Quick Sort

Quick Sort

Problem Definition: Implement Quick Sort algorithm and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n.

.

Objectives of the Experiment:

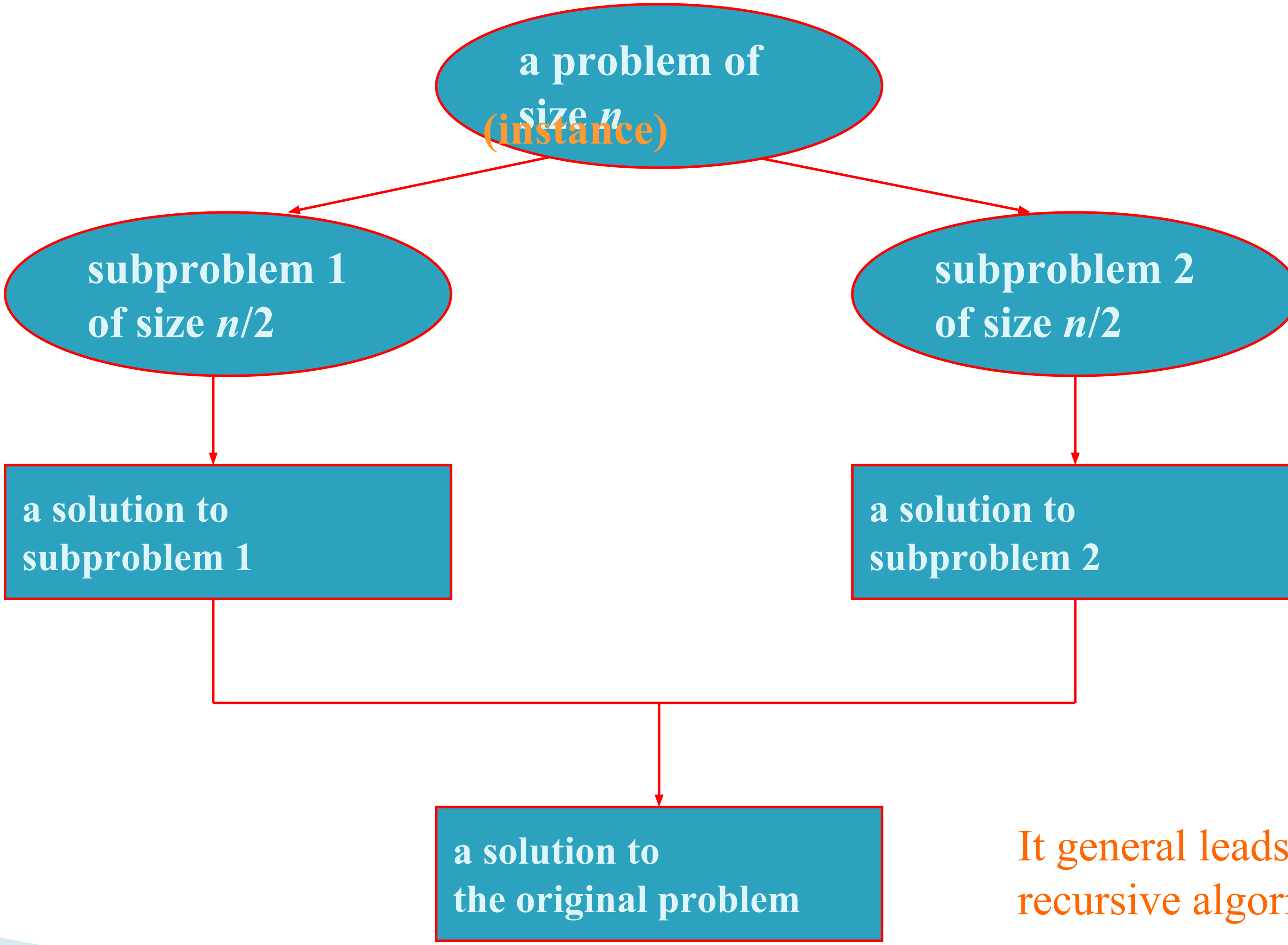
1. To introduce the divide and conquer strategy
2. Present the working of Quick Sort
3. Analyze the Algorithm & Estimate computing time

Divide-and-Conquer

The most-well known algorithm design strategy:

1. Divide instance of problem into two or more smaller instances.
2. Solve smaller instances recursively.
3. Obtain solution to original (larger) instance by combining these solutions.

Divide-and-Conquer Technique (cont.)



