Quicksort

09114319: Data Structures and Algorithms

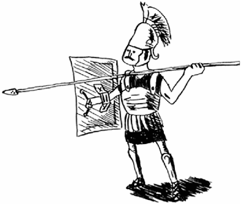
Ratthaprom PROMKAM, Dr. rer. nat.

Department of Mathematics and Computer Science, RMUTT

Outline

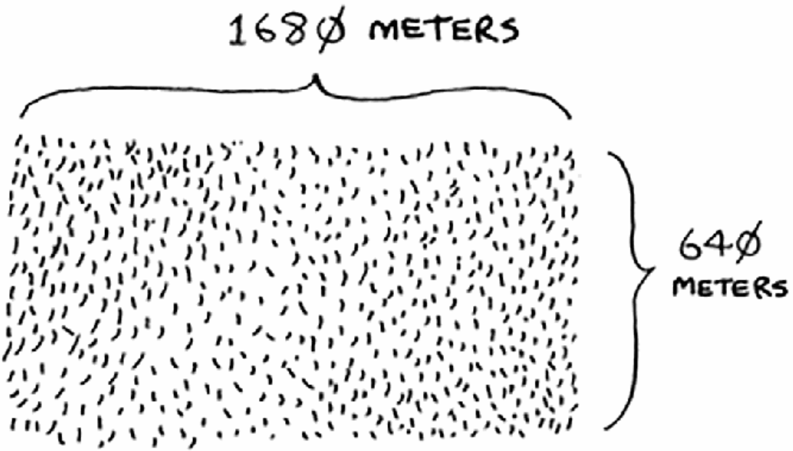
Divide and Conquer

- ✎ Divide and Conquer (DC) is a strategy to solve complex problems by breaking them into smaller subproblems.
- ✎ Steps to solve a problem using DC:
  1. Identify the **base case**, the simplest case that can be solved directly.
  2. Recursively divide the problem into smaller parts until the base case is reached.



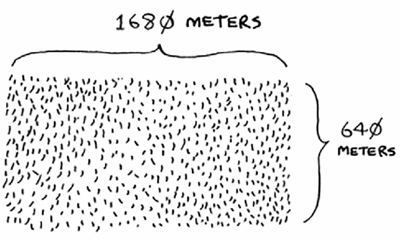
Example 1 (Largest Square Plots)

Dividing a 1680m $\times$ 640m farm into the largest square plots.



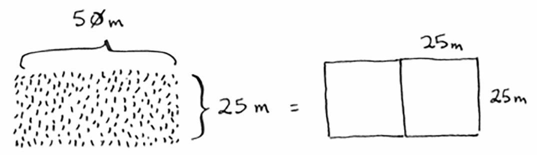
Example 1 (Largest Square Plots)

Dividing a 1680m $\times$ 640m farm into the largest square plots.



Example 1 (Largest Square Plots)

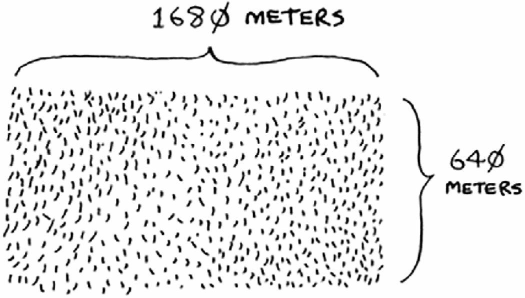
Dividing a 1680m $\times$ 640m farm into the largest square plots.



- ✎ Base case: When one side is a multiple of the other.
- ✎ Start with the entire farm.
- ✎ Recursively divide the leftover area using the largest square possible.

Example 1 (Largest Square Plots)

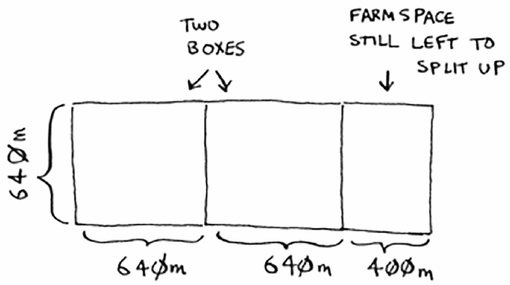
Dividing a 1680m $\times$ 640m farm into the largest square plots.



- Base case: When one side is a multiple of the other.
- Start with the entire farm.
- Recursively divide the leftover area using the largest square possible.

