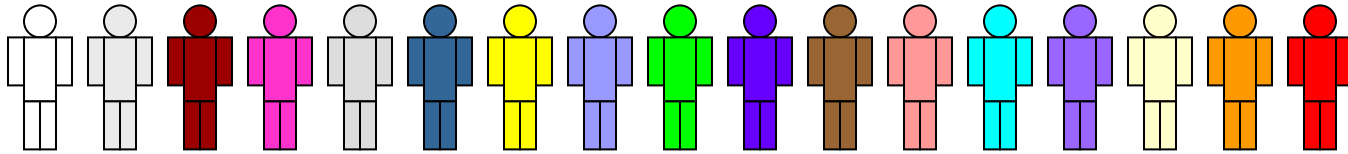
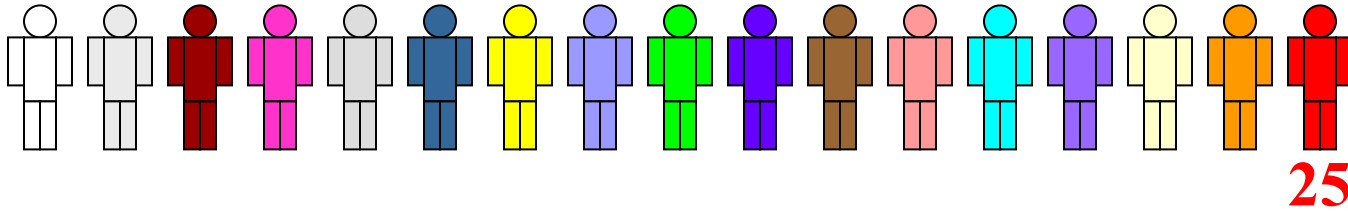

