# Section 2 Supplement

CPSC 365: Algorithms

Amy Zhao

February 3, 2024

# Outline

# Divide and Conquer

## Divide and Conquer: Main Idea

1. Divide the input into smaller subproblems.
2. Conquer the subproblems recursively.
3. Combine the solutions for the subproblems into a solution for the original problem.

All divide and conquer algorithms follow a similar format: start with the entire input data and make a comparison of two values. Depending on this comparison, determine which portion of the input data to recurse on.

# Divide and Conquer: Quicksort

**Main Idea:** Divide the array on a pivot element which is positioned in a way such that elements less than the pivot are kept on the left side, and elements greater than the pivot are kept on the right side.

Recurse on each side until the subarrays become single elements before combining.

**Time Complexity:** $O(n^2)$ (worst case); $O(n \log n)$ (average case)

```
Function Quicksort(A, leftIndex, rightIndex):
    if leftIndex < rightIndex then
        pivot ← Partition(A, leftIndex, rightIndex)
        Quicksort(A, leftIndex, pivot − 1)
        Quicksort(A, pivot, rightIndex)

Function Partition (A, leftIndex, rightIndex):
    set rightIndex as pivot
    storeIndex ← leftIndex − 1

    for i ← leftIndex + 1 to rightIndex do
        if A[i] < A[pivot] then
            swap A[i] and A[storeIndex]
            storeIndex ← storeIndex + 1
        swap A[pivot] and A[storeIndex + 1]
    end
    return storeIndex + 1
```

# Divide and Conquer: Quickselect

**Main Idea:** Quickselect uses the same overall approach as quicksort to find the $k$th smallest element in an unordered list.

It chooses a pivot element and partitions the data on the pivot. It only recurses on the side where the target element is.

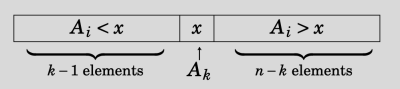
**Time Complexity:** $O(n)$

1. Divide the $n$ elements of $A$ into $\lfloor \frac{n}{5} \rfloor$ groups of five elements each. Add the additional elements to their own group of size $n \bmod 5$.

2. Find the median of each group.

3. Use the algorithm recursively to find the median (denote it with $x$) of the medians found in the previous step.

4. Partition $A$ around $x$ (reorder the elements of $A$ in place so that all elements prior to $x$ are less than $x$). Let $k = \text{rank}(x)$.




5. Recursively call the algorithm on the appropriate part of the array.

   - If $i = k$, return $x$.

   - If $i < k$, recursively call the algorithm on $A[1, \ldots k-1]$ with target order $i$.

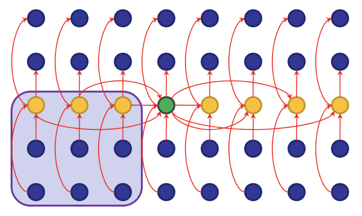
   - If $i > k$, recursively call the algorithm on $A[k+1, \ldots i]$ with new target order $i - k$.

## Quickselect: Time Complexity

The non-recursive steps all take $O(n)$ time.

Notice that the recursive call to find the median of medians takes input data of size $\lceil \frac{n}{5} \rceil$ elements. How do we analyze the runtime for the recursive call in step 5?

WLOG, assume that we need to recurse on the elements larger than the median of medians $x$.



In the diagram, the median of each group is shown in yellow, and $x$ is shown in green. The smaller elements point to the larger elements.

In general, there are $\lceil \frac{n}{5} \rceil$ groups, which implies there are $\lfloor \frac{1}{2} \lceil \frac{n}{5} \rceil \rfloor$ groups with median at most $x$, and $\lfloor \frac{1}{2} \lceil \frac{n}{5} \rceil \rfloor - 1$ groups with median less than $x$.

