

## Design Paradigm: Divide and Conquer

- Finding Rank - Merge Sort
- Counting Inversions
- Karatsuba Algorithm for Integers Multiplication
- Finding Closest Pair in Plane

IMDAD ULLAH KHAN

# Algorithm Design Paradigm: Divide and Conquer

- Break the problem into several subproblems – **Divide**
- Recursively, solve the subproblems
- Combine subproblems solutions into an overall solution – **Combine**



source: Khan Academy

## Rank<sub>A</sub>(x)

$A$  is an array of  $n$  integers

**Rank of  $x$  in  $A$**  is the number of elements in  $A$  smaller than  $x$

$$\text{Rank}_A(x) = |\{a \in A : a < x\}|$$

$A = [5 \ 4 \ 6 \ 9 \ 2 \ 7 \ 5 \ 8]$

- $\text{Rank}_A(5) = 2$
- $\text{Rank}_A(3) = 1$
- $\text{Rank}_A(1) = 0$
- $\text{Rank}_A(-10) = 0$
- $\text{Rank}_A(\min(A)) = 0$
- $\text{Rank}_A(\max(A)) = n - \text{freq of max}$

- What if  $\text{Rank}_A(x) = |\{a \in A : a \leq x\}|$ ?
- What if  $A$  is required to have distinct integers/real numbers?

## Compute $\text{Rank}_A(x)$

**Input:** A sorted array  $A$  of  $n$  distinct integers and  $x \in \mathbb{Z}$

**Output:**  $\text{Rank}_A(x)$

- EXTENDED BINARY SEARCH for  $x$  in  $A$

Takes  $\log n$  comparisons

- Linear scan  $A$  and count  $A[i] < x$

Takes  $n$  comparisons

## Compute Rank of 2 numbers

---

**Input:** A sorted array  $A$  of  $n$  distinct integers and  $x_1 < x_2 \in \mathbb{Z}$

**Output:**  $\text{Rank}_A(x_1)$ ,  $\text{Rank}_A(x_2)$

- EXTENDED BINARY SEARCH for  $x_1$  and  $x_2$  in  $A$

Takes  $2 \log n$  comparisons (worst case)

$\text{Rank}_A(x_1) = t \rightarrow$  EXTENDED BINARY SEARCH for  $x_2$  in  $A[t \cdots n]$

▷  $\log n + \log(n - t)$

▷ Worst case:  $\text{Rank}_A(x_1) = 0$

- Linear scan  $A$  and count  $A[i] < x_1$  and  $A[i] < x_2$

Takes  $2n$  comparisons

## Compute Rank of 3 numbers

---

**Input:** A sorted array  $A$  of  $n$  distinct integers and  $x_1 < x_2 < x_3 \in \mathbb{Z}$

**Output:**  $\text{Rank}_A(x_1)$ ,  $\text{Rank}_A(x_2)$ ,  $\text{Rank}_A(x_3)$

- Three EXTENDED BINARY SEARCH for  $x_1, x_2, x_3$  in  $A$

Takes  $3 \log n$  comparisons (worst case)

- Linear scan  $A$ : count  $A[i] < x_1$ ,  $A[i] < x_2$ ,  $A[i] \leq x_3$

Takes  $3n$  comparisons

## Compute Rank of $n$ numbers

**Input:** A sorted array  $A$  of  $n$  distinct integers and  $x_1 < x_2 < \dots < x_n \in \mathbb{Z}$

**Output:**  $\text{Rank}_A(x_i)$ , for  $1 \leq i \leq n$

- $n$  EXTENDED BINARY SEARCH for each  $x_i \in X$  in  $A$

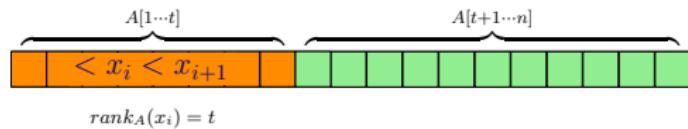
Takes  $n \log n$  comparisons (worst case)

- Linear scan  $A$ : count  $A[i] < x_j$  for  $1 \leq j \leq n$

Takes  $n^2$  comparison

- $\text{Rank}_A(x_i) = t \implies$  For  $x_{i+1}$  **continue** scan from  $A[t+1]$

$\because A[1 \dots t] < x_i \implies A[1 \dots t] < x_{i+1}$



Takes  $2n$  comparisons (worst case)

## Compute Rank of $n$ numbers

---

**Input:** A sorted array  $A$  of  $n$  distinct integers and  $x_1 < x_2 < \dots < x_n \in \mathbb{Z}$

**Output:**  $\text{Rank}_A(x_i)$ , for  $1 \leq i \leq n$

- $\text{Rank}_A(x_i) = t \implies$  For  $x_{i+1}$  **continue** scan from  $A[t+1]$ 
  - ▷ Because  $A[1 \dots t] < x_i \implies A[1 \dots t] < x_{i+1}$