Quicksort

High-level illustration of Quicksort

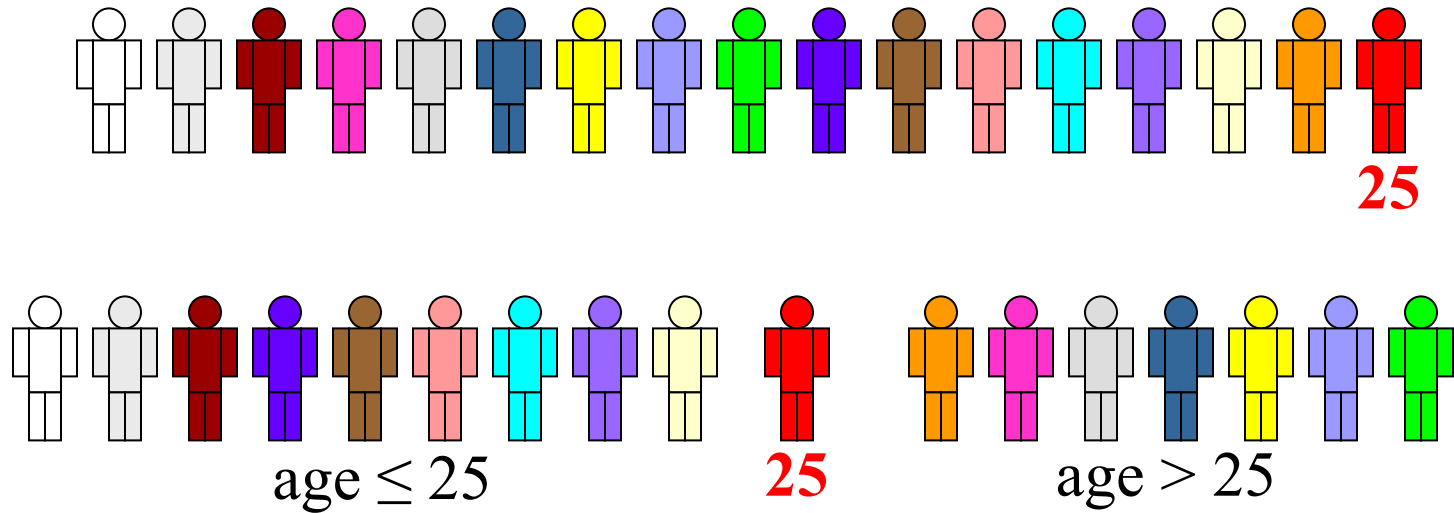


High-level illustration of Quicksort



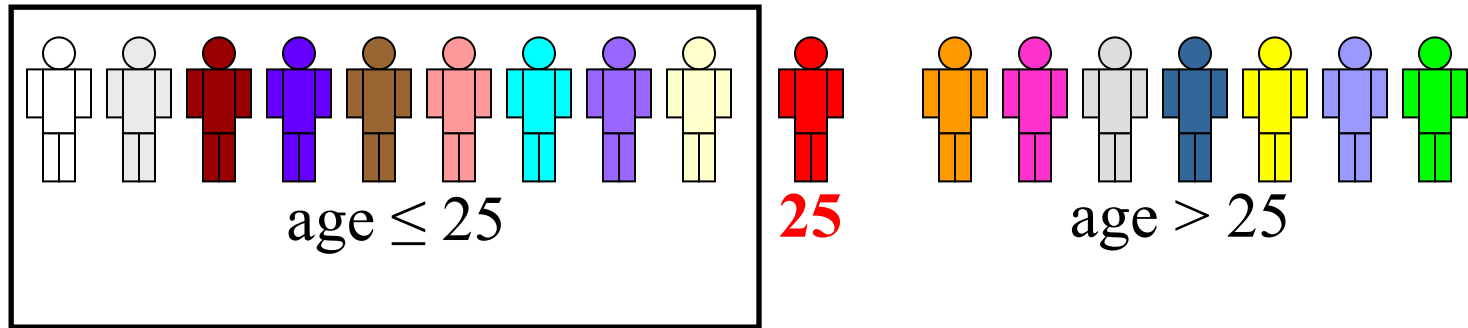
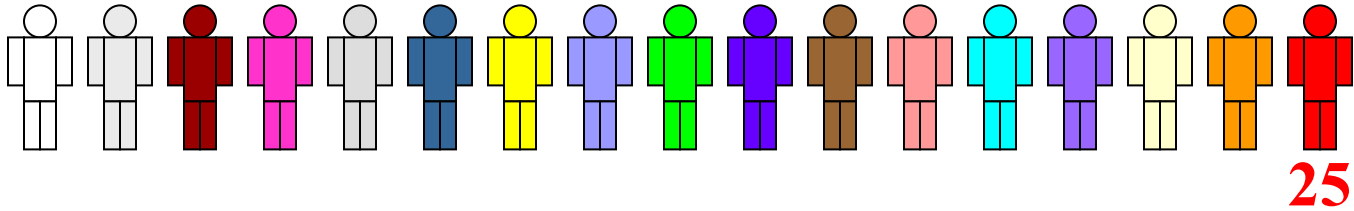
Select pivot

