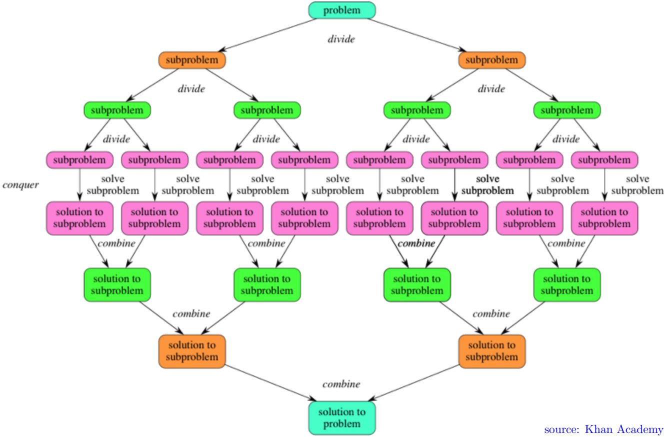
Design Paradigm: Divide and Conquer

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

Imdad ullah Khan

Algorithm Design Paradigm: Divide and Conquer

- Break a problem into several subproblems
- Solve each part recursively
- Combine solutions of sub-problems into overall solution



source: Khan Academy

$Rank_A(x)$

A : is an array of n integers

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

$$Rank_A(x) \ = \ \left|\left\{a \in A \ : \ a < x\right\}\right|$$

$A =$

5	4	6	9	2	7	5	8

- $Rank_A(5) \ = \ 2$
- $Rank_A(3) \ = \ 1$
- $Rank_A(1) \ = \ 0$
- $Rank_A(-10) \ = \ 0$
- $Rank_A\left(min(A)\right) \ = \ 0$
- $Rank_A\left(max(A)\right) \ = \ n -$ freq of max

Compute $Rank_A(x)$

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

Output: $Rank_A(x)$

- 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 $x < y \in \mathbb{Z}$

Output: $Rank_A(x)$, $Rank_A(y)$

- EXTENDED BINARY SEARCH for x and y in A

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

  $Rank_A(x) = t \rightarrow$ next EXTENDED BINARY SEARCH for y in $A[t \ldots n]$

  $\triangleright \log n + \log(n - t)$

  $\triangleright$ Worst case: $Rank_A(x) = 0$

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

  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: $Rank_A(x_1)$, $Rank_A(x_2)$, $Rank_A(x_3)$

- 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 $x_1 < x_2 < \ldots, x_n \in \mathbb{Z}$

Output: $Rank_A(x_i)$, 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

- $Rank_A(x_1) = t \implies$ for x_2 **continue** scan from $A[t+1]$

  ▷ Because $A[1 \ldots t] < x_1 \implies A[1 \ldots t] < x_2$

  Takes $2n$ comparisons (worst case)

Compute Rank of n numbers

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

Output: $Rank_A(x_i)$, for $1 \leq i \leq n$

- $Rank_A(x_1) = t \implies$ for x_2 **continue** scan from $A[t+1]$

  $\triangleright \because A[1 \ldots t] < x_1 \implies A[1 \ldots t] < x_2$

  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 ← r + 1
    else
        rank_A(x_j) ← r
        j ← j + 1
        i ← i − 1                      ▷ need to repeat this i
```

Merge

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

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

$A =$

2	4	7	10	12

$B =$

3	9	14	15	18

$C =$

2	3	4	7	9	10	12	14	15	18

The brute-force algorithm (just implements the definition)

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

Can make use of the FINDRANK algorithm

Merge

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

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

$A =$

2	4	7	10	12

$B =$

3	9	14	15	18

$C =$

2	3	4	7	9	10	12	14	15	18

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

$C =$

	$rank_A(b_1)+1$			$rank_A(b_2)+2$			$rank_A(b_3)+3$	
	3			9			14	

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 $\textsc{MergeSort}(A, st, end\ )$
  $n \leftarrow end - st + 1$
  if $n = 1$ **then**
    return A
  else
    $L \leftarrow \textsc{MergeSort}(A, st, n/2)$
    $R \leftarrow \textsc{MergeSort}(A, n/2 + 1, end)$
    return $\textsc{Merge}(L, R)$

Merge Sort: Runtime

Input: Array A of n distinct integers

Output: Sorted A

Algorithm Merge Sort

```
function MERGESORT(A, st, end )
    n ← end − st + 1
    if n = 1 then
        return A
    else
        L ← MERGESORT(A, st, n/2)
        R ← MERGESORT(A, n/2 + 1, end)
        return MERGE(L, R)
```

$T(n)$: runtime of $\textsc{MergeSort}(A, n)$

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

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