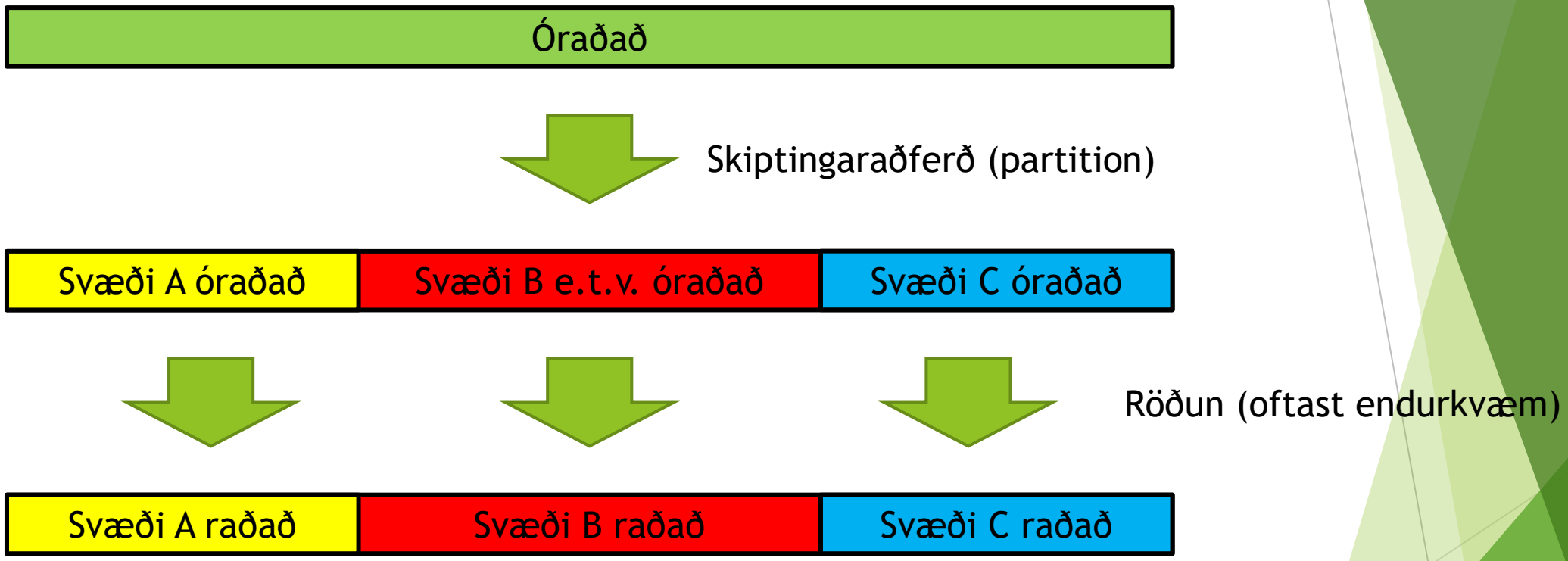
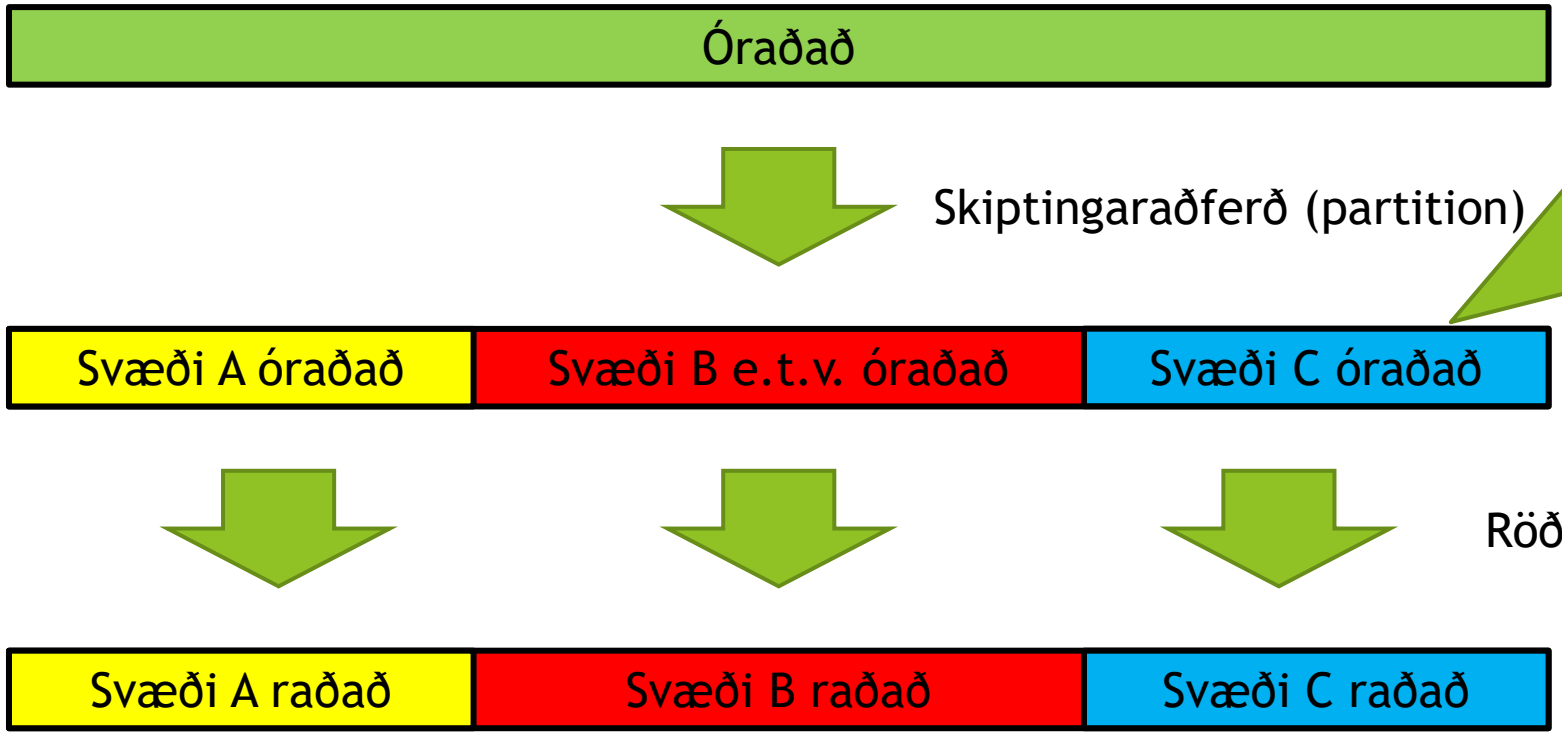
Quicksort