High-level illustration of Quicksort



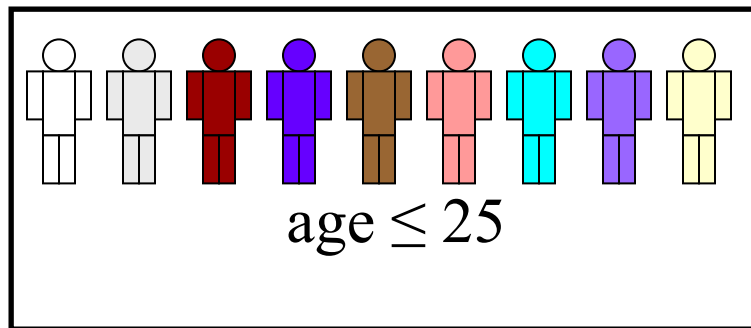
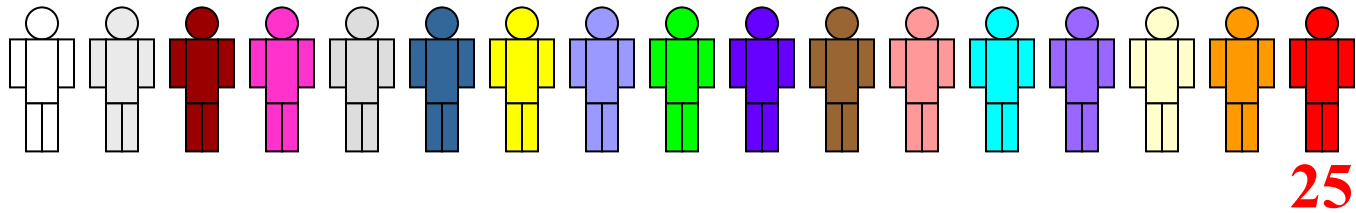
Partition

High-level illustration of Quicksort

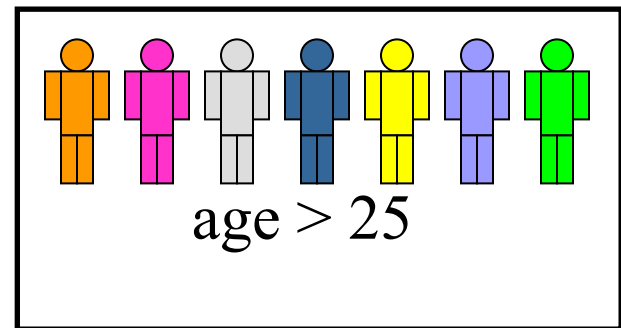
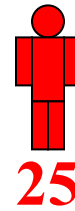


Quicksort left

High-level illustration of Quicksort



Quicksort left



Quicksort right

The Algorithm

Quicksort(A, p, r)

```
1  if p < r
3    q := Partition(A, p, r)
4    Quicksort(A, p, q-1)
5    Quicksort(A, q+1, r)
```

Partition(A, p, r)

```
1  x := A[r]
2  i := p-1
3  for j:=p to r-1 do
4    if A[j] ≤ x then
5      i := i+1
6      temp := A[i]
7      A[i] := A[j]
8      A[j] := temp
9  temp := A[i+1]
10 A[i+1] := A[r]
11 A[r] := temp
12 return i+1
```

How Is *partition* Implemented?

Ideally, we can rearrange the list, so that half elements go to left and half elements go to right of k . In practice:

1. Choose $A[r]$ as the *pivot* element to go to the correct place in the list.
2. If $p \leq i \leq q-1$, then $A[i] \leq \text{pivot}$;
3. If $q+1 \leq j \leq r$, then $A[j] > \text{pivot}$;
4. Scan from left to right, and do $O(1)$ work at each position.

Example

p
■ 2, 8, 7, 1, 3, 5, 6, 4
i
j

x=4

Example

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

x=4

Example

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

x=4

Example

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

x=4

Example

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

x=4

Example

Diagram illustrating the selection of the pivot element (8) in the partitioning step of the Quick Sort algorithm. The array is [2, 8, 7, 1, 3, 5, 6, 4]. The pivot is 8, and the elements are being compared to it. The pivot is highlighted in red, and the elements being compared are highlighted in blue.

x=4

Example

Diagram illustrating the selection of the pivot element (4) in the partitioning step of the Quick Sort algorithm. The array is [2, 8, 7, 1, 3, 5, 6, 4]. The pivot is 4, and the elements are being compared to it. The pivot is highlighted in blue, and the elements being compared are highlighted in red.

x=4

Example

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

x=4

Example

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

x=4

Example

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

x=4

Example

Diagram illustrating the selection of the pivot element (p) and the partitioning process:

The array is: 2, 1, 7, 8, 3, 5, 6, 4.

