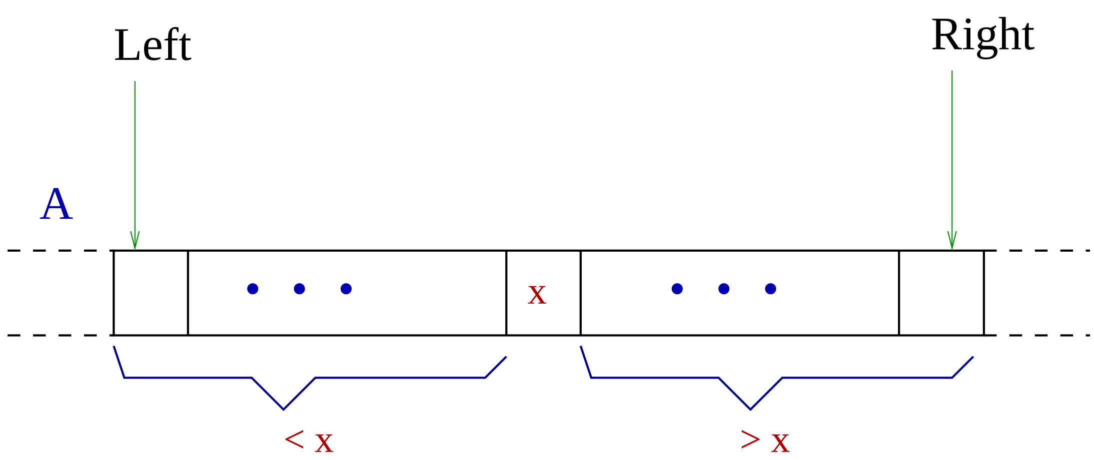
What does Partition(int[] A, int left, right) do ?

For an element `x` $\in$ {`A[left],...,A[right]`}, do rearrangement so that



What does Partition(int[] A, int left, right) do ?

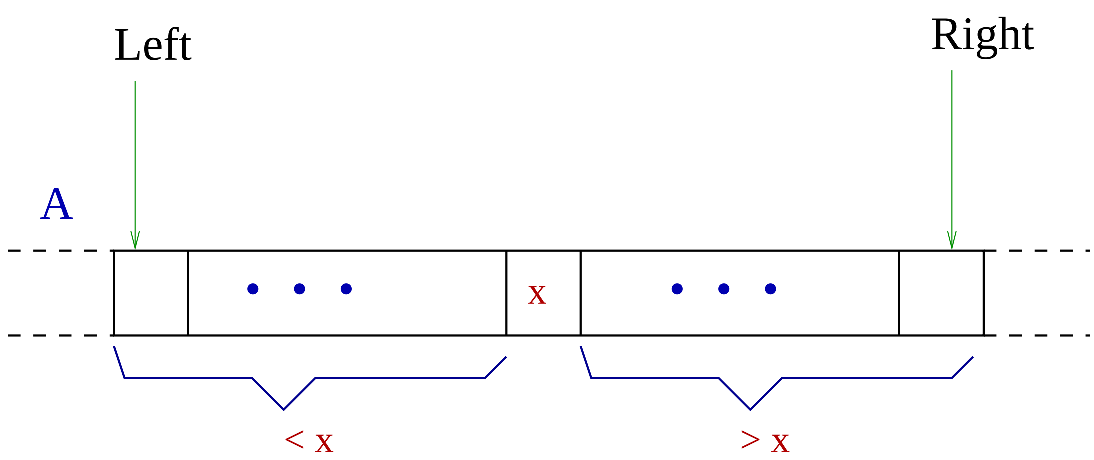
For an element $\mathtt{x} \in \{\mathtt{A[left],...,A[right]}\}$, do rearrangement so that



We used x=A[right]

What does Partition(int[] A, int left, right) do ?

For an element $\mathtt{x} \in \{\mathtt{A[left]}, \ldots, \mathtt{A[right]}\}$, do rearrangement so that



We used x=A[right]

Number of instructions = **c(right - left)**, for some constant **c**

```
public static void Qsort(int[] A, int left, int right)
    {
        if(left<right)
        {
            int mid = partition(A, left, right);
            Qsort(A,left,mid-1);
            Qsort(A,mid+1,right);
        }
    }
```

