Divide-and-Conquer: Searching in an Array

Neil Rhodes

Department of Computer Science and Engineering
University of California, San Diego

Data Structures and Algorithms
Algorithmic Toolbox

Outline

1. Main Idea of Divide-and-Conquer
2. Linear Search
3. Binary Search

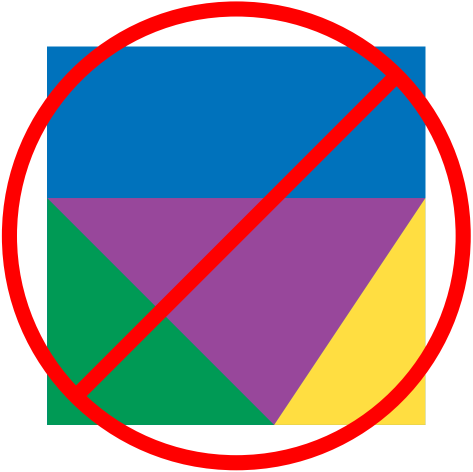
a problem to be solved

Divide: Break into non-overlapping subproblems of the same type

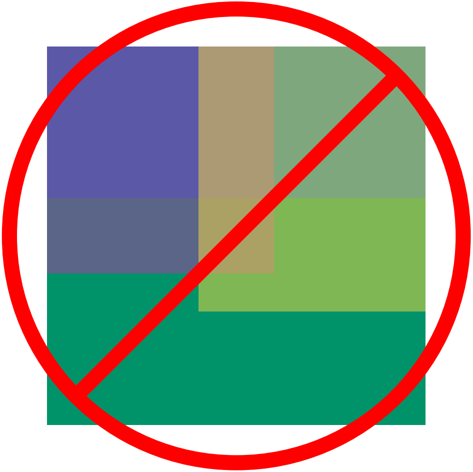
Divide: Break into non-overlapping subproblems of the same type

Divide: Break into non-overlapping subproblems of the same type

Divide: Break into non-overlapping subproblems of the same type



not the
same type

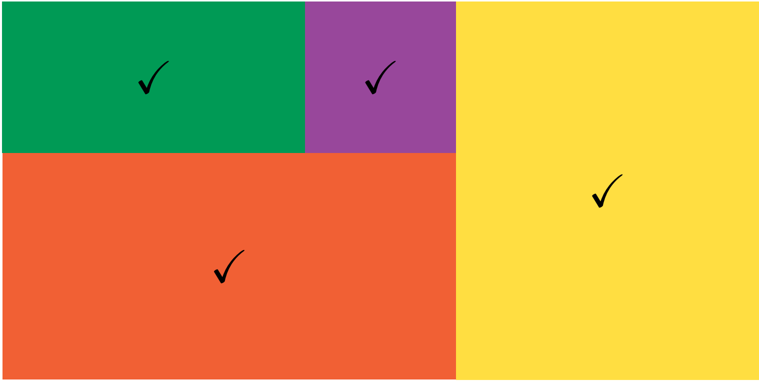


overlapping

Divide: break apart

Conquer: solve subproblems

Conquer: combine



✓

1. Break into non-overlapping subproblems of the same type
2. Solve subproblems
3. Combine results

Outline

1. Main Idea of Divide-and-Conquer

2. Linear Search

3. Binary Search

Searching in an array

Input: An array A with n elements.
A key k.

Output: An index, i, where $A[i] = k$.
If there is no such i, then
NOT_FOUND.

Recursive Solution

$\texttt{LinearSearch}(A, \mathit{low}, \mathit{high}, \mathit{key})$

```
if high < low:
  return NOT_FOUND
if A[low] = key:
  return low
return LinearSearch(A, low + 1, high, key)
```

Definition

A recurrence relation is an equation recursively defining a sequence of values.

Fibonacci recurrence relation

$$F(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F(n-1) + F(n-2) & \text{if } n > 1 \end{cases}$$

$$0, 1, 1, 2, 3, 5, 8, \ldots$$

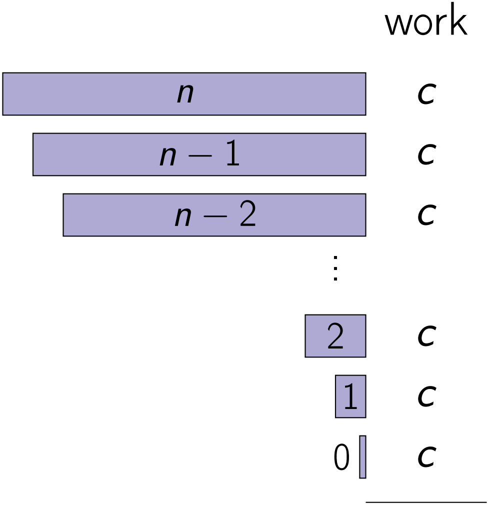
$\texttt{LinearSearch}(A, low, high, key)$

```
if high < low:
    return NOT_FOUND
if A[low] = key:
    return low
return LinearSearch(A, low + 1, high, key)
```