The pivot (p) is 2, and the rightmost element (r) is 4.

The partitioning process involves moving elements greater than the pivot to the right and elements less than the pivot to the left. The elements 7 and 8 are highlighted in red, indicating they are being moved to the right.

The final array after partitioning is: 2, 1, 3, 5, 6, 4, 7, 8.

x=4

Example

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

x=4

Example

Diagram illustrating an array structure with indices p , i , j , and r . The array contains the values 2, 1, 7, 8, 3, 5, 6, 4. The elements 7 and 8 are highlighted in red, indicating they are the current elements being compared or swapped. The element 4 is highlighted in blue, indicating it is the current element being placed in its sorted position.

x=4

x=4

x=4

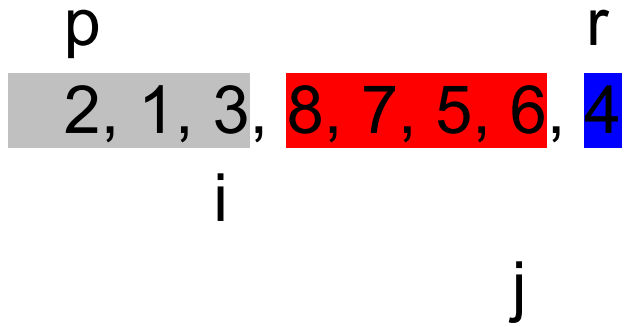
Example

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

x=4

x=4

Example



x=4

Example

Diagram illustrating an array partitioning process. The array is `2, 1, 3, 8, 7, 5, 6, 4`. The partitioning is shown with indices `p` (0), `r` (7), `i` (3), and `j` (7). The array is divided into three segments: `2, 1, 3` (grey background), `8, 7, 5, 6` (red background), and `4` (blue background).