Úrval afbrigða

Grunnhugmynd



Grunnhugmynd



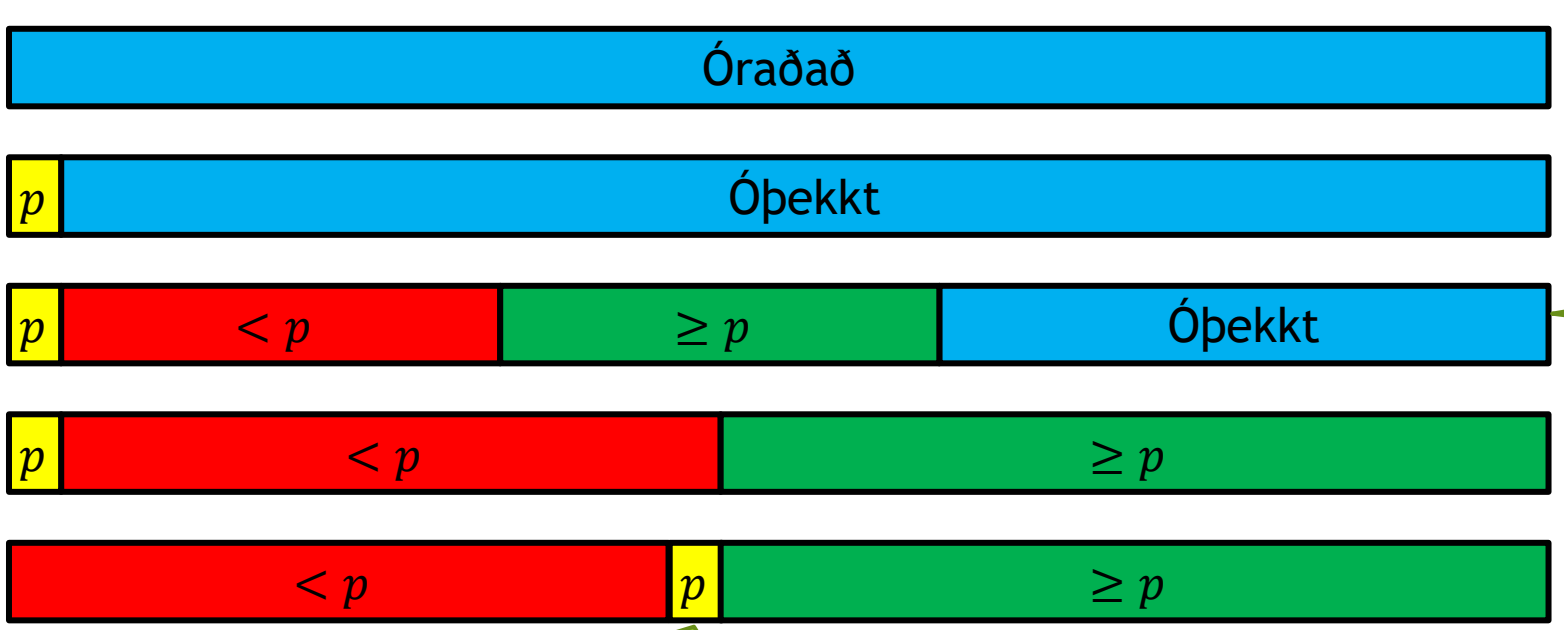
Nauðsynlegt er að $A \leq B \leq C$, þ.e. að fyrir öll fyrir gildi $a \in A$, $b \in B$, $c \in C$ sé $a \leq b \leq c$.

Svæði má vera tómt, en a.m.k. tvö svæði verða að vera ekki tóm.

Ýmsar skiptingaraðferðir

- Lomuto
  - Einfaldasta aðferðin og er mikið notuð í kennslu
  - Hryllilega hægvirk fyrir mörg jöfn gildi
- Dijkstra
  - Meðhöndlar vel röðun á mörgum jöfnum gildum
- Hoare
  - Upphaflega aðferðin skilgreind af C.A.R. Hoare 1959-1961
  - Ennþá notuð því hún er hraðvirk og meðhöndlar oftast slæm tilvik vel
- Tvö vendigildi
  - Dual-pivot aðferð Yaroslavskiy, Bentley og Bloch var á tímabili í Java klasasafninu
  - Tvö vendigildi sameina kosti Hoare og Dijkstra
- Þrjú vendigildi eru einnig möguleg á tiltölulega einfaldan hátt
- Fleiri en þrjú vendigildi eru erfið í meðhöndlun

