Quicksort Correctness Proof

https://cs.pomona.edu/classes/cs140/
Outline

Topics and Learning Objectives
- Learn how quicksort works
- Learn how to partition an array

Exercise
- Loop Invariant
Extra Resources

- [https://me.dt.in.th/page/Quicksort/](https://me.dt.in.th/page/Quicksort/)
- [https://www.youtube.com/watch?v=ywWBy6J5gz8](https://www.youtube.com/watch?v=ywWBy6J5gz8)
- CLRS Chapter 7
What do we need to do?

1. Prove that **Partition** works
2. Prove that **QuickSort** works
Not a copy! (In-place)

(partition)

(doesn't match function from slides)
<table>
<thead>
<tr>
<th>Base</th>
<th>23</th>
<th>31</th>
<th>51</th>
<th>89</th>
<th>67</th>
<th>91</th>
<th>47</th>
</tr>
</thead>
</table>

Not a copy!

(partition)

doesn't match function from slides
Not a copy!

11 23 31 51 89 67 91 47

(partition)

(base)

doesn't match function from slides

11 23 31 51 89 67 91 47

(base)

Not a copy!

11 23 31 51 89 67 91 47

(base)

Not a copy!

11 23 31 51 89 67 91 47

(base)
Partition proof of correctness

<table>
<thead>
<tr>
<th>Value</th>
<th>67</th>
<th>44</th>
<th>...</th>
<th>21</th>
<th>-87</th>
<th>...</th>
<th>5</th>
<th>101</th>
<th>-31</th>
<th>...</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>left</td>
<td>left + 1</td>
<td>...</td>
<td>right - 1</td>
<td>right</td>
<td>right + 1</td>
<td>...</td>
<td>n - 1</td>
</tr>
</tbody>
</table>

FUNCTION Partition(array, left_index, right_index)

pivot_value = array[left_index]
i = left_index + 1

FOR j IN [left_index + 1 ..< right_index]
  IF array[j] < pivot_value
    swap(array, i, j)
    i = i + 1
  swap(array, left_index, i - 1)

RETURN i - 1
## Partition proof of correctness

<table>
<thead>
<tr>
<th>Value</th>
<th>67</th>
<th>44</th>
<th>…</th>
<th>21</th>
<th>-87</th>
<th>…</th>
<th>5</th>
<th>101</th>
<th>-31</th>
<th>…</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>0</td>
<td>1</td>
<td>…</td>
<td>left</td>
<td>left + 1</td>
<td>…</td>
<td>right - 1</td>
<td>right</td>
<td>right + 1</td>
<td>…</td>
<td>n - 1</td>
</tr>
</tbody>
</table>

**FUNCTION** Partition(array, left_index, right_index)

1. pivot_value = array[left_index]
2. i = left_index + 1
3. **FOR** j **IN** [left_index + 1 ..< right_index]
   1. **IF** array[j] < pivot_value
      1. swap(array, i, j)
      2. i = i + 1
4. swap(array, left_index, i - 1)
5. **RETURN** i - 1

---

How do we prove that **Partition** is correct?
Loop Invariant Proofs

1. State the loop invariant
   1. A statement that can be easily proven true or false
   2. The statement should reference the purpose of the loop
   3. The statement should reference variables that change each iteration

2. Show that the loop invariant is true before the loop starts

3. Show that the loop invariant holds when executing any iteration

4. Show that the loop invariant holds once the loop ends
Loop Invariant

• Loop invariants will usually look something like this:

At the start of the iteration with index $i$, ... the subarray $array[0 \ldots]$ ...
1. **State the loop invariant**
   1. A statement that can be easily proven true or false
   2. The statement should **reference** the purpose of the loop
   3. The statement should **reference** variables that change each iteration
   
     **Initialization**

2. **Show that the loop invariant is true before the loop starts**

   **Maintenance**

3. **Show that the loop invariant holds when executing any iteration**

4. **Show that the loop invariant holds once the loop ends**

**Termination**

---

**FUNCTION** Partition(array, left_index, right_index)

pivot_value = array[left_index]

i = left_index + 1

FOR j IN [left_index + 1 ..< right_index]