x=4

x=4

Example

x=4

2, 8, 7, 1, 3, 5, 6, 4

2, 8, 7, 1, 3, 5, 6, 4

2, 8, 7, 1, 3, 5, 6, 4

2, 8, 7, 1, 3, 5, 6, 4

2, 1, 7, 8, 3, 5, 6, 4

2, 1, 3, 8, 7, 5, 6, 4

2, 1, 3, 8, 7, 5, 6, 4

2, 1, 3, 8, 7, 5, 6, 4

2, 1, 3, 4, 7, 5, 6, 8

Example

2, 8, 7, 1, 3, 5, 6, 4

x=4

2, 8, 7, 1, 3, 5, 6, 4

2, 8, 7, 1, 3, 5, 6, 4

2, 8, 7, 1, 3, 5, 6, 4

2, 1, 7, 8, 3, 5, 6, 4

2, 1, 3, 8, 7, 5, 6, 4

2, 1, 3, 8, 7, 5, 6, 4

2, 1, 3, 8, 7, 5, 6, 4

1, 2, 3, 4, 7, 5, 6, 8

Example

2, 8, 7, 1, 3, 5, 6, 4

x=4

2, 8, 7, 1, 3, 5, 6, 4

2, 8, 7, 1, 3, 5, 6, 4

2, 8, 7, 1, 3, 5, 6, 4

2, 1, 7, 8, 3, 5, 6, 4

2, 1, 3, 8, 7, 5, 6, 4

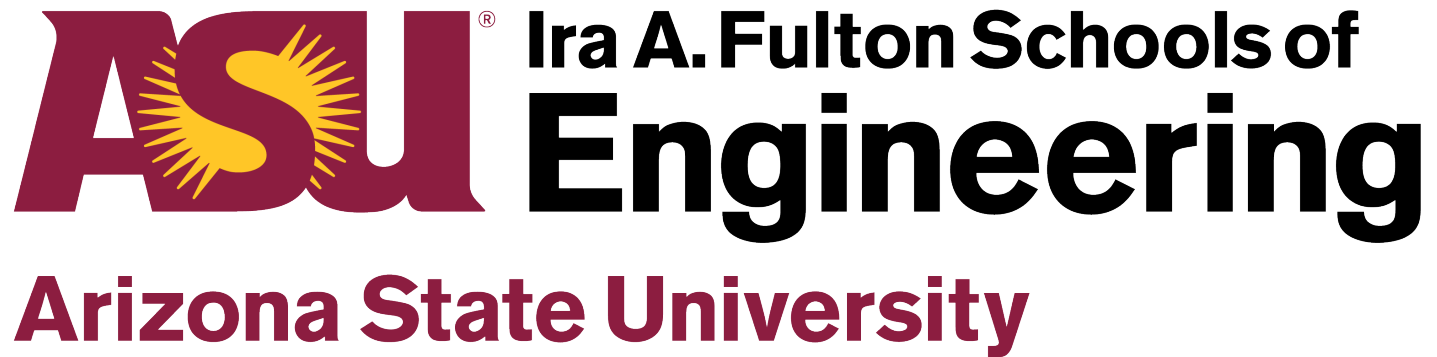
2, 1, 3, 8, 7, 5, 6, 4

2, 1, 3, 8, 7, 5, 6, 4

1, 2, 3, 4, 7, 5, 6, 8

Summary

- **We have presented the quicksort algorithm**
- **We have illustrated the steps in partition**
- **We will study the time complexity of quicksort next**



Time Complexity of Partition

The (best-case, worst-case) time complexity of partition is $\Theta(n)$, where n is the number of elements in the array to be partitioned.

We are scanning the array from left to right, spending $O(1)$ time at each position.

Time Complexity of Quick-Sort

The complexity depends on the partition.

Best-case partitioning: divide into 2 equal sublists

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{otherwise} \end{cases} \Rightarrow T(n) = \Theta(n \lg n)$$

Worst-case partitioning: if the list is already sorted:

The right sublist: 0, and left sublist: $n - 1$ elements

$$T(n) = T(n-1) + \Theta(n) = T(n-2) + \Theta(n-1) + \Theta(n)$$

$$= \dots = \sum_{k=1}^n \Theta(k) = \Theta \sum_{k=1}^n (k) = \Theta \left(\frac{n(n+1)}{2} \right) = \Theta(n^2)$$

Average Time Complexity of Quicksort

- The pivot $A[r]$ partitions the array $A[1:n]$ into a left part of $q-1$ elements and a right part of $n-q$ elements.
- We call this a $(q-1)$ -to- $(n-q)$ split.
- The number q can take any of the integers between 1 and n .
- Since all permutations are equally likely, the probability for q to be equal to k is $1/n$, for $k=1, 2, \dots, n$.

Average Time Complexity of Quicksort

- $T(n) = n + 1 + 2(T(0) + T(1) + \dots + T(n-1))/n$
- $nT(n) = n(n+1) + 2(T(0) + T(1) + \dots + T(n-1))$
- $(n-1)T(n-1) = (n-1)n + 2(T(0) + T(1) + \dots + T(n-2))$
- $nT(n) - (n-1)T(n-1) = 2n + 2T(n-1)$
- $nT(n) = 2n + (n+1)T(n-1)$
- $T(n)/(n+1) = 2/(n+1) + T(n-1)/n$
- $T(n) = (n+1)2(1 + 1/2 + 1/3 + \dots + 1/(n+1)) = \Theta(n \log n)$

Stable Sorting and Non-Stable Sorting

- A sorting algorithm is stable if $A[i]=A[j]$ implies the relative order of the two elements will not change during the sorting process.
- Insertion sort is stable.
- MergeSort is stable
- Quicksort is NOT stable.

Tony Hoare and Quicksort

- **Invented by Tony Hoare in 1959**
- **Divide and Conquer algorithm**
- **Selecting the pivot**
- **Using the pivot to partition**
- **Wiki page**

Tony Hoare