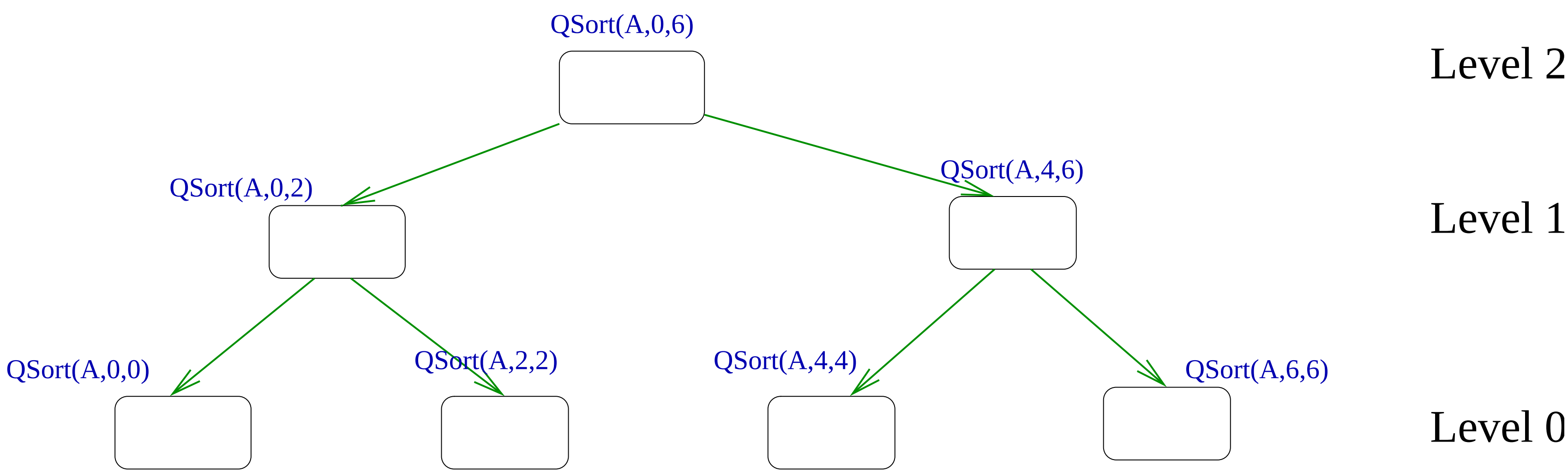
`mid` is the position of pivot element after partition()

What values can `mid` take ?

Number of instructions taken in Quick Sort on n numbers

For array $A = \{1, 9, 5, 21, 40, 29, 13\}$

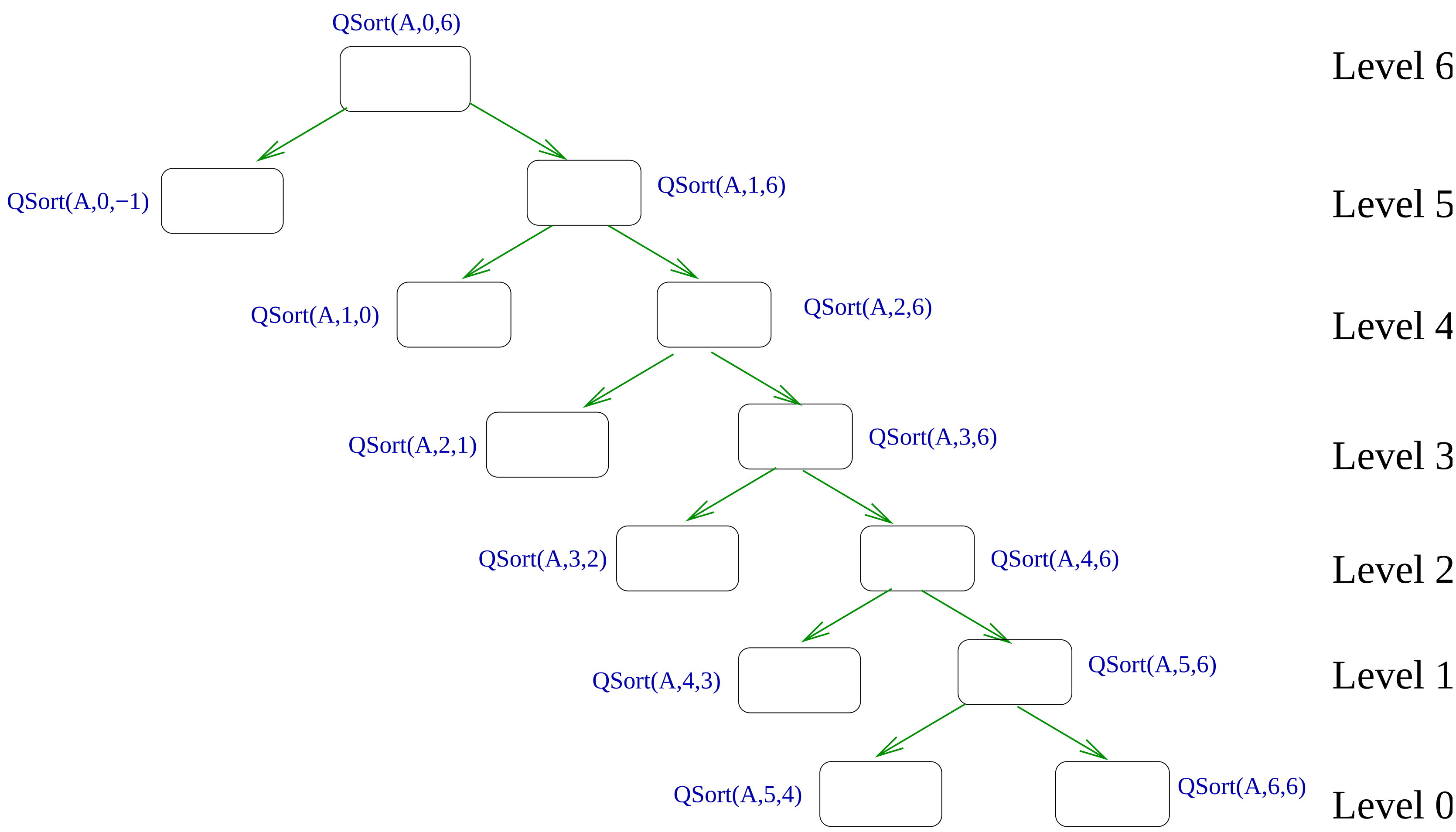
Quick sort on $\{1, 9, 5, 21, 40, 29, 13\}$