    IF array[j] < pivot_value
    swap(array, i, j)
    i = i + 1

swap(array, left_index, i - 1)

RETURN i - 1
Partition proof of correctness

<table>
<thead>
<tr>
<th>Value</th>
<th>67</th>
<th>44</th>
<th>...</th>
<th>21</th>
<th>-87</th>
<th>...</th>
<th>5</th>
<th>101</th>
<th>-31</th>
<th>...</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>left</td>
<td>left + 1</td>
<td>...</td>
<td>right - 1</td>
<td>right</td>
<td>right + 1</td>
<td>...</td>
<td>n - 1</td>
</tr>
</tbody>
</table>

**FUNCTION** `Partition(array, left_index, right_index)`

```
pivot_value = array[left_index]
i = left_index + 1
FOR j IN [left_index + 1 ..< right_index]
    IF array[j] < pivot_value
        swap(array, i, j)
i = i + 1
swap(array, left_index, i - 1)
RETURN i - 1
```

How do we prove that `Partition` is correct?

Exercise
**Partition proof of correctness**

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>67</td>
</tr>
<tr>
<td>44</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>21</td>
</tr>
<tr>
<td>-87</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>101</td>
</tr>
<tr>
<td>-31</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>left</td>
</tr>
<tr>
<td>left + 1</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>right - 1</td>
</tr>
<tr>
<td>right</td>
</tr>
<tr>
<td>right + 1</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>n - 1</td>
</tr>
</tbody>
</table>

**FUNCTION Partition(array, left_index, right_index)**

```plaintext
pivot_value = array[left_index]
i = left_index + 1
FOR j IN [left_index + 1 ..< right_index]
    IF array[j] < pivot_value
        swap(array, i, j)
i = i + 1
swap(array, left_index, i - 1)
RETURN i - 1
```

*Loop Invariant:* At the start of the iteration with indices i and j:
1. All items in `array[i+1 ..= i-1]` are `< pivot_value`
2. All items in `array[i ..= j-1]` are `≥ pivot_value`

**How do we prove that Partition is correct?**
Loop Invariant: At the start of the iteration with indices $i$ and $j$:
1. All items in $a[l+1 \ldots i-1]$ are $<$ pivot
2. All items in $a[i \ldots j-1]$ are $\geq$ pivot

FUNCTION Partition(a, l, r)
    pivot_value = a[l]
    i = l + 1
    FOR j IN [l + 1 ..< r]
        IF a[j] < pivot_value
            swap(a, i, j)
            i = i + 1
    swap(a, l, i - 1)
    RETURN i - 1
Partition Proof

**Initialization:** Show that the loop invariant is true before the loop starts

1. No numbers in $a[l+1..i-1]$ ≤
2. No numbers in $a[i..j-1]$ ≥

**Loop Invariant:** At the start of the iteration with indices $i$ and $j$:

1. All items in $a[l+1..i-1]$ are < $pivot$
2. All items in $a[i..j-1]$ are ≥ $pivot$

**FUNCTION** Partition($a$, $l$, $r$)

- $pivot_value = a[l]$
- $i = l + 1$

FOR $j$ IN $[l + 1..< r]$

IF $a[j] < pivot_value$

    swap($a$, $i$, $j$)
    $i = i + 1$

swap($a$, $l$, $i - 1$)

RETURN $i - 1$
**Partition Proof**

**FUNCTION** 

```plaintext
Partition(a, l, r)

pivot_value = a[l]
i = l + 1

FOR j IN [l + 1 ..< r]

    IF a[j] < pivot_value
        swap(a, i, j)
i = i + 1

swap(a, l, i - 1)

RETURN i - 1
```

**Loop Invariant:** At the start of the iteration with indices i and j:

1. All items in \(a[l+1 ..= i-1]\) are < \(\text{pivot}\)
2. All items in \(a[i ..= j-1]\) are ≥ \(\text{pivot}\)

**Maintenance (case 1):** Show that the loop invariant holds when executing any iteration

- Suppose conditions 1 and 2 are met.
- Now, suppose \(a[j] < \text{pivot}\)

...
**FUNCTION** Partition(a, l, r)

```
pivot_value = a[l]
i = l + 1
FOR j IN [l + 1 ..< r]
  IF a[j] < pivot_value
    swap(a, i, j)
i = i + 1
```
Partition Proof