Each group contributes three elements less than $x$, so we have

$$
\begin{aligned}
3\left(\left\lfloor \frac{1}{2}\left\lceil \frac{n}{5} \right\rceil \right\rfloor - 1\right) &\geq 3\left(\left\lfloor \frac{n}{10} \right\rfloor - 1\right) \\
&= 3\left(\frac{n}{10} - 2\right) = \frac{3n}{10} - 6
\end{aligned}
$$

minimum total number of elements less than $x$, and $\frac{7n}{10} + 6$ elements greater than $x$. Therefore, our recurrence is:

$$T(n) = O(n) + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right)$$

We can solve this recurrence through substitution. Assume $T(n) < cn$:

$$T(n) = an + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right)$$

$$cn \geq c\left(\frac{n}{5}\right) + c\left(\frac{7n}{10}\right) + an$$

$$c \geq \frac{9c}{10} + a \Longrightarrow \frac{c}{10} \geq a \Longrightarrow c \geq 10a$$

Thus, $T(n) = O(n)$.

# Divide and Conquer: Mergesort

**Main Idea:** Divide the array into smaller subarrays, sort each subarray, and merge the sorted subarrays back together.

At each iteration, the algorithm contains two recursive calls because both subarrays need to be sorted.

**Time Complexity:** $O(n \log n)$

```
Function Mergesort(A, n):
    if n > 1 then
        r ← n/2
        L ← A[: r], M ← A[r :]

        Mergesort(L, r)
        Mergesort(M, r)

        i ← 0, j ← 0, k ← 0

        while i < r and j < r do
            if L[i] < M[j] then
                A[k] ← L[i], i ← i + 1
            else
                A[k] ← M[j], j ← j + 1
            k ← k + 1
        end

        while i < r do
            A[k] ← L[i], i ← i + 1, k ← k + 1
        end

        while j < r do
            A[k] ← M[j], j ← j + 1, k ← k + 1
        end
```

# Divide and Conquer: Counting Inversions

**Main Idea:** Divide the array into two subarrays, and recursively count the inversions in each half. In the step to combine, count inversions where $a_i$ and $a_j$ are in different halves, and return the sum of the three quantities.

**Time Complexity:** $O(n \log n)$

Suppose we divide array $A$ into $L$ and $R$ subarrays. There are three cases for an inversion $(i, j)$:

- $i \in L$ and $j \in R$
- $i \in L$ and $j \in L$
- $i \in R$ and $j \in R$

We need to handle case one differently.

```
Function CaseOne(L, R):
    answer ← 0, j ← 0

    for i in 0 to len(L) − 1 do
        while j < len(R) and R[j] < L[i] do
            j ← j + 1
        end
        answer ← answer + j
    end

    return answer

Function CountInversions(A, n):
    if n ≤ 1 then
        return 0
    L ← A[: n/2 − 1], R ← A[n/2 :]
    answer ← CountInversions(L, len(L))
    answer ← answer + CountInversions(R, len(R))
    answer ← answer + CaseOne(L, R)

    return answer
```

# Divide and Conquer: Closest Pair

**Main Idea:** Compute a vertical line that roughly divides the points in half. Find the closest pair in each side recursively, and find the closest pair with one point in each side. Return the best of three solutions.

**Time Complexity:** $O(n \log^2(n))$, but can be reduced to $O(n \log n)$

**Function** `ClosestPair(`$P$`)`:

  $P_x \leftarrow P$ sorted by $x$-coordinate
  $P_y \leftarrow P$ sorted by $y$-coordinate
  $(p_0^*, p_1^*) \leftarrow$ `ClosestPairRec(`$P_x$, $P_y$`)`

**Function** `ClosestPairRec(`$P_x$, $P_y$`)`:

  **if** $|P| \leq 3$ **then**
    **return** *closest pair through computation*

  $Q \leftarrow$ first $\left\lceil \frac{n}{2} \right\rceil$ points in $P_x$, $Q_x \leftarrow Q$ sorted by $x$-coordinate, $Q_y \leftarrow Q$ sorted by $y$-coordinate

  $R \leftarrow$ remaining $\left\lfloor \frac{n}{2} \right\rfloor$ points in $P_x$, $R_x \leftarrow R$ sorted by $x$-coordinate, $R_y \leftarrow R$ sorted by $y$-coordinate

  $(q_0^*, q_1^*) \leftarrow$ `ClosestPairRec(`$Q_x$, $Q_y$`)`
  $(r_0^*, r_1^*) \leftarrow$ `ClosestPairRec (`$R_x$, $R_y$`)`

  $\delta \leftarrow \min\{\mathtt{dist}(q_0^*, q_1^*), \mathtt{dist}(r_0^*, r_1^*)\}$
  $x^* \leftarrow$ maximum $x$-coordinate $\in Q$