Number of instructions taken in Quick Sort on n numbers

For array $A = \{40, 29, 21, 13, 9, 5, 1\}$

Quick sort on $\{40, 29, 21, 13, 9, 5, 1\}$



Number of instructions taken in Quick Sort on n numbers

- If in each recursive call, the pivot element partitions the array into equal half always, then the number of instructions is =
  ????
-

Number of instructions taken in Quick Sort on n numbers

- If in each recursive call, the pivot element partitions the array into equal half always, then the number of instructions is =
  $\mathbf{cn}\log\mathbf{n}$
-

Number of instructions taken in Quick Sort on n numbers

- If in each recursive call, the pivot element partitions the array into equal half always, then the number of instructions is = $\mathbf{cn} \log \mathbf{n}$.
- If in each recursive call, the pivot element is always either the smallest or greatest, then the number of instructions is = ?????.

Number of instructions taken in Quick Sort on n numbers

- If in each recursive call, the pivot element partitions the array into equal half always, then the number of instructions is =
  $\mathbf{cn \log n}$.
- If in each recursive call, the pivot element is always either the smallest or greatest, then the number of instructions is =
  $\mathbf{dn^2}$, for some constant $\mathbf{d}$.

Why is Quick Sort still the most efficient practically ?

1. No overhead of extra array and copying like in Merge sort.
2. The fraction of premutations which correspond to the worst case is very very small.
3. As a consequence, the average number of steps are close to the best case : $O(n \log n)$.

Why is Quick Sort still the most efficient practically ?

1. No overhead of extra array and copying like in Merge sort.
2. The fraction of premutations which correspond to the worst case is very very small.
3. As a consequence, the average number of steps are close to the best case : $O(n \log n)$.

For a formal analysis, do the course
ES0211 : Data structures and Algorithms

Which of the two algorithms would you call faster than the other?

- **Algorithm A** worst case number of instructions : 10n + 200
- **Algorithm B** worst case number of instructions : 10n + 100000

Answer :

Which of the two algorithms would you call faster than the other?

- **Algorithm A** worst case number of instructions : 10n + 200
- **Algorithm B** worst case number of instructions : 10n + 100000

Answer : **Algorithm A**

Which of the two algorithms would you call faster than the other?

- **Algorithm A** worst case number of instructions : $n^2 + 10n + 1000$
- **Algorithm B** worst case number of instructions : $n^2 + n$

Answer :

Which of the two algorithms would you call faster than the other?

- **Algorithm A** worst case number of instructions : $n^2 + 10n + 1000$
- **Algorithm B** worst case number of instructions : $n^2 + n$

Answer : **Algorithm B**

Interesting question on the following slide ...

Which of the two algorithms would you call faster than the other?

- **Algorithm A** worst case number of instructions : $10n + 200$
- **Algorithm B** worst case number of instructions : n^2

Answer : ????

Which of the two algorithms would you call faster than the other?

- **Algorithm A** worst case number of instructions : 10n + 200
- **Algorithm B** worst case number of instructions : n^2

Answer : ????

For $n < 20$, **B** is faster and for $n > 20$, **A** is faster

Realize the following important fact

The number of instructions executed or the time taken by an algorithm becomes an important issue only when the input is very large.

So we should compare the number of instructions of two algorithms for asymptotically large values of input.

Which of the two algorithms would you call faster than the other?

- **Algorithm A** worst case number of instructions : $10n + 200$
- **Algorithm B** worst case number of instructions : n^2

Answer :

Asmptotically **A** is faster than **B**

Time complexity of an algorithm

Definition :

it is a measure of how many steps are executed by an algorithm on a given input **asymptotically**, i.e., for *large* input size.

Worst case time complexity of an algorithm

Definition : Over all inputs of size **n**, what is the worst case number of steps taken by the algorithm ?