FUNCTION Partition(a, l, r)
    pivot_value = a[l]
    i = l + 1
    FOR j IN [l + 1 ..< r]
        IF a[j] < pivot_value
            swap(a, i, j)
            i = i + 1
        true
    swap(a, l, i - 1)
    RETURN i - 1

Maintenance (case 1):
- Suppose conditions 1 and 2 are met.
- Now, suppose a[j] < pivot
- Then a[j] and a[i] are swapped
- By (2), a[i] was > pivot so now a[i] < pivot and a[j] > pivot

Loop Invariant: At the start of the iteration with indices i and j:
1. All items in a[l+1 ..= i-1] are < pivot
2. All items in a[i ..= j-1] are ≥ pivot
**Partition Proof**

**FUNCTION** `Partition(a, l, r)`

- `pivot_value = a[l]`
- `i = l + 1`
- `FOR j IN [l + 1 ..< r]`
- `true` → IF `a[j] < pivot_value`
- `swap(a, i, j)`
- `i = i + 1`
- `swap(a, l, i - 1)`
- `RETURN i - 1`

**Loop Invariant:** At the start of the iteration with indices `i` and `j`:
1. All items in `a[l+1 ..= i-1]` are `< pivot`
2. All items in `a[i ..= j-1]` are `≥ pivot`

**Maintenance (case 1):**
- Suppose conditions 1 and 2 are met.
- Now, suppose `a[j] < pivot`
- Then `a[j]` and `a[i]` are swapped
- By (2), `a[i]` was `> pivot` so now `a[i] < pivot` and `a[j] > pivot`
- Incrementing `i` and `j` satisfies 1 and 2
FUNCTION Partition(a, l, r)
  pivot_value = a[l]
  i = l + 1
  FOR j IN [l + 1 ..< r]
    IF a[j] < pivot_value
      swap(a, i, j)
      i = i + 1
    swap(a, l, i - 1)
  RETURN i - 1

Maintenance (case 2):
• Suppose conditions 1 and 2 are met.
• Now, suppose a[j] ≥ pivot

Loop Invariant: At the start of the iteration with indices i and j:
1. All items in a[1 ..= i-1] are < pivot
2. All items in a[i ..= j-1] are ≥ pivot
**FUNCTION**  \( \text{Partition}(a, l, r) \)

\[
pivot\_value = a[l] \\
i = l + 1 \\
\text{FOR } j \text{ IN } [l + 1 .. < r] \\
\text{IF } a[j] < pivot\_value \\
\quad \text{swap}(a, i, j) \\
\quad i = i + 1 \\
\quad \text{swap}(a, l, i - 1) \\
\text{RETURN } i - 1
\]

**Loop Invariant:** At the start of the iteration with indices \( i \) and \( j \):
1. All items in \( a[l+1 ..= i-1] \) are \(< pivot\)
2. All items in \( a[i ..= j-1] \) are \(\geq pivot\)

**Maintenance (case 2):**
- Suppose conditions 1 and 2 are met.
- Now, suppose \( a[j] \geq pivot\)
- We do not change \( i \) so (1) holds

**Diagram:**
- Partition Proof
- Maintenance (case 2):
FUNCTION Partition(a, l, r)

pivot_value = a[l]
i = l + 1

FOR j IN [l + 1 ..< r]

IF a[j] < pivot_value
    swap(a, i, j)
i = i + 1

swap(a, l, i - 1)

RETURN i - 1

Loop Invariant: At the start of the iteration with indices i and j:
1. All items in a[l+1 ..= i-1] are < pivot
2. All items in a[i ..= j-1] are ≥ pivot

Maintenance (case 2):
• Suppose conditions 1 and 2 are met.
• Now, suppose a[j] ≥ pivot
• We do not change i so (1) holds
• We increment j so (2) holds
Termination: Show that the loop invariant holds once the loop ends

- Now $j = r$
- All items have been considered
- All items in $a[l+1 ..= i-1]$ are $< \text{pivot}$
- All items in $a[i ..= j-1]$ are $\geq \text{pivot}$