  $L \leftarrow \{(x, y) : x = x^*\}$
  $S \leftarrow$ points $\in P$ within distance $\delta$ of $L$, $S_y \leftarrow S$ sorted by $y$-coordinate

  **for** $s \in S_y$ **do**
    compute distance from $s$ to next 15 points in $S_y$
    $(s, s') \leftarrow$ pair which achieves minimum distance
  **end**

  **if** $\mathtt{dist}(s, s') < \delta$ **then**
    **return** $(s, s')$
  **else if** $\mathtt{dist}(q_0^*, q_1^*) < \mathtt{dist}(r_0^*, r_1^*)$ **then**
    **return** $(q_0^*, q_1^*)$
  **else**
    **return** $(r_0^*, r_1^*)$

# Divide and Conquer: Strassen's Algorithm

**Main Idea:** Divide the two matrices into smaller sub-matrices, and calculate the values recursively.

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} S_1 + S_2 - S_4 + S_6 & S_4 - S_5 \\ S_6 + S_7 & S_2 - S_3 + S_5 - S_7 \end{pmatrix}$$

$$\begin{aligned} S_1 &= (B - D)(G + H) & S_2 &= (A + D)(E + H) & S_3 &= (A - C)(E + F) \\ S_4 &= (A + B)H & S_5 &= A(F - H) & S_6 &= D(G - E) \\ S_7 &= (C + D)E \end{aligned}$$

**Time Complexity:** $O(n^{\log_2 7})$

**Function** `Strassen`$(M, N)$:

  **if** $M$ $1 \times 1$ **then**

    **return** $M_{11} N_{11}$

  $M \leftarrow \begin{pmatrix} A & B \\ C & D \end{pmatrix}$, $N \leftarrow \begin{pmatrix} E & F \\ G & H \end{pmatrix}$

  $S_1 \leftarrow$ `Strassen`$(B - D, G + H)$

  $S_2 \leftarrow$ `Strassen`$(A + D, E + H)$

  $S_3 \leftarrow$ `Strassen`$(A - C, E + F)$

  $S_4 \leftarrow$ `Strassen`$(A + B, H)$

  $S_5 \leftarrow$ `Strassen`$(A, F - H)$

  $S_6 \leftarrow$ `Strassen`$(D, G - E)$

  $S_7 \leftarrow$ `Strassen`$(C + D, E)$

  **return** $\begin{pmatrix} S_1 + S_2 - S_4 + S_6 & S_4 - S_5 \\ S_6 + S_7 & S_2 - S_3 + S_5 - S_7 \end{pmatrix}$

# Divide and Conquer: Karatsuba's Algorithm

**Main Idea:** Given two integers (in base-2), perform three recursive multiplications. It relies on the fact that the two integers can be written as lower order and higher order bits:

$$x = x_1 \cdot 2^{\frac{n}{2}} + x_0$$
$$y = y_1 \cdot 2^{\frac{n}{2}} + y_0$$

The product becomes the following:

$$\begin{aligned} xy &= (x_1 \cdot 2^{\frac{n}{2}} + x_0)(y_1 \cdot 2^{\frac{n}{2}} + y_0) \\ &= x_1 y_1 \cdot 2^n + (x_1 y_0 + x_0 y_1) \cdot 2^{\frac{n}{2}} + x_0 y_0 \end{aligned}$$

**Time Complexity:** $O(n^{\log_2 3})$

**Function** FasterMultiplication($x$, $y$):

  write $x$ as $x_1 \cdot 2^{\frac{n}{2}} + x_0$

  write $y$ as $y_1 \cdot 2^{\frac{n}{2}} + y_0$

  $x_{10} \leftarrow x_1 + x_0$

  $y_{10} \leftarrow y_1 + y_0$

  $p \leftarrow$ FasterMultiplication($x_{10}$, $y_{10}$)

  $x_1 y_1 \leftarrow$ FasterMultiplication($x_1$, $y_1$)

  $x_0 y_0 \leftarrow$ FasterMultiplication($x_0$, $y_0$)

  **return** $x_1 y_1 \cdot 2^n + (p - x_1 y_1 - x_0 y_0) \cdot \frac{n}{2} + x_0 y_0$