Example 1 (Largest Square Plots)

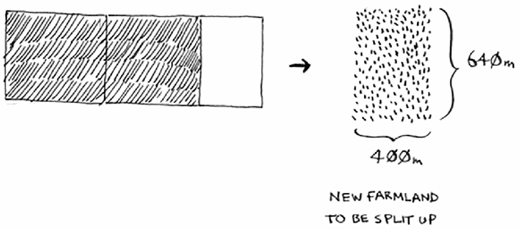
Dividing a 1680m $\times$ 640m farm into the largest square plots.



- ✎ Base case: When one side is a multiple of the other.
- ✎ Start with the entire farm.
- ✎ Recursively divide the leftover area using the largest square possible.

Example 1 (Largest Square Plots)

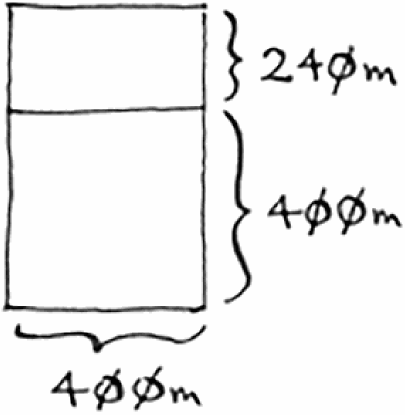
Dividing a 1680m $\times$ 640m farm into the largest square plots.



- Base case: When one side is a multiple of the other.
- Start with the entire farm.
- Recursively divide the leftover area using the largest square possible.

Example 1 (Largest Square Plots)

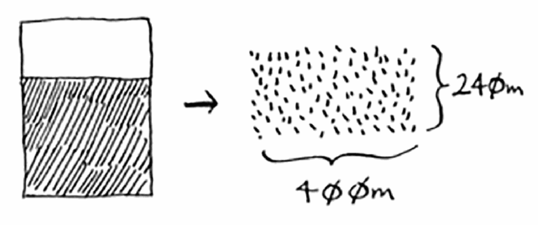
Dividing a 1680m $\times$ 640m farm into the largest square plots.



- ✎ Base case: When one side is a multiple of the other.
- ✎ Start with the entire farm.
- ✎ Recursively divide the leftover area using the largest square possible.

Example 1 (Largest Square Plots)

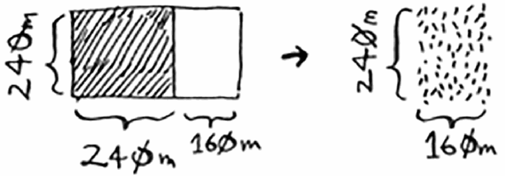
Dividing a 1680m × 640m farm into the largest square plots.



- ✎ Base case: When one side is a multiple of the other.
- ✎ Start with the entire farm.
- ✎ Recursively divide the leftover area using the largest square possible.

Example 1 (Largest Square Plots)

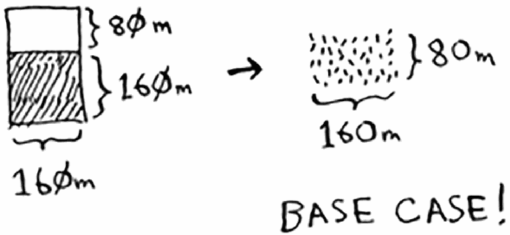
Dividing a 1680m × 640m farm into the largest square plots.



- ✎ Base case: When one side is a multiple of the other.
- ✎ Start with the entire farm.
- ✎ Recursively divide the leftover area using the largest square possible.

Example 1 (Largest Square Plots)

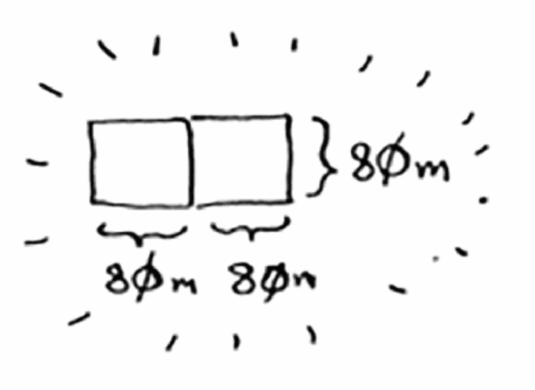
Dividing a 1680m $\times$ 640m farm into the largest square plots.



- ✎ Base case: When one side is a multiple of the other.
- ✎ Start with the entire farm.
- ✎ Recursively divide the leftover area using the largest square possible.