Aðferð Lomuto



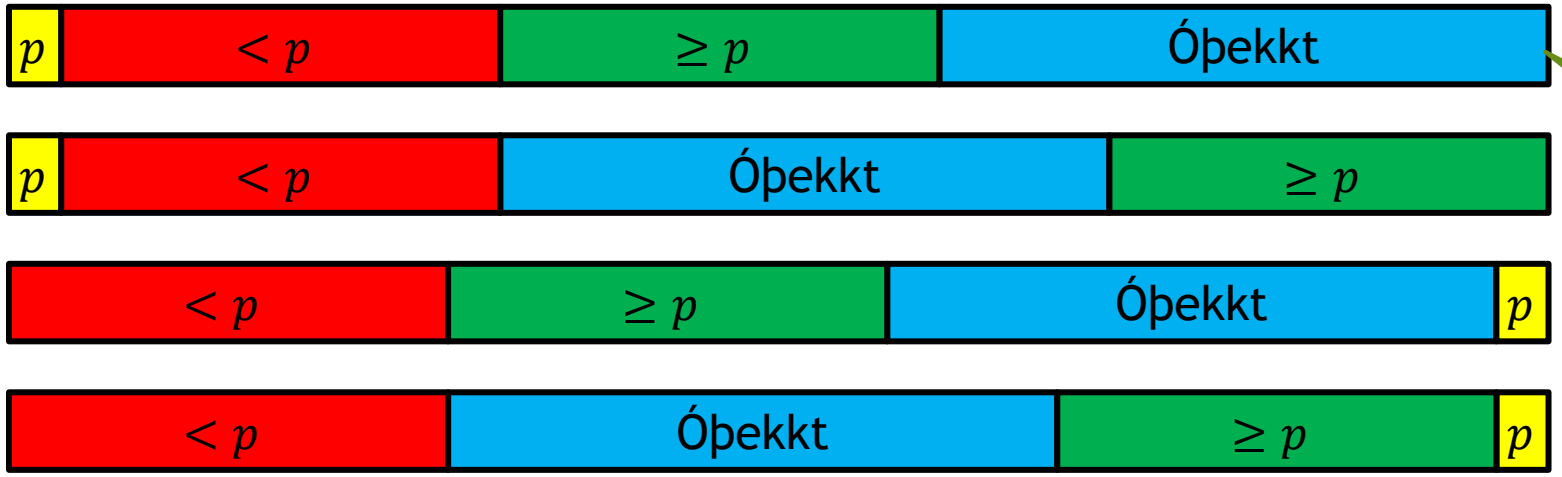
Frumstilling

Fastayrðing lykkju

Eftirmeðhöndlun

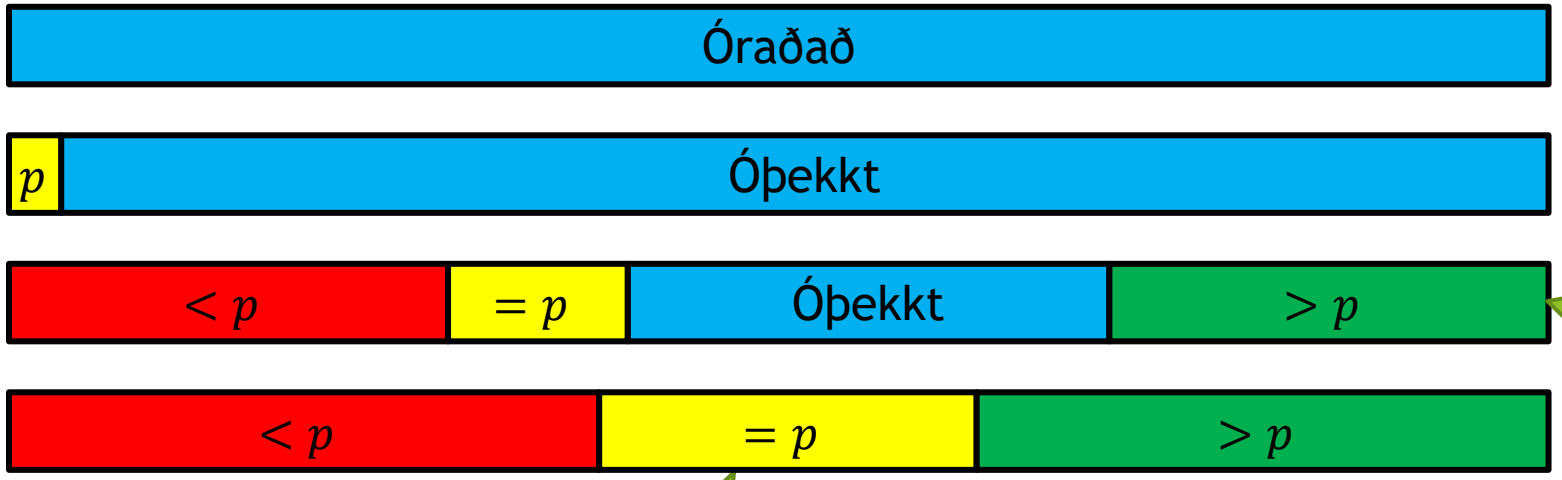
Óheppilegt ástand ef fá eða engin gildi eru $< p$.
Til dæmis ef öll gildi eru jöfn í fylkinu eða ef p er minnst.
Þá mun þetta afbrigði quicksort hegða sér svipað og selection sort.

Svipaðar fastayrðingar



Lomuto

Aðferð Dijkstra (Dutch National Flag)

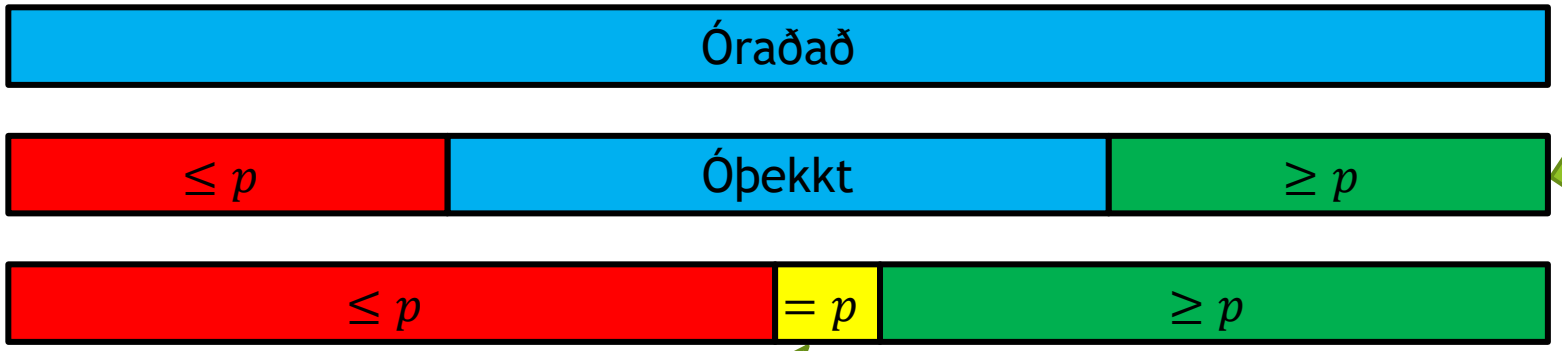


Fastayrðing lykkju.

Athugið að það þarf oft tvo samanburði í umferð lykkjunnar til að viðhalda fastayrðingunni.

Gott ástand ef mörg gildi eru $= p$.

Aðferð Hoare, fastayrðing ytri lykkju



Miðjusvæðið, $= p$, er svæðið sem er í lokin bæði rautt ($\leq p$) og grænt ($\geq p$).

Það verður annaðhvort tómt eða inniheldur eitt sæti.

Ef öll gildi í fylkinu eru $= p$ fer helmingur þeirra í rauða svæðið, helmingur í græna og ef fjöldinn er oddatala fer eitt gildi í gula svæðið.

