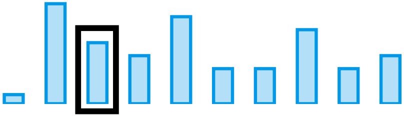
Quick Sort (Divide & Conquer)

(Slides from Alan P. Sexton)

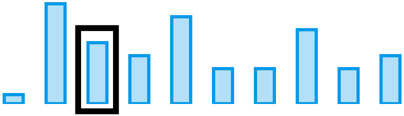
Quick Sort

1. Select an element of the array, which we call the **pivot**.

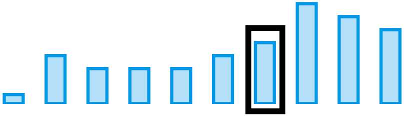


Quick Sort

1. Select an element of the array, which we call the **pivot**.

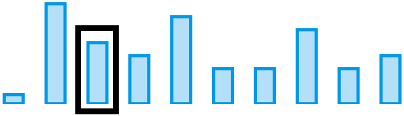


2. Partition the array so that the *"small entries"* ($\leq$ pivot) are on the left, then the pivot, then the *"large entries"* ($>$ pivot).

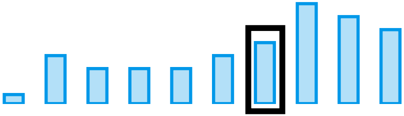


Quick Sort

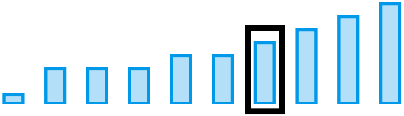
1. Select an element of the array, which we call the **pivot**.



2. Partition the array so that the *"small entries"* ($\leq$ pivot) are on the left, then the pivot, then the *"large entries"* ($>$ pivot).



3. Recursively (quick)sort the two partitions.



Example: Quick Sort run

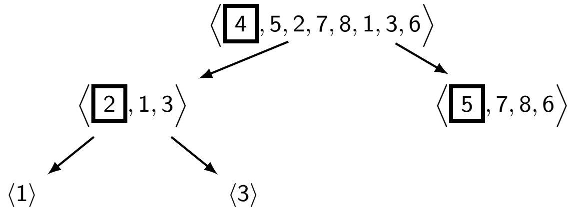
Initial pivot selection strategy: we always choose the leftmost entry.

$$\left\langle \boxed{4}, 5, 2, 7, 8, 1, 3, 6 \right\rangle$$

$$\left\langle \boxed{2}, 1, 3 \right\rangle \qquad\qquad \left\langle \boxed{5}, 7, 8, 6 \right\rangle$$

Example: Quick Sort run

Initial pivot selection strategy: we always choose the leftmost entry.



Example: Quick Sort run

Initial pivot selection strategy: we always choose the leftmost entry.

$$
\begin{tikzcd}
 & & & \left\langle \boxed{4}, 5, 2, 7, 8, 1, 3, 6 \right\rangle \arrow[dll] \arrow[dr] & \\
 & \left\langle \boxed{2}, 1, 3 \right\rangle \arrow[dl] \arrow[dr] & & & \left\langle \boxed{5}, 7, 8, 6 \right\rangle \\
\langle 1 \rangle \arrow[dr] & & \langle 3 \rangle \arrow[dl] & & \\
 & \left\langle 1, \boxed{2}, 3 \right\rangle & & &
\end{tikzcd}
$$

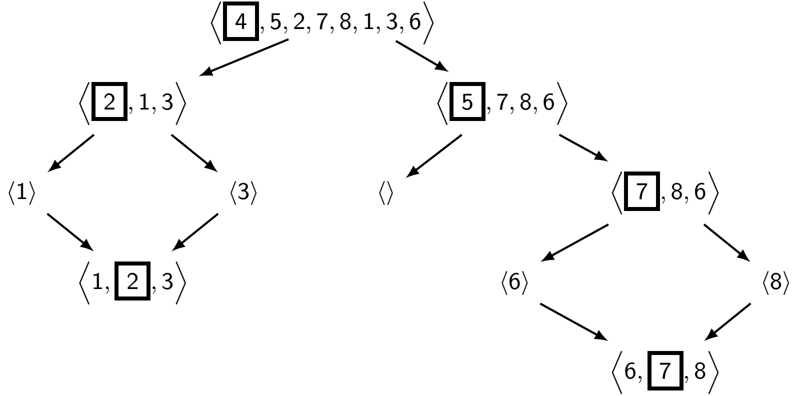
Example: Quick Sort run

Initial pivot selection strategy: we always choose the leftmost entry.

$$
\begin{tikzcd}
 & & & \left\langle \boxed{4}, 5, 2, 7, 8, 1, 3, 6 \right\rangle \arrow[dll] \arrow[drr] & & & \\
 & \left\langle \boxed{2}, 1, 3 \right\rangle \arrow[dl] \arrow[dr] & & & & \left\langle \boxed{5}, 7, 8, 6 \right\rangle \arrow[dl] \arrow[dr] & \\
\langle 1 \rangle \arrow[dr] & & \langle 3 \rangle \arrow[dl] & & \langle \rangle & & \left\langle \boxed{7}, 8, 6 \right\rangle \\
 & \left\langle 1, \boxed{2}, 3 \right\rangle & & & & &
\end{tikzcd}
$$