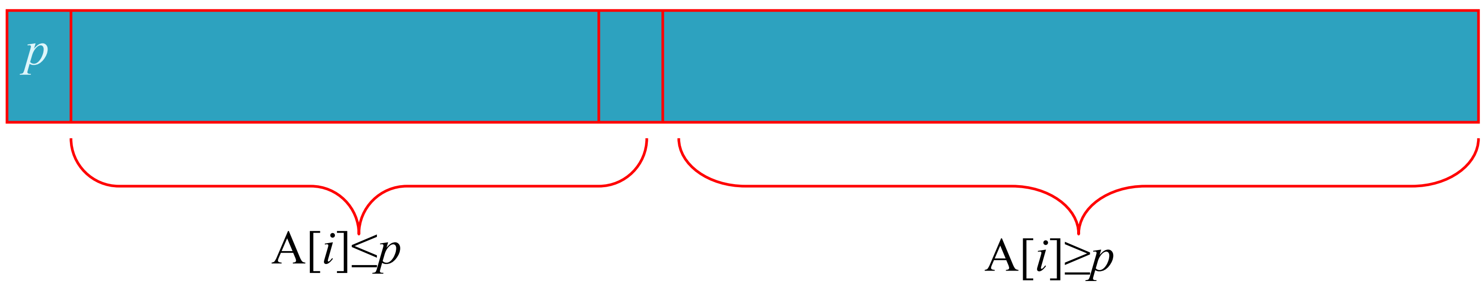
It general leads to a recursive algorithm!

Divide-and-Conquer Examples

- Sorting: mergesort and quicksort
- Binary tree traversals
- Binary search

Quicksort

- Select a *pivot* (partitioning element) – here, the first element
- Rearrange the list so that all the elements in the first *s* positions are smaller than or equal to the pivot and all the elements in the remaining *n-s* positions are larger than or equal to the pivot (see next slide for an algorithm)



Exchange the pivot with the last element in the first (i.e., ≤) subarray — the pivot is now in its final position

- Sort the two subarrays recursively

Quick Sort Algorithm

```
QuickSort(int a[], int low, int high)
{
   long int s;
   if(low<high)
   {
      s=Partition(a,low,high)
      QuickSort (a,low,s-1)
      QuickSort (a,s+1,high)

   }
}
```

Partitioning Algorithm

Algorithm $Partition(A[l..r])$
//Partitions a subarray by using its first element as a pivot
//Input: A subarray $A[l..r]$ of $A[0..n-1]$, defined by its left and right
//   indices l and r $(l < r)$
//Output: A partition of $A[l..r]$, with the split position returned as
//   this function's value
$p \leftarrow A[l]$
$i \leftarrow l$; $j \leftarrow r + 1$
repeat
    repeat $i \leftarrow i + 1$ **until** $A[i] \geq p$ or $i > r$
    repeat $j \leftarrow j - 1$ **until** $A[j] \leq p$ or $j = l$
    swap$(A[i], A[j])$
until $i \geq j$
swap$(A[i], A[j])$ //undo last swap when $i \geq j$
swap$(A[l], A[j])$
return j