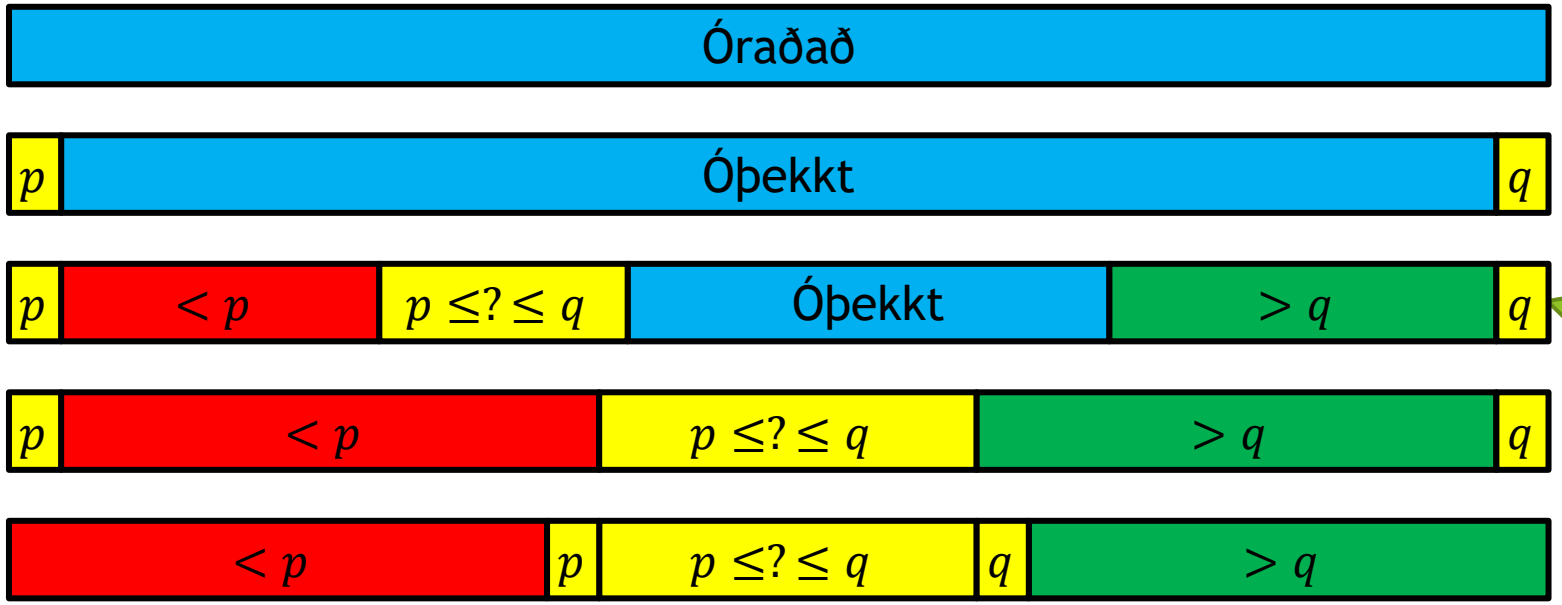
Fastayrðing lykkju.
Athugið að myndin gefur ekki tæmandi upplýsingar um fastayrðinguna.

Einnig þarf að tryggja að annaðhvort séu bæði endasvæðin tóm eða hvorugt.

Auk þess þarf að tryggja að p sé einhvers staðar í fylkinu.

Inni í ytri lykkjunni eru tvær þrumuhraðvirkar innri lykkjur sem annarsvegar stækka fremsta svæðið með gildum $< p$ og hins vegar stækka aftasta svæðið með gildum $> p$. Eftir báðar lykkjur má stækka bæði svæðin um eitt sæti hvort með einni víxlun.

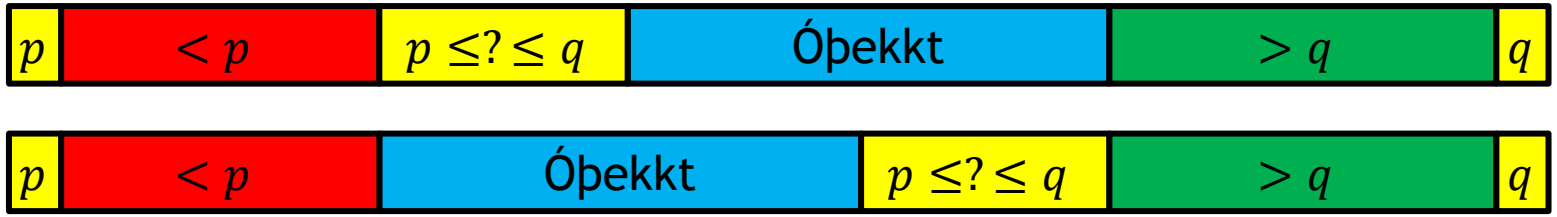
Aðferð með tveimur vendigildum



Fastayrðing lykkju.

Athugið að það þarf oft tvo samanburði í umferð lykkjunnar til að viðhalda fastayrðingunni.