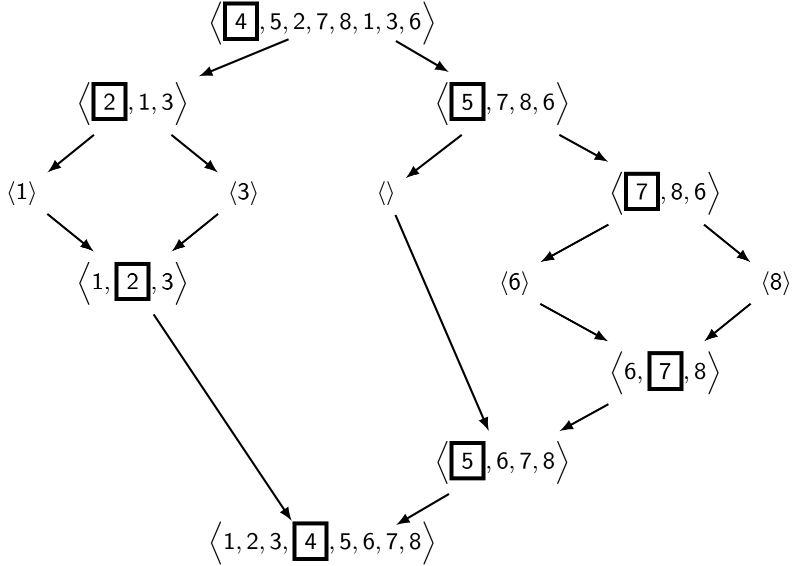
Example: Quick Sort run

Initial pivot selection strategy: we always choose the leftmost entry.



Example: Quick Sort run

Initial pivot selection strategy: we always choose the leftmost entry.



Quick Sort (pseudocode)

```
void quicksort(a, n){
    quicksort_run(a, 0, n-1)
}

quicksort_run(a, left, right) {
  if ( left < right ) {
     pivotindex = partition(a, left, right)
     quicksort_run(a, left, pivotindex-1)
     quicksort_run(a, pivotindex+1, right)
  }
}
```

Where `partition` rearranges the array so that

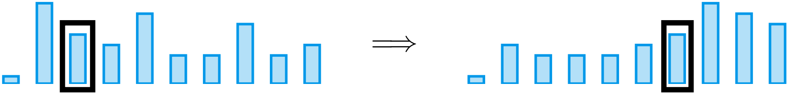
- the small entries are stored on positions `left, left+1, left+2, ..., pivot_index-1`,
- pivot is stored on position `pivot_index` and
- the large entries are stored on `pivot_index+1, pivot_index+2, ..., right`.

Partitioning array `a`

Idea:

1. Choose a pivot `p` from `a`.
2. Allocate two temporary arrays: `tmpLE` and `tmpG`.
3. Store all elements *less than or equal to* `p` to `tmpLE`.
4. Store all elements *greater than* `p` to `tmpG`.
5. Copy the arrays `tmpLE` and `tmpG` back to `a` and return the index of `p` in `a`.

The time complexity of partitioning is $O(n)$.



Partitioning array a , using temporary storage

```
partition(array a, int left, int right) {
  create new array b of size right-left+1
  pivotindex = choosePivot(a, left, right)
  pivot = a[pivotindex]
  acount = left
  bcount = 1
  for ( i = left ; i <= right ; i++ ) {
    if ( i == pivotindex )
      b[0] = a[i]
    else if ( a[i] < pivot ||
              (a[i] == pivot && i < pivotindex) )
      a[acount++] = a[i]
    else
      b[bcount++] = a[i]
  }
  for ( i = 0 ; i < bcount ; i++ )
    a[acount++] = b[i]
  return right-bcount+1
}
```

Time Complexity of Quicksort

Best Case: If the pivot is the *median* in every iteration, then the two partitions have approximately $\frac{n}{2}$ elements.

$\Longrightarrow$ The time complexity is as for Merge Sort, i.e. $O(n \log n)$.

Time Complexity of Quicksort

Best Case: If the pivot is the *median* in every iteration, then the two partitions have approximately $\frac{n}{2}$ elements.

$\implies$ The time complexity is as for Merge Sort, i.e. $O(n \log n)$.

Worst Case: If the pivot is always the *least* element in every iteration, then the second partition contains all elements except for the pivot; it has $n - 1$ elements. In the consecutive iterations:

the second partition has $n - 1, n - 2, n - 3, ..., 1$ elements.

$\implies$ The time complexity is $O(n^2)$.

Time Complexity of Quicksort

Best Case: If the pivot is the *median* in every iteration, then the two partitions have approximately $\frac{n}{2}$ elements.

$\Longrightarrow$ The time complexity is as for Merge Sort, i.e. $O(n \log n)$.

Worst Case: If the pivot is always the *least* element in every iteration, then the second partition contains all elements except for the pivot; it has $n - 1$ elements. In the consecutive iterations:

the second partition has $n - 1, n - 2, n - 3, \ldots, 1$ elements.

$\Longrightarrow$ The time complexity is $O(n^2)$.

Average Case: Depends on the strategy which chooses the pivots! If there are $\geq 25\%$ many small entries or $\geq 25\%$ many large entries in almost every iteration, then the partitioning happens approximately $\log_{4/3} n$-many times

$\Longrightarrow$ The time complexity is $O(n \log n)$.

Pivot-selection strategies

Choose pivot as:

1. the middle entry
   (good for sorted sequences, unlike the leftmost-strategy),
2. the median of the leftmost, rightmost and middle entries,
3. a random entry (there is 50% chance for a good pivot).

Remark: In practice, usually 3. or a variant of 2. is used.

Also, for both quicksort and mergesort, when you reach a small region that you want to sort, it's faster to use selection sort or other sort algorithms. The overhead of Quick S. or Merge .Sort. is big for small inputs.