# Divide and Conquer

Henry Z. Lo

June 16, 2014

## 1 Divide and Conquer

### 1.1 Merge sort

Recall the merge sort algorithm for sorting an integer array $x$:

1. Split $x$ into one element subarray.
2. Merge pairs of neighboring subarrays into a sorted subarray. Weave together the smaller elements of both arrays first.
3. Continue step 2 until done.

See Figure 1 for a visual.

By the time the algorithm finishes, the array is sorted. This scheme is efficient because there are merging sorted arrays can be done in linear time. Since there are $\log n$ merges, this is an $O(n \log n)$ algorithm.

### 1.2 Divide and conquer algorithms

Divide and conquer algorithms can be broken down into three steps:

- Fracturing the problem into subproblems.
- Solving subproblems recursively.
- Combining these subproblems to arrive at the optimal solution.

Problems which can be solved by divide and conquer usually have the following properties:

- The smallest subproblems are trivial to solve.
- Larger problems can be solved efficiently by combining solved subproblems.

Merge sort fits into this example.

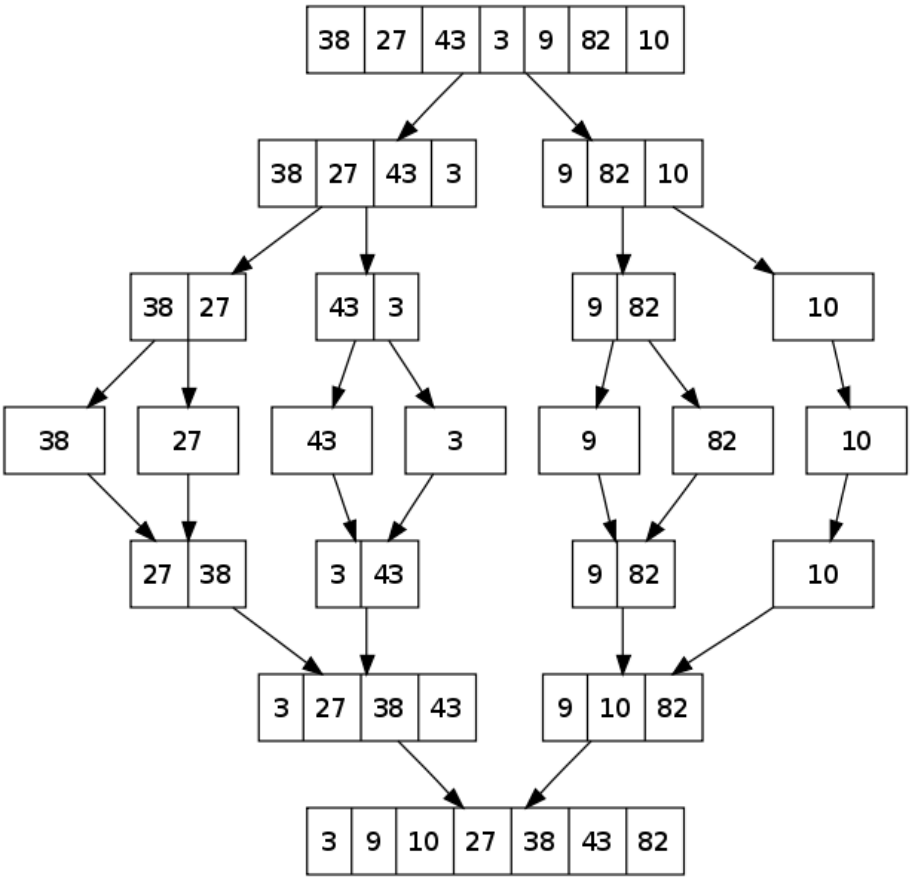


Figure 1: Merge sort.

### 1.3 Binary search

Given a sorted integer array, if we are looking for an integer $x$, we can throw away roughly half the remaining array with one comparison. For example, if we are looking for 6 in the following array, this would be the series of viable candidates during binary search.

```
[1,5,6,8,10,13,14]
[1,5,6]
[6]
```

This solution yields $O(\log n)$ comparisons. This algorithm works because $x$ is in at most one half of the array; once we have deduced which part it's not in, we can disregard it.