Afbrigði fastayrðinga með tveimur vendigildum

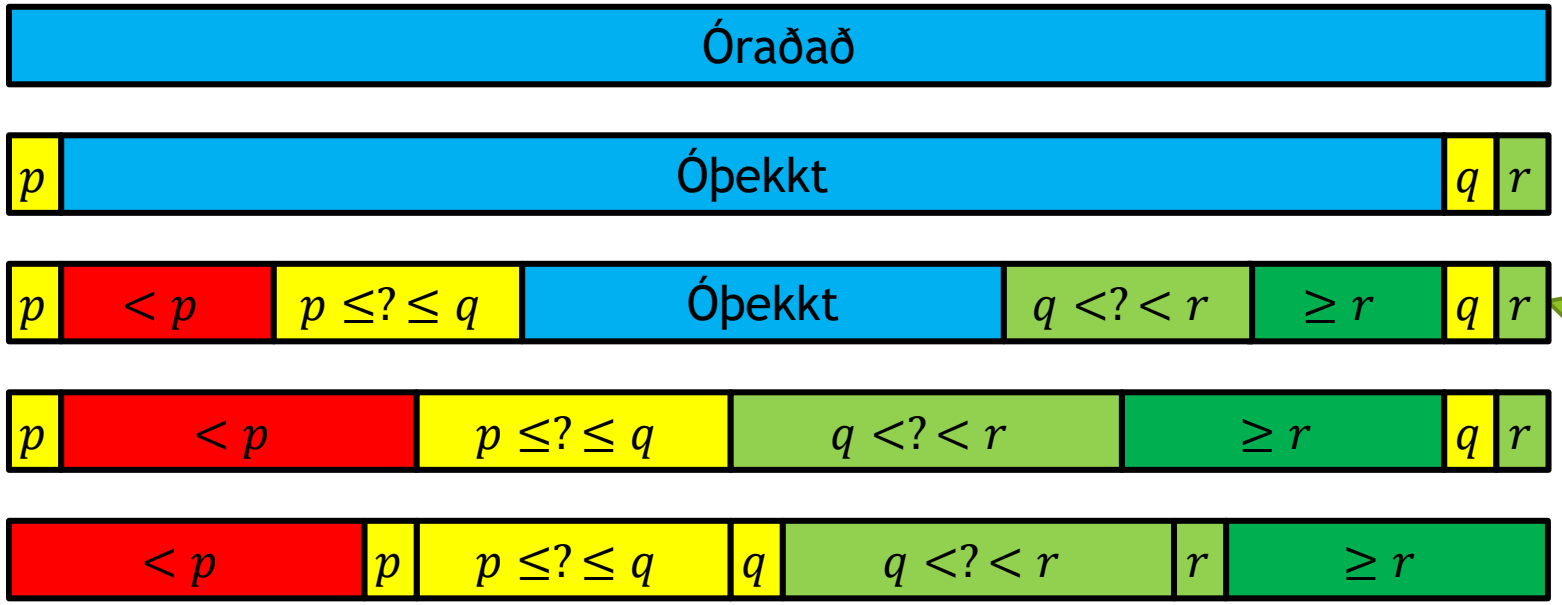


- Augljóslega eru fleiri afbrigði þar sem p og q eru staðsett öðruvísi á öðrum hvorum enda.
- Val á afbrigði getur oft ráðist af möguleikum á að kreista hraðann með einföldunum á meðhöndlun vísa.
- Yaroslavskiy, Bentley og Bloch hafa þróað þrumuhraðvirk quicksort í Java með báðum þessum afbrigðum (seinni myndin er sú sem þeir nota í nýjustu útgáfu, frá janúar 2018).

Yaroslavskiy, Bentley & Bloch, nýrri útgáfa, sótt af vefnum:

```
/*
 * Backwards 3-interval partitioning
 *
 *   left part                  central part          right part
 * +------------------------------------------------------------+
 * |  < pivot1  |   ?   |  pivot1 <= && <= pivot2  |  > pivot2  |
 * +------------------------------------------------------------+
 *              ^       ^                          ^
 *              |       |                          |
 *            lower     k                        upper
 *
 * Invariants:
 *
 *                all in (low, lower] < pivot1
 *    pivot1 <=   all in (k, upper)  <= pivot2
 *                all in [upper, end) > pivot2
 *
 * Pointer k is the last index of ?-part
 */
```

Aðferð með þremur vendigildum



Fastayrðing lykkju.

Það þarf mest tvo samanburði í hverri umferð lykkjunnar til að viðhalda fastayrðingunni ef við byrjum á að bera saman við q.

Þrjú vendigildi hafa ekki gefið sérlega hraðvirka röðun, hingað til. Þó er ekki vert að afskrifa hugmyndina því hraðinn fæst oft með því að kreista meðhöndlun vísanna sem skilgreina svæðin. Einnig er aðferðin skyndiminniskær.

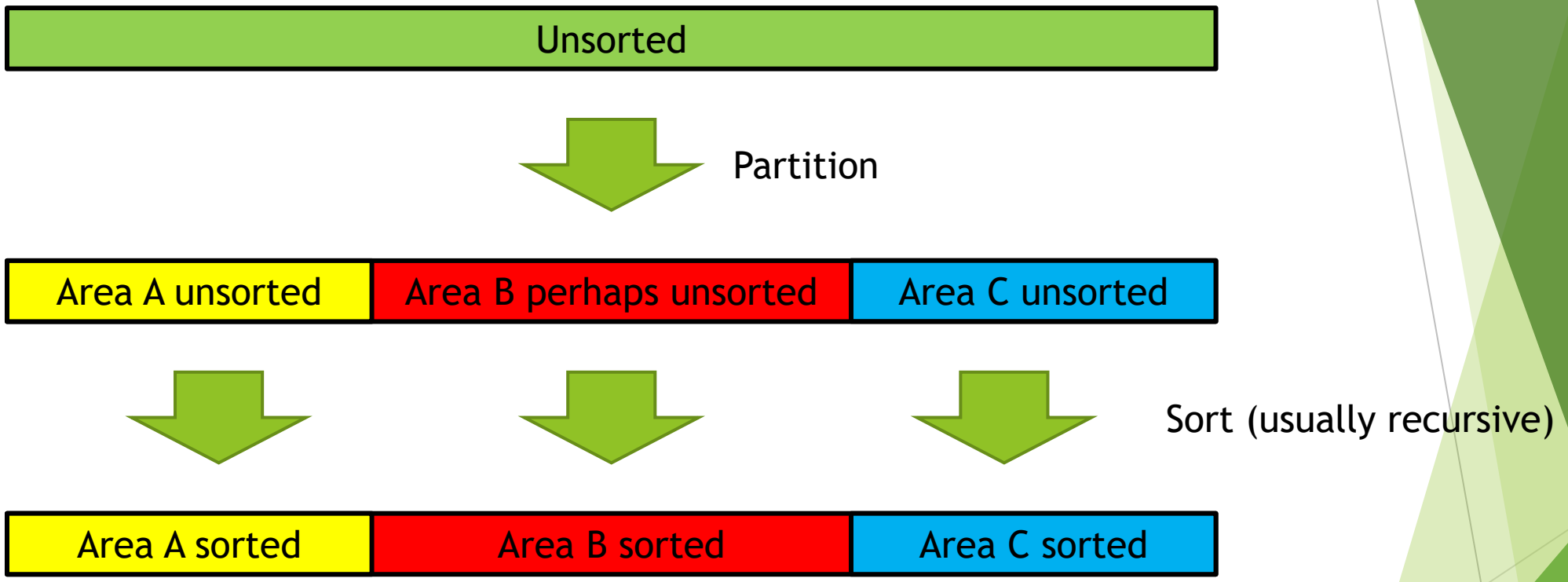
Afbrigði með þremur vendigildum

Fleiri afbrigði koma til greina. Afbrigðin geta hegðað sér mjög mismunandi í tilvikum þegar $p = q$ eða $q = r$. Takið til dæmis eftir því að fyrsta og síðasta myndin gefa mjög slæma skiptingu ef öll gildi í fylkinu eru jöfn. Þá er $p = q = r$ og öll gildin fara í fremsta eða aftasta svæðið eftir því hvor myndin er í gildi. Miðmyndin gefur hins vegar líklega góða skiptingu ef í lokin er athugað hvort $p = q$ og sleppt því að raða gula svæðinu ef svo er.