Example 1 (Largest Square Plots)

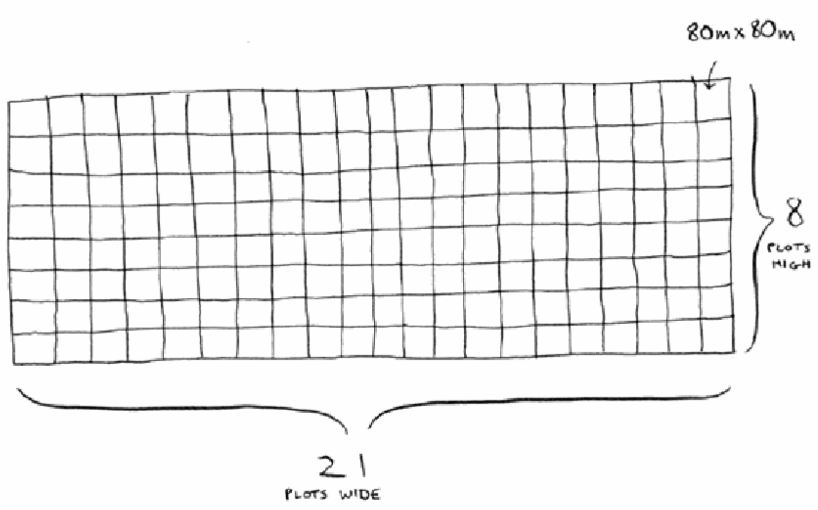
Dividing a 1680m $\times$ 640m farm into the largest square plots.



- ✎ Base case: When one side is a multiple of the other.
- ✎ Start with the entire farm.
- ✎ Recursively divide the leftover area using the largest square possible.

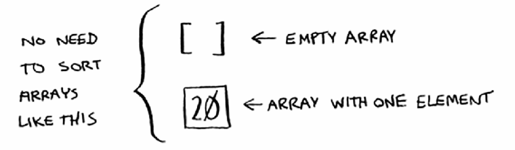
Example 1 (Largest Square Plots)

Dividing a 1680m $\times$ 640m farm into the largest square plots.



Quicksort Algorithm

- ✎ Quicksort is a **divide-and-conquer** sorting algorithm.
- ✎ Base case: Arrays with 0 or 1 element are already sorted.



- ✎ Steps of Quicksort:
  1. Choose a **pivot** element from the array.
  2. Partition the array into two sub-arrays:
     - ✎ Elements less than the pivot.
     - ✎ Elements greater than the pivot.
  3. Recursively apply quicksort on the sub-arrays.
  4. Combine the sorted sub-arrays and the pivot.

Example 2

Sort the array [10, 5, 2, 3] with quicksort algorithm.

1. Choose pivot $= 10$ $\Longrightarrow$ $\begin{array}{rl} \text{Less:} & [5, 2, 3] \\ \text{Greater:} & [\ ] \end{array}$

Quicksort Algorithm

Example 2

Sort the array [10, 5, 2, 3] with quicksort algorithm.

1. Choose pivot $= 10 \implies$
$$\begin{array}{rl} \text{Less:} & [5,\ 2,\ 3] \\ \text{Greater:} & [\ ] \end{array}$$

Quicksort Algorithm

Example 2

Sort the array [10, 5, 2, 3] with quicksort algorithm.

1. Choose pivot $= 10 \implies \begin{array}{rl} \text{Less:} & [5, 2, 3] \\ \text{Greater:} & [\ ] \end{array}$
2. Recursively sort [5, 2, 3]:

   ✎ Pivot $= 5 \implies \begin{array}{rl} \text{Less:} & [2, 3] \\ \text{Greater:} & [\ ] \end{array}$

Quicksort Algorithm

Example 2

Sort the array [10, 5, 2, 3] with quicksort algorithm.

1. Choose pivot $= 10 \Longrightarrow$ $\begin{array}{rl} \text{Less:} & [5, 2, 3] \\ \text{Greater:} & [\ ] \end{array}$
2. Recursively sort [5, 2, 3]:

   ✎ Pivot $= 5 \Longrightarrow$ $\begin{array}{rl} \text{Less:} & [2, 3] \\ \text{Greater:} & [\ ] \end{array}$
3. Recursively sort [2, 3]:

   ✎ Pivot $= 2 \Longrightarrow$ $\begin{array}{rl} \text{Less:} & [\ ] \\ \text{Greater:} & [3] \end{array}$

Example 2

Sort the array [10, 5, 2, 3] with quicksort algorithm.