Without question, this algorithm involves division. It is arguable whether or not there is a "conquer" step.

## 2 Other Examples

### 2.1 Quicksort

Quicksort is another divide and conquer sorting algorithm, similar to mergesort. However, the sorting is done during the divide step, and the combined result is the sorted array.

The algorithm:

1. Pick an element $x$ from the array.
2. Put everything less than $x$ to the left of $x$, and everything greater or equal to its right.
3. Repeat the second step, until we get to trivially sorted arrays.
4. Combine pairs of subarrays until we get the sorted array.

Even though it is $O(n^2)$ in the worst case (if we keep splitting off one element at a time), on average it is $O(n \log n)$.

### 2.2 Closest pair of points

Consider a set of points $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$. We want to find the closest pair of points. See Figure **??**.

We can do this with a similar algorithm with merge sort:

1. Sort all points by $x$ values.
2. Divide:

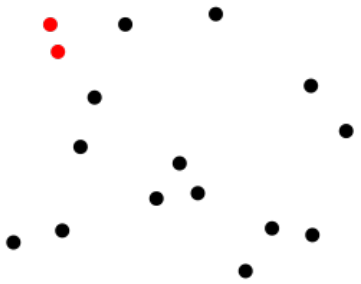

Figure 2: Closest pair of points.

   (a) Find the point on the $x$ axis where half the points are on the left, and half on the right.

   (b) Split into two subarrays.

   (c) Repeat steps 2 and 3 until we get to subarrays with two points.

3. Calculate nearest pair for each subarray.

4. Combine:

   (a) Return the new nearest pair. This is either one of the two subarray pairs, or a new pair with points in different subarrays.

We need to find the nearest pair between subarrays to compare with the two nearest subarray pairs. Let the smaller distance of the nearest subarray pairs be $d$. If there is a pair of points in different subarrays which is closer than $d$, then that is the pair we want.

See Figure 3 for a visual. Let $p_1$ and $p_2$ be those two points. $p_2$ must be within a $d \times 2d$ rectangle of $p_1$, and on the opposite side of the line. There are a limited number of possible points on the opposite side, because they must be at least $d$ apart.

## 2.3 Convex hull

The convex hull of a set of points is a subset which contains all the others. You can think of it as the set of points a rubber band would touch. See Figure 4.

There are many methods to finding convex hulls. We can use divide and conquer, if we divide just as we did in the closest pairs problem. Given two sets of points, their convex hulls can be combined to form the convex hull of the two sets combined.

However, we need to compute the upper and lower *tangents* that connect the two convex hulls (see Figure **??**), and throw out all the points in between.

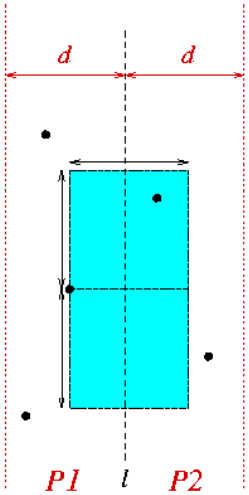


Figure 3: Closest pair of points problem, when points are in different subarrays.

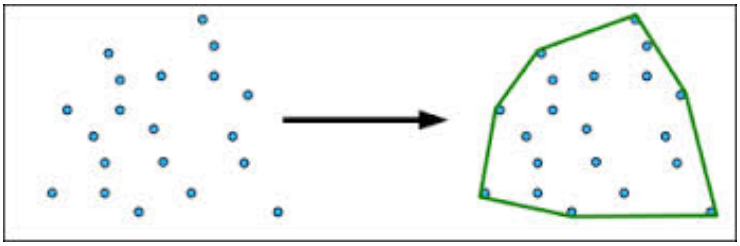

Figure 4: Convex hull.

Figure 5: Combining convex hulls.

We can do this using the following iterative procedure for the bottom tangent:

1. Start at the leftmost point ($a$) on the left set of points, and the rightmost point ($b$) on the right set.
2. If some point on the left convex hull is below the line $ab$, then select the next counterclockwise point to be the new $a$.
3. Likewise for $b$, but update clockwise.

Once the procedure is done, then $ab$ is the lower tangent. This is a linear time procedure.