Loop Invariant: At the start of the iteration with indices $i$ and $j$:
1. All items in $a[l+1 ..= i-1]$ are $< \text{pivot}$
2. All items in $a[i ..= j-1]$ are $\geq \text{pivot}$

FUNCTION Partition(a, l, r)

\[
pivot\_value = a[l] \\
i = l + 1 \\
\text{FOR } j \text{ IN } [l + 1 .. < r] \\
\text{IF } a[j] < pivot \_\text{value} \\
\quad \text{swap}(a, i, j) \\
\quad i = i + 1 \\
\text{swap}(a, l, i - 1) \\
\text{RETURN } i - 1
\]
FUNCTION Partition(a, l, r)  
  pivot_value = a[l]  
  i = l + 1  
  FOR j IN [l + 1 ..< r]  
      IF a[j] < pivot_value  
          swap(a, i, j)  
          i = i + 1  
  swap(a, l, i - 1)  
  RETURN i - 1

Loop Invariant: At the start of the iteration with indices i and j:
1. All items in a[l+1 ..= i-1] are < pivot
2. All items in a[i ..= j-1] are ≥ pivot

After the loop we do the final swap
What do we need to do?

1. Prove that PARTITION works
   • **Proof by loop invariant**

2. Prove that Quicksort works
   • Proof by induction
Proof by Induction in General

Some property $P$ that we want to prove
- A base case: some statement regarding $P(1)$
- An inductive hypothesis: assume we know that $P(n)$ is true
- An inductive step: if $P(n)$ is correct then so is $P(n+1)$ because...

For quicksort we are going to use a slightly different form
- If $P(k)$ where $k < n$ is correct, then $P(n)$ is also correct
- An inductive hypothesis: assume we know that $P(k)$ is true
- An inductive step: if $P(k)$ is correct then so is $P(n)$ because...
Proof by Induction Cheat-sheet

Proof by induction that $P(n)$ holds for all $n$

1. $P(1)$ holds because <something about the code/problem>
2. Let’s assume that $P(k)$ (where $k < n$) holds.
3. $P(n)$ holds because of $P(k)$ and <something about the code>
4. Thus, by induction, $P(n)$ holds for all $n$
Quicksort Proof

P(n) = Quicksort is always correct for arrays of length n.

• P(1) is an array of one element, and any such array is always sorted.
• Assume (hypothesis) that P(k) is correct for k < n
• P(n) holds because:
Proof by induction that $P(n)$ holds for all $n$
- $P(1)$ holds because ...
- Let’s assume that $P(k)$ (where $k < n$) holds.
- $P(n)$ holds because of $P(k)$ and ...
- Thus, by induction, $P(n)$ holds for all $n$

$P(n) = \text{Quicksort is always correct for arrays of length } n.$

- $P(1)$ is an array of one element, and any such array is always sorted.
- Assume (hypothesis) that $P(k)$ is correct for $k < n$
- $P(n)$ holds because:
  - Let $k_{\text{left}}, k_{\text{right}}$ = the lengths of the left and right subarrays
  - $k_{\text{left}}, k_{\text{right}} < n$ (strictly less than $n$)
  - By our \textbf{inductive hypothesis}, the left and right subarrays are correctly sorted
  - The partition loop-invariant guarantees that the pivot is in the correct spot
Quicksort Proof

P(n) = Quicksort is always correct for arrays of length n.

• P(1) is an array of one element, and any such array is always sorted. **Base case**

• Assume (hypothesis) that P(k) is correct for k < n **Inductive Hypothesis**

• P(n) holds because:
  • Let \( k_{\text{left}}, k_{\text{right}} \) = the lengths of the left and right subarrays
  • \( k_{\text{left}}, k_{\text{right}} < n \) (strictly less than n)
  • By our hypothesis, the left and right subarrays are correctly sorted
  • The partition loop-invariant guarantees that the pivot is in the correct spot **Inductive Step**

Proof by induction that P(n) holds for all n
• P(1) holds because ...
• Let’s assume that P(k) (where k < n) holds.
• P(n) holds because of P(k) and ...
• Thus, by induction, P(n) holds for all n
What do we need to do?

1. Prove that PARTITION works
   • Proof by loop invariant

2. Prove that Quicksort works
   • Proof by induction