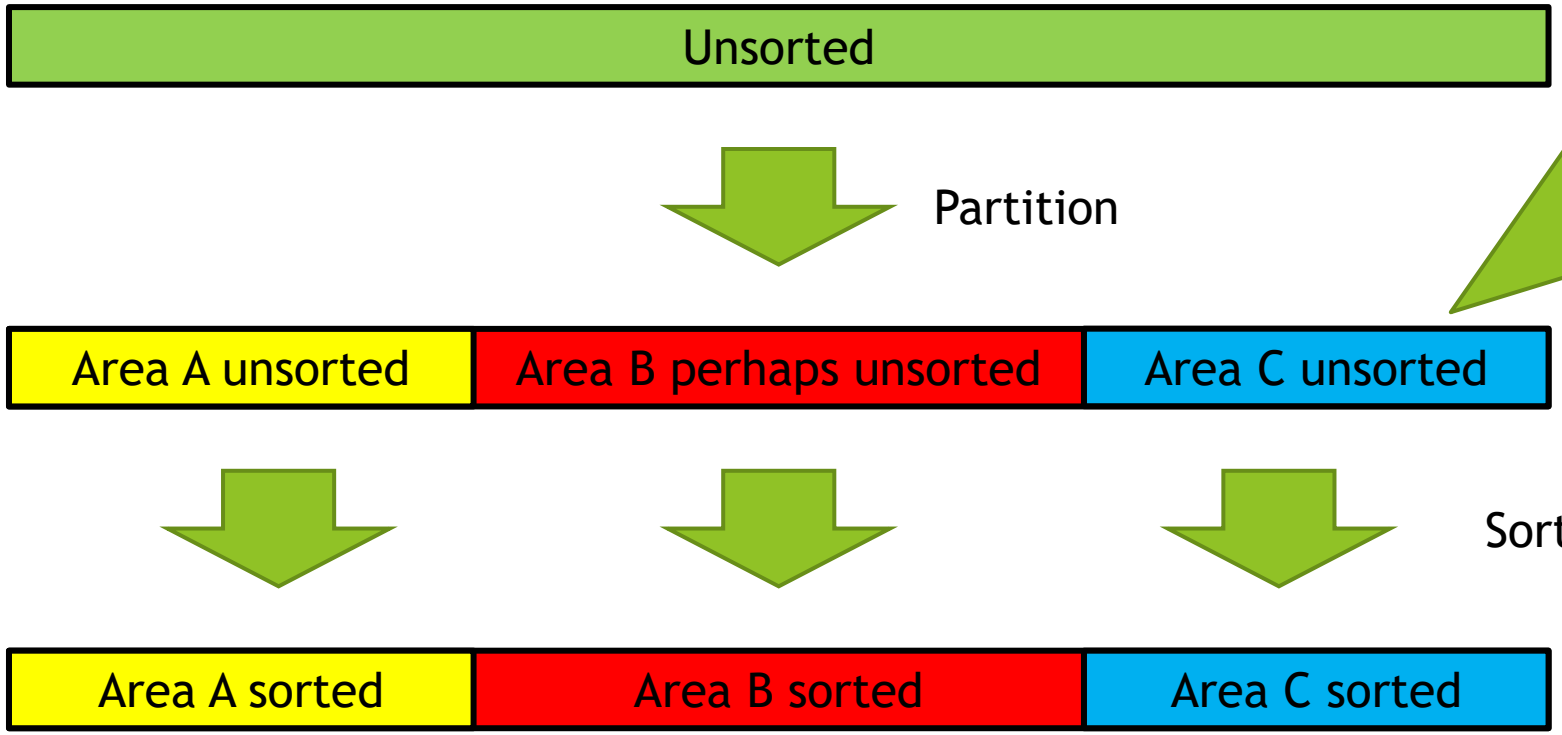
Quicksort

A Selection of Variants

Fundamental Idea



Fundamental Idea



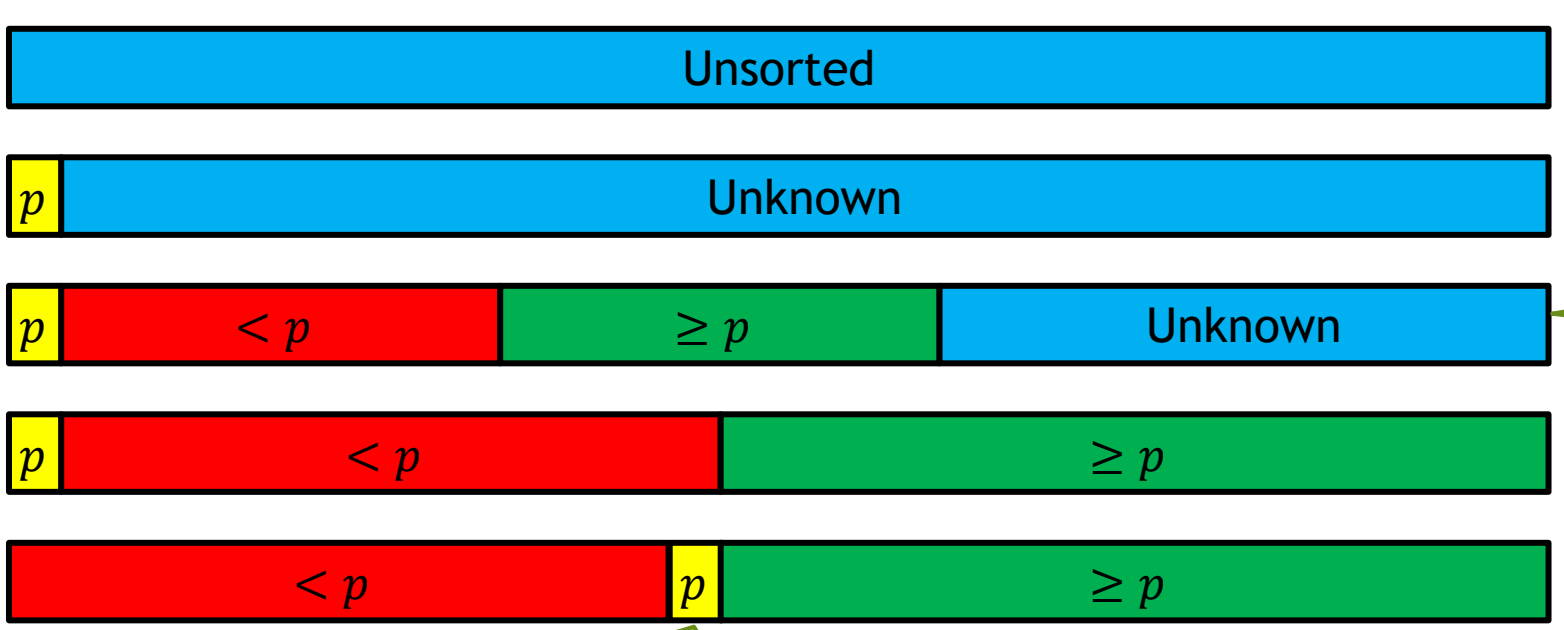
It is necessary that
$A \leq B \leq C$
i.e. that að for all values
$a \in A,\ b \in B,\ c \in C$
we have $a \leq b \leq c$.