Takes  $2n$  comparisons (worst case)

---

### Algorithm Find Ranks

---

```
j ← 1                                ▷ index of current  $x_j$ 
r ← 0                                  ▷ running rank
for  $i = 1$  to  $n$  do
    if  $A[i] < x_j$  then
         $r \leftarrow r + 1$ 
    else
         $\text{rank}_A(x_j) \leftarrow r$ 
         $j \leftarrow j + 1$ 
         $i \leftarrow i - 1$ 
    ▷ need to repeat this  $i$ 
```

---

## Merge

---

**Input:** Sorted array  $A$  and sorted array  $B$  of  $n$  distinct integers

**Output:** Sorted  $C = A \cup B$ ,  $|C| = 2n$

$$A = \boxed{2 \mid 4 \mid 7 \mid 10 \mid 12} \quad B = \boxed{3 \mid 9 \mid 14 \mid 15 \mid 18}$$

$$C = \boxed{2 \mid 3 \mid 4 \mid 7 \mid 9 \mid 10 \mid 12 \mid 14 \mid 15 \mid 18}$$

The brute-force algorithm (just implements the definition)

Make  $C = A \cup B$  and SORT  $C$  ▷  $O(n^2)$  comparisons

Can make use of the FINDRANK algorithm

## Merge

**Input:** Sorted array  $A$  and sorted array  $B$  of  $n$  distinct integers

**Output:** Sorted  $C = A \cup B$ ,  $|C| = 2n$

$$A = \boxed{2 \quad 4 \quad 7 \quad 10 \quad 12} \quad B = \boxed{3 \quad 9 \quad 14 \quad 15 \quad 18}$$

$$C = \begin{array}{cccccccccc} 2 & 3 & 4 & 7 & 9 & 10 & 12 & 14 & 15 & 18 \end{array}$$

What will be index of  $B[1]$  in  $C$ ?

In  $C$ , elements of  $A$  smaller than  $B[1]$  are to the left of  $B[1]$ .

- Index of  $B[1]$  in  $C$  is  $rank_A(B[1]) + 1$
  - Index of  $B[2]$  in  $C$  is  $rank_A(B[2]) + 2$
  - Index of  $B[3]$  in  $C$  is  $rank_A(B[3]) + 3$

$$C = \begin{array}{ccccccccc} & rank_A(b_1) + 1 & & rank_A(b_2) + 2 & & rank_A(b_3) + 3 \\ \hline & 3 & & & 9 & & & 14 & \end{array}$$

Merging is just FINDRANK

▷ Runtime:  $2n$  comparisons

# Merge Sort

---

**Input:** Array  $A$  of  $n$  distinct integers

**Output:** Sorted  $A$

- Divide  $A$  into left and right halves
- Recursively sort the left and right halves
- Merge the sorted halves

---

**Algorithm** Merge Sort

---

```
function MERGESORT( $A, st, end$  )  
     $n \leftarrow end - st + 1$   
    if  $n = 1$  then  
        return  $A$   
    else  
         $L \leftarrow MERGESORT(A, st, n/2)$   
         $R \leftarrow MERGESORT(A, n/2 + 1, end)$   
        return MERGE( $L, R$ )
```

---

## Merge Sort: Runtime

---

**Input:** Array  $A$  of  $n$  distinct integers

**Output:** Sorted  $A$

---

**Algorithm** Merge Sort

---

```
function MERGESORT( $A, st, end$  )  
     $n \leftarrow end - st + 1$   
    if  $n = 1$  then  
        return  $A$   
    else  
         $L \leftarrow$  MERGESORT( $A, st, n/2$ )  
         $R \leftarrow$  MERGESORT( $A, n/2 + 1, end$ )  
        return MERGE( $L, R$ )
```

---

$T(n)$  : runtime of MERGESORT( $A, n$ )

$$T(n) = \begin{cases} 2T(n/2) + n & \text{if } n > 1 \\ 1 & \text{else} \end{cases}$$

This evaluates to  $O(n \log n)$

▷ matches the lower bound

## Divide and Conquer Design Paradigm

- Break a problem into several parts (**Divide Part**)
- Solve each part recursively
- Combine sub-problems solutions into overall solution (**Combine Part**)
- Sometimes divide part is straight-forward (e.g. Mergesort)
- Sometimes divide part is difficult and combine part is straight-forward (Quicksort)
- Runtime of divide and conquer based algorithm is modeled by a recurrence relation
- Number of operations per call (work for division and combine) plus the number of calls (on certain problem sizes)