Quicksort Example

5 3 1 9 8 2 4 7

2 3 1 4 5 8 9 7

1 2 3 4 5 7 8 9

1 2 3 4 5 7 8 9

1 2 3 4 5 7 8 9

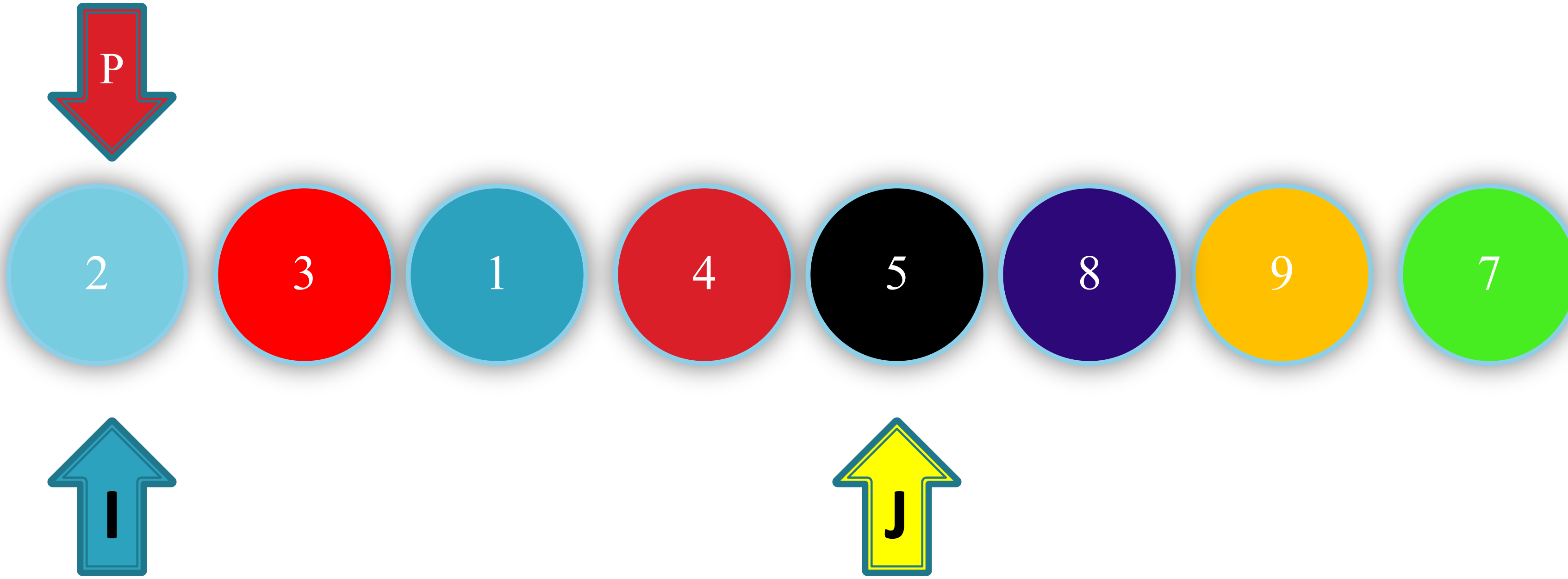
1 2 3 4 5 7 8 9

Undo last swap

i < j ??   Continue

Swap I-th and J-th element

STOP

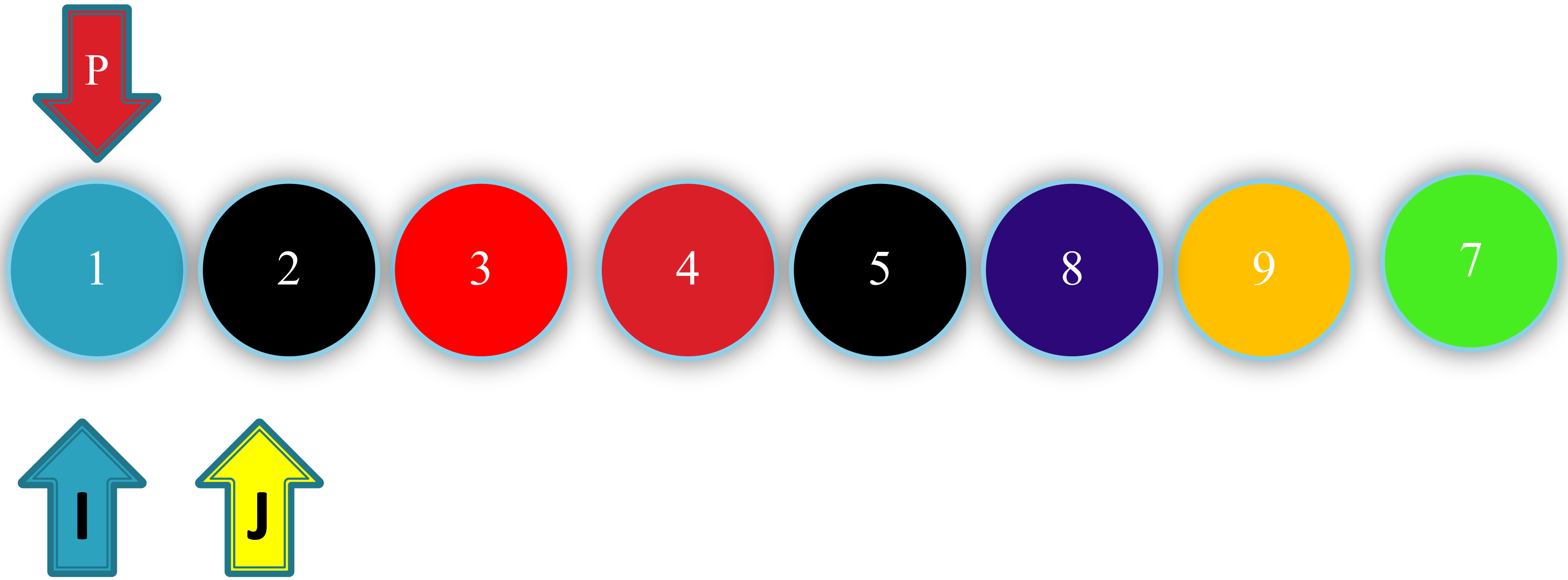


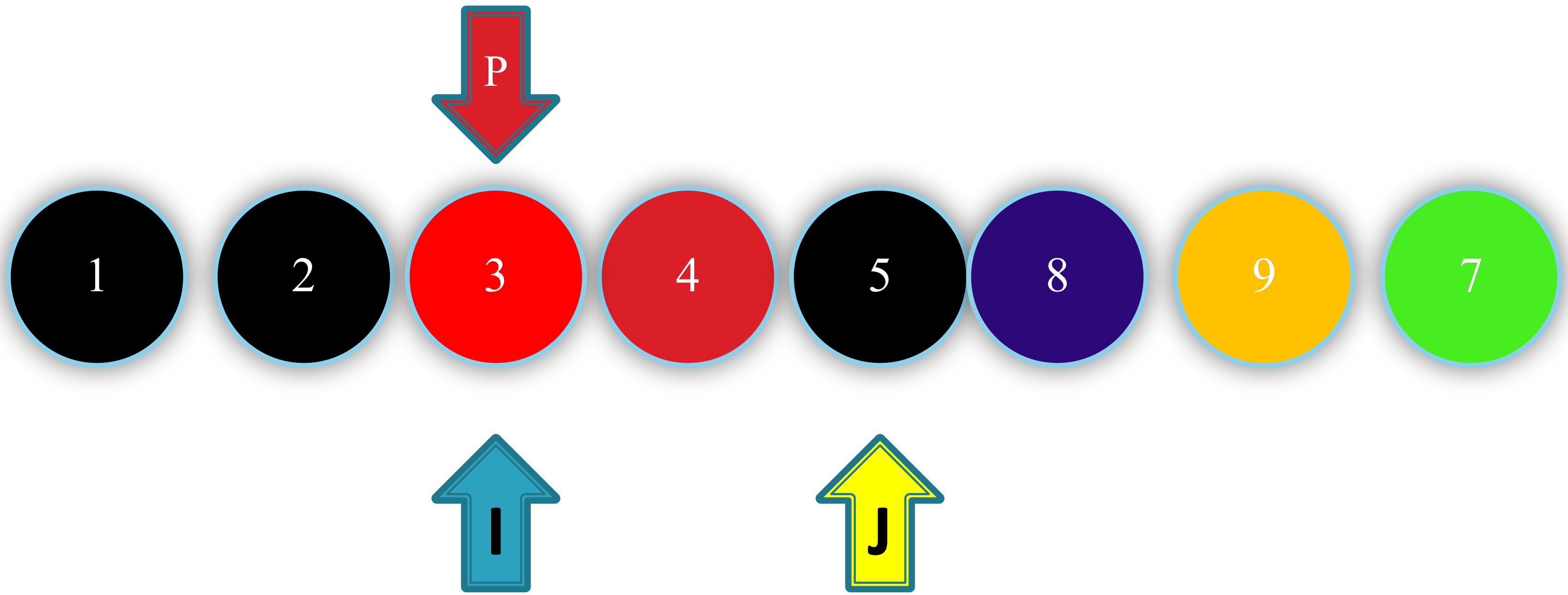
Undo last swap