1. Choose pivot $= 10 \implies \begin{array}{rl} \text{Less:} & [5,\ 2,\ 3] \\ \text{Greater:} & [\ ] \end{array}$
2. Recursively sort [5, 2, 3]:

   ✎ Pivot $= 5 \implies \begin{array}{rl} \text{Less:} & [2,\ 3] \\ \text{Greater:} & [\ ] \end{array}$
3. Recursively sort [2, 3]:

   ✎ Pivot $= 2 \implies \begin{array}{rl} \text{Less:} & [\ ] \\ \text{Greater:} & [3] \end{array}$
4. Combine results: $[2,\ 3] \to [2,\ 3,\ 5] \to [2,\ 3,\ 5,\ 10]$.

Quicksort Algorithm

```python
def quicksort(array):
    if len(array) < 2:  # Base case
        return array
    else:
        pivot = array[0]  # Choose the pivot
        less = less_elements(array[1:], pivot)
        greater = greater_elements(array[1:], pivot)
        return quicksort(less) + [pivot] + quicksort(greater)

def less_elements(array, value):
    return [k for k in array if k <= value]

def greater_elements(array, value):
    return [k for k in array if k > value]
```

Merge Sort Algorithm

- Merge sort is a **divide-and-conquer** algorithm that divides the array into halves, sorts them, and merges the results.
- Steps of Merge Sort:
  1. Divide the array into two halves.
  2. Recursively sort each half.
  3. Merge the sorted halves into a single sorted array.
- Always runs in $O(n \log n)$ time complexity.
- Requires additional memory for merging.

Example 3

Sort the array [10, 5, 2, 3] with merge sort algorithm.

Example 3

Sort the array [10, 5, 2, 3] with merge sort algorithm.

1. Divide into two halves: [10, 5] and [2, 3].

Example 3

Sort the array [10, 5, 2, 3] with merge sort algorithm.

1. Divide into two halves: [10, 5] and [2, 3].
2. Recursively sort each half:

Merge Sort Algorithm

Example 3

Sort the array [10, 5, 2, 3] with merge sort algorithm.

1. Divide into two halves: [10, 5] and [2, 3].
2. Recursively sort each half:
   - ✎ Sort [10, 5]: Divide into [10] and [5], merge to [5, 10].

Merge Sort Algorithm

Example 3

Sort the array [10, 5, 2, 3] with merge sort algorithm.

1. Divide into two halves: [10, 5] and [2, 3].
2. Recursively sort each half:
   - ✎ Sort [10, 5]: Divide into [10] and [5], merge to [5, 10].
   - ✎ Sort [2, 3]: Divide into [2] and [3], merge to [2, 3].

Merge Sort Algorithm

Example 3

Sort the array [10, 5, 2, 3] with merge sort algorithm.

1. Divide into two halves: [10, 5] and [2, 3].
2. Recursively sort each half:
   - ✎ Sort [10, 5]: Divide into [10] and [5], merge to [5, 10].
   - ✎ Sort [2, 3]: Divide into [2] and [3], merge to [2, 3].
3. Merge [5, 10] and [2, 3] to form [2, 3, 5, 10].

Merge Sort Code Example

```python
def merge_sort(array):
    if len(array) < 2:  # Base case
        return array
    mid = len(array) // 2
    left = merge_sort(array[:mid])
    right = merge_sort(array[mid:])
    return merge(left, right)

def merge(left, right):
    result = []
    while left and right:
        if left[0] <= right[0]:
            result.append(left.pop(0))
        else:
            result.append(right.pop(0))
    result.extend(left or right)
    return result
```

Big-O Notations Revisited

	Quicksort	**Merge Sort**
Worst Case	$O(n^2)$ Poor pivot choice	$O(n \log n)$
Best Case	$O(n \log n)$ Pivot with Equal splits	$O(n \log n)$
Average Case	$O(n \log n)$	$O(n \log n)$
Call Stacks	$O(\log n)$ to $O(n)$	$O(\log n)$
Memory	In-place (No extra array)	$O(n)$ Temporary arrays

Real-World Applications of Quicksort and Merge Sort

- **Quicksort:**
  - Faster for in-memory sorting.
  - Commonly used in programming libraries.
- **Merge Sort:**
  - Preferred for external sorting (e.g., large datasets).
  - Suitable for linked lists as it doesn't require random access.

Recap

- Divide and Conquer is a powerful problem-solving strategy:
  - Identify the base case.
  - Recursively divide the problem.
- Quicksort:
  - Efficient sorting algorithm using DC.
  - Average time complexity: $O(n \log n)$.
- Merge Sort:
  - Always $O(n \log n)$ with consistent performance.
  - Useful for external sorting and linked lists.