An area may be empty, but at least two areas must be non-empty.

Various partitioning methods

- Lomuto
  - The simplest method and most common when teaching
  - Horribly slow for many equal values
- Dijkstra
  - Handles well sorting many equal values
- Hoare
  - The original method defined by C.A.R. Hoare 1959-1961
  - Still used becaue it is fast and usually handles bad cases well
- Two pivots
  - Dual-pivot method from Yaroslavskiy, Bentley and Bloch was for a while in the Java class library
  - Two pivots unify the advantages of Hoare and Dijkstra
- Three pivots are also possible in a fairly simple manner
- More than three pivots are hard to implement

Lomuto partitioning



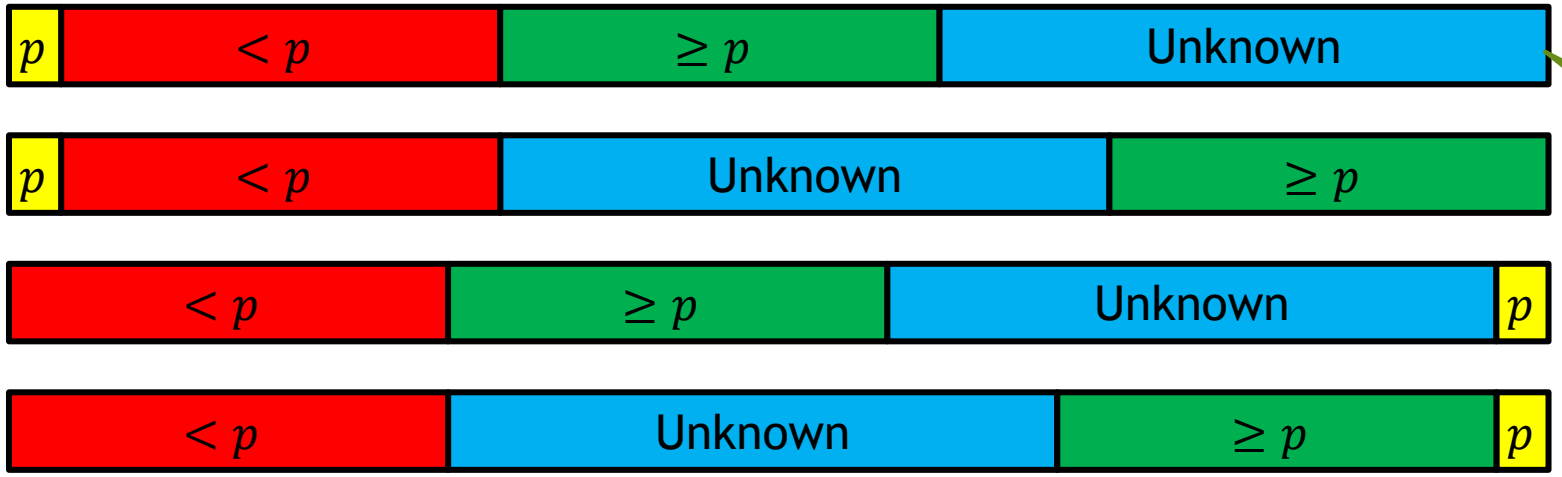
Initialization

Loop invariant

Postprocessing

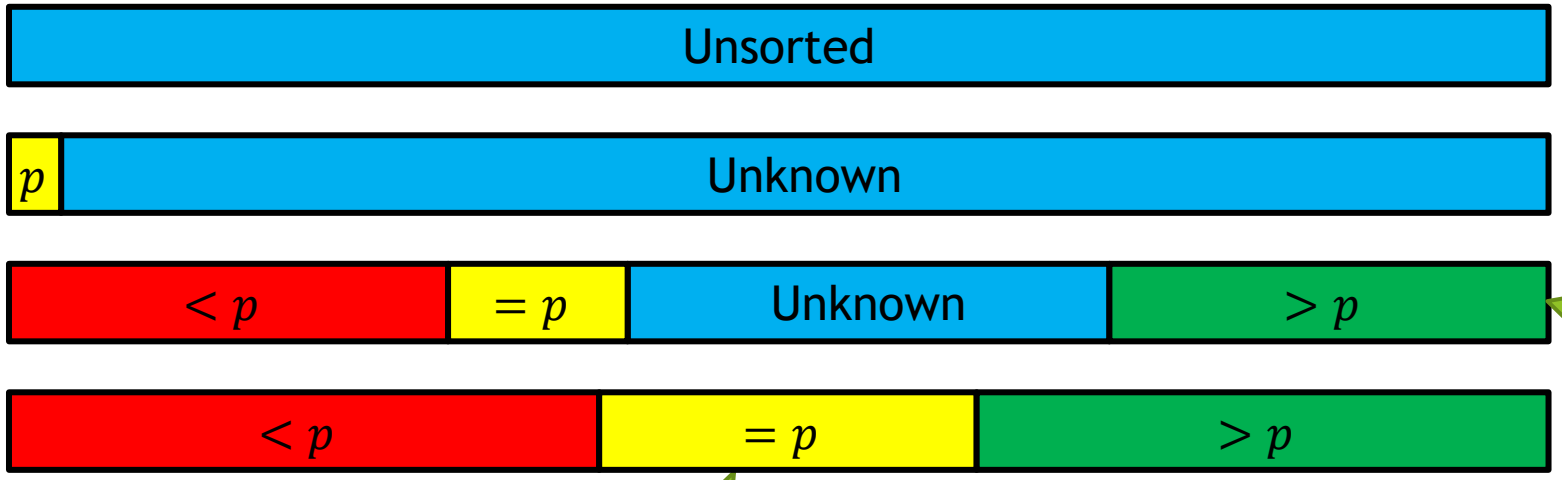
Unfortunate state if few or no values are $< p$.
For example if all the values are equal in the array or if p is smallest.
Then this variant of quicksort will behave similarly to selection sort.