Recurrence defining worst-case time:

$$T(n) = T(n-1) + c$$
$$T(0) = c$$

Runtime of Linear Search



Total: $\sum_{i=0}^{n} c = \Theta(n)$

Iterative Version

LinearSearchIt$(A, low, high, key)$

```
for i from low to high:
  if A[i] = key:
    return i
return NOT_FOUND
```

Summary

- Create a recursive solution
- Define a corresponding recurrence relation, T
- Determine $T(n)$: worst-case runtime
- Optionally, create iterative solution

Outline

1. Main Idea of Divide-and-Conquer

2. Linear Search

3. Binary Search

Searching in a sorted array

Input: A sorted array $A[low \dots high]$
($\forall low \leq i < high$: $A[i] \leq A[i+1]$).
A key k.

Output: An index, i, ($low \leq i \leq high$) where
$A[i] = k$.
Otherwise, the greatest index i,
where $A[i] < k$.
Otherwise ($k < A[low]$), the result is
$low - 1$.

Searching in a Sorted Array

Example

$$
\begin{array}{ll}
\mathit{search}(2) \to 0 & \mathit{search}(20) \to 4 \\
\mathit{search}(3) \to 1 & \mathit{search}(20) \to 5 \\
\mathit{search}(4) \to 1 & \mathit{search}(60) \to 7 \\
 & \mathit{search}(90) \to 7
\end{array}
$$

3	5	8	20	20	50	60
1	2	3	4	5	6	7

BinarySearch($A, low, high, key$)

```
if high < low:
  return low − 1
mid ← ⌊low + (high − low)/2⌋
if key = A[mid]:
  return mid
else if key < A[mid]:
  return BinarySearch(A, low, mid − 1, key)
else:
  return BinarySearch(A, mid + 1, high, key)
```

Example: Searching for the key 50

$\texttt{BinarySearch}(A, 1, 11, 50)$

1	2	3	4	5	6	7	8	9	10	11
3	5	8	10	12	15	18	20	20	50	60

low *mid* *high*

Example: Searching for the key 50

$\texttt{BinarySearch}(A, 1, 11, 50)$
$\texttt{BinarySearch}(A, 7, 11, 50)$

1	2	3	4	5	6	7	8	9	10	11
3	5	8	10	12	15	18	20	20	50	60

low *mid* *high*

Example: Searching for the key 50

$\texttt{BinarySearch}(A, 1, 11, 50)$
$\texttt{BinarySearch}(A, 7, 11, 50)$
$\texttt{BinarySearch}(A, 10, 11, 50) \to 10$

1	2	3	4	5	6	7	8	9	10	11
3	5	8	10	12	15	18	20	20	50	60

Summary

- Break problem into non-overlapping subproblems of the same type.
- Recursively solve those subproblems.
- Combine results of subproblems.

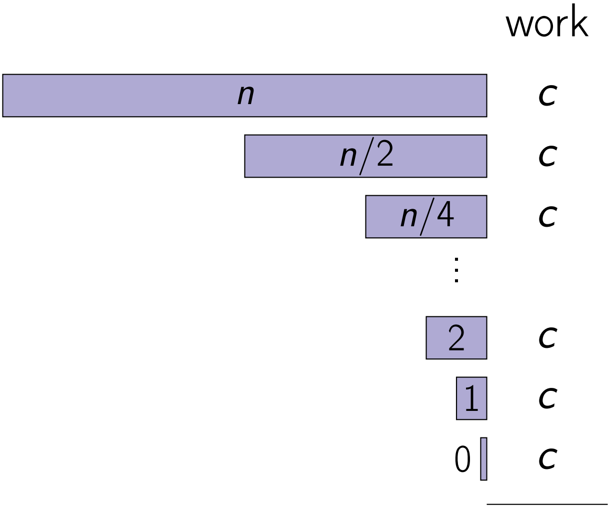
BinarySearch$(A, low, high, key)$

```
if high < low:
  return low − 1
mid ← ⌊low + (high−low)/2⌋
if key = A[mid]:
  return mid
else if key < A[mid]:
  return BinarySearch(A, low, mid − 1, key)
else:
  return BinarySearch(A, mid + 1, high, key)
```

Binary Search Recurrence Relation

$$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + c$$
$$T(0) = c$$

Runtime of Binary Search



Total: $\sum_{i=0}^{\log_2 n} c = \Theta(\log_2 n)$

Iterative Version

$\texttt{BinarySearchIt}(A, low, high, key)$

```
while low ≤ high:
    mid ← ⌊low + (high − low)/2⌋
    if key = A[mid]:
        return mid
    else if key < A[mid]:
        high = mid − 1
    else:
        low = mid + 1
return low − 1
```

Real-life Example

english	**french**	**italian**	**german**	**spanish**
house	maison	casa	Haus	casa
chair	chaise	sedia	Sessel	silla
pimple	bouton	foruncolo	Pickel	espenilla

english sorted
2
1
3

spanish sorted
1
3
2

Summary

The runtime of binary search is $\Theta(\log n)$.