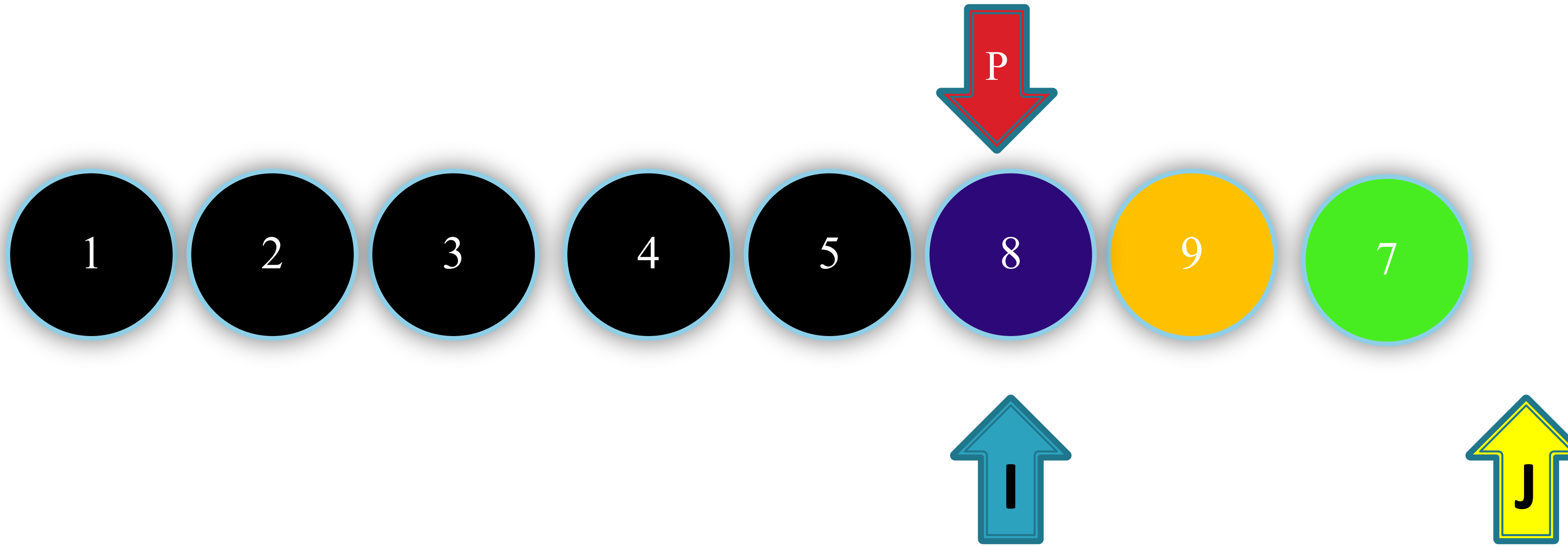
Swap I-th and J-th element

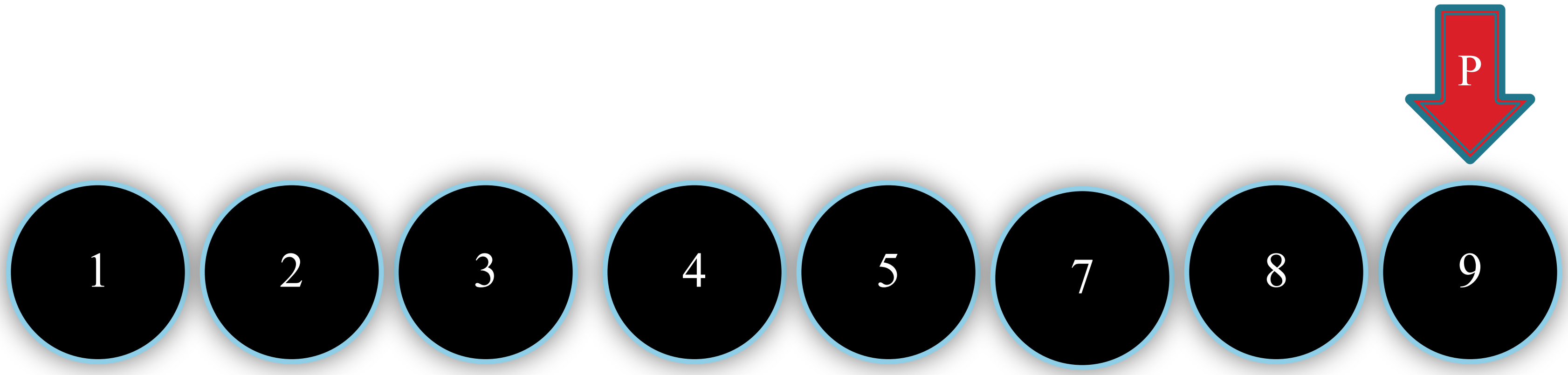
Swap pivot and J-th element

STOP


P
1
2
3
4
5
8
9
7
I
J


P
1
2
3
4
5
8
9
7
I
J


P
1
2
3
4
5
8
9
7
I
J


P
1
2
3
4
5
7
8
9

Analysis of Quicksort

- Best case: split in the middle — $\Theta(n \log n)$
- Worst case: sorted array! — $\Theta(n^2)$
- Average case: random arrays — $\Theta(n \log n)$

$T(n) = T(n-1) + \Theta(n)$

- Improvements:
  - better pivot selection: median of three partitioning
  - switch to insertion sort on small subfiles
  - elimination of recursion

  These combine to 20-25% improvement

- Considered the method of choice for internal sorting of large files ($n \geq 10000$)