Similar invariants



Lomuto

Dijkstra‘s method (Dutch National Flag)

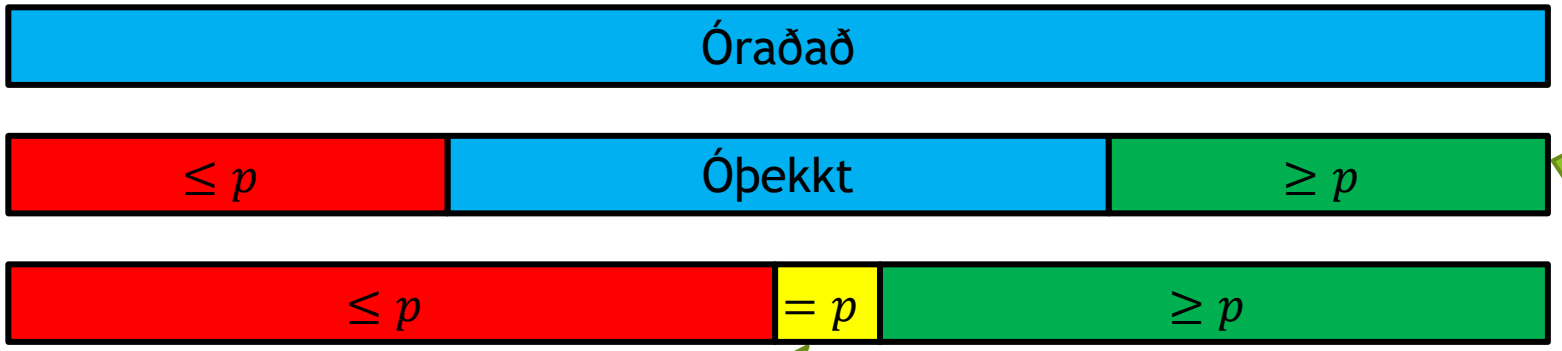


Loop invariant.

We sometimes need two comparisons in a pass through the loop in order to maintain the invariant.

Good state if many values are $= p$.

Hoare method, invariant of outer loop



The middle area, $= p$, is the area that in the end is both red ($\leq p$) and green ($\geq p$).

It will either be empty or contain one position.

If all values in the array are $= p$ then half of them will og to the red area, half to the green area, and if the number of them is odd then one will go to the yellow area.

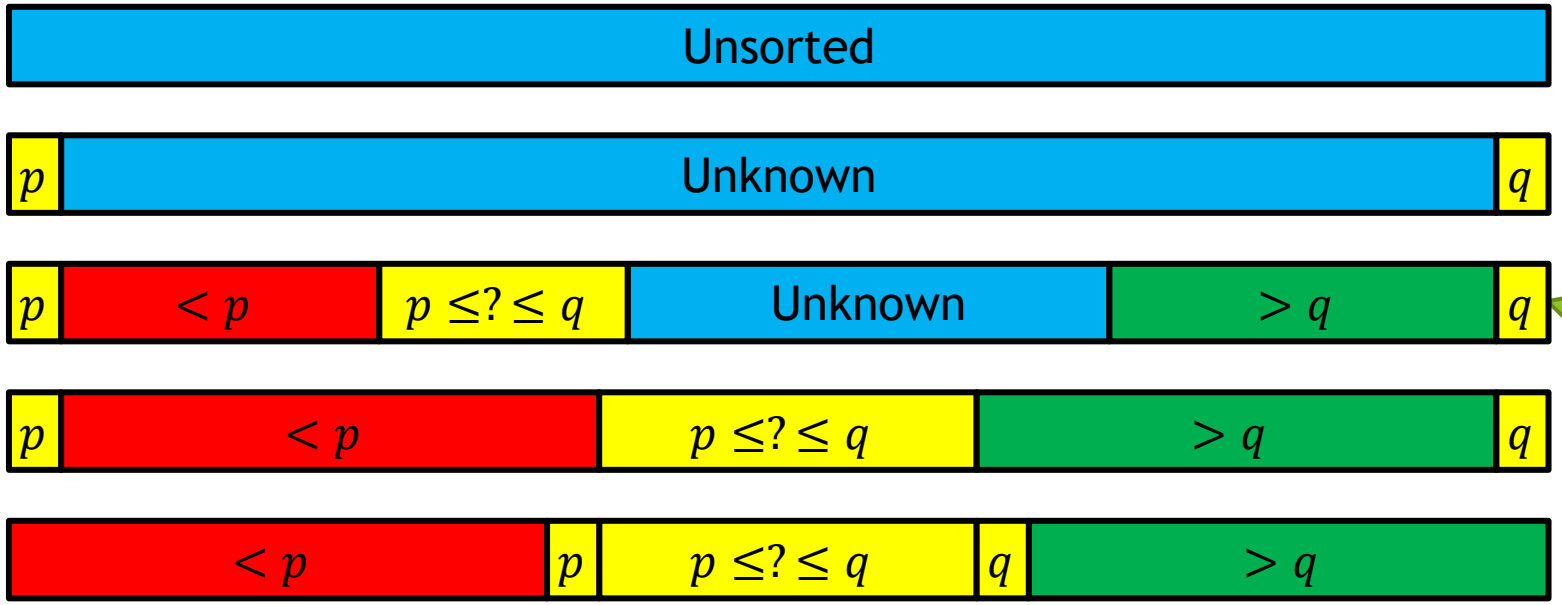
Loop invariant.
The picture does not completely describe the loop invariant.

Also need to ensure that either both end areas are empty or nether of them is.

Also need to ensure that p exists in the array.

Inside the outer loop there are two super-fast inner loops that increase the first area with values $< p$ and increase the last area with values $> p$. After both loops, both the areas can be increased with one swap.

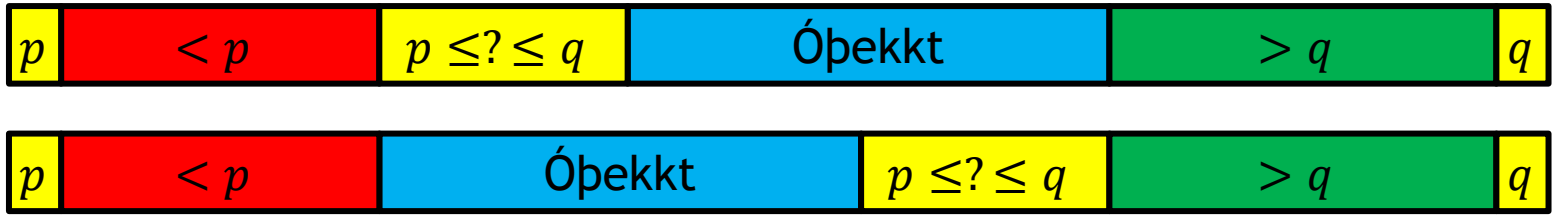
Method with two pivots



Loop invariant.

We sometimes need two comparisons to maintain the loop invariant.

Variations with two pivots

