3. In general, at the $i$th step, examine array elements from $i$ to $N - 1$, find the smallest one in that range, and exchange it with the $i$th element of the array.

2. Exchange all elements from 1 to $N - 1$, find the smallest one and swap it with the 1st element of the input array.

1. Exchange all elements from 0 to $N - 1$, and the smallest one will be at the 1st position.

**Algorithm**: Assume $N$ is size of array

**Input**: same array sorted in ascending order

**Output**: same array sorted in ascending order

**Selected Sort**

**Question**: How can we find a way to speed up selection sort?

This is called an $O(N^2)$ algorithm since the number of steps is $N^2$ (we compare each element with every other element). If it is easy to show that selection sort requires $N^2/2 - \mathcal{O}(N)$ time.

**QuickSort**: $O(n \log n)$ time

**MergeSort**: $O(n \log n)$ time

**Selection Sort**: $O(n^2)$ time

**Sorting Algorithms**:

The last piece?

Binary search works great but how do we create a sorted array in

**Sorts**
If either of the above arrays is completed, copy over all remaining elements into the other array.

If either of the above arrays has a smaller one into which, increment the appropriate index.

In general, compare elements at [p1] with [p2] [a1] and move the

in between to [p2]

and move the smaller one

Keep three indices: p1 into a1, p2 into a2, pm into m.

Create an array whose size = size of a1 + size of a2

Merge sorted arrays a1 and a2.

Can we apply this divide and conquer approach to sorting?

Merge all the partitioning and assembly processes should not be

independent; we can have the running time.

If we divide problem into two subproblems that can be solved

roughly in [O(n)] time, so time for doing all processes is

roughly in [O(n)] time. Suppose you break problem into sub

problem into subproblems and solve subproblems separately

Any time you have a [O(n)] algorithm, it pays to break the

How do we merge sorted arrays? Let us write some code.

How do we sort the parts? Call merge sort recursively.

How do we divide array into two equal parts? Use indices into

Three questions:

Divide array into two equal parts, sort each part and merge

Quicksort, divide-and-conquer algorithm
QuickSort

Good sorting algorithm that does not use extra array storage.

In practice, this can be a disadvantage even though merge sort is

not applicable completely on merge sort: $O(n \log n)$

Advantages of merge sort: need extra storage for temporary

Drawbacks of merge sort: need extra storage for temporary

In partition, how do we partition array in place?

In partition, how do we partition array in place?

In QuickSort:

Why partition needs to be created?

Why partition needs to be created?

When array needs to be created:

When array needs to be created:

For SX and SY separately

For SX and SY separately

SX contains only elements greater than a

SX contains only elements less than or equal to a

Partition array elements into two subarrays SX and SY

Partition array elements into two subarrays SX and SY

Delete idea:

Delete idea:
some people use middle element as pivot.

Heuristic: use first array value as pivot.

However, for array of equal size, any of the two values can be chosen as pivot.

Partitioning abduction: which number do we choose as pivot?

Ideally, we would choose median value since that partitions array into two pieces of equal size.

Remaining question: what number do we choose for pivot?

Solution: keep two indices LEFT and RIGHT.

Keep advancing the two indices till they cross.

Once indices cross over, partitioning is done.

Inplace Partitioning

Can we get all blue balls to the left of all red balls?

Some use median of first, middle and last element.
There are many other sorting methods in the literature.

- Bucket sort
- Radix sort
- Bubble sort
- Shell sort
- Heap sort (see later)

Since extra storage for sorting is needed in practice because it does not make sense. Some asymptotically optimal sorting methods are discussed in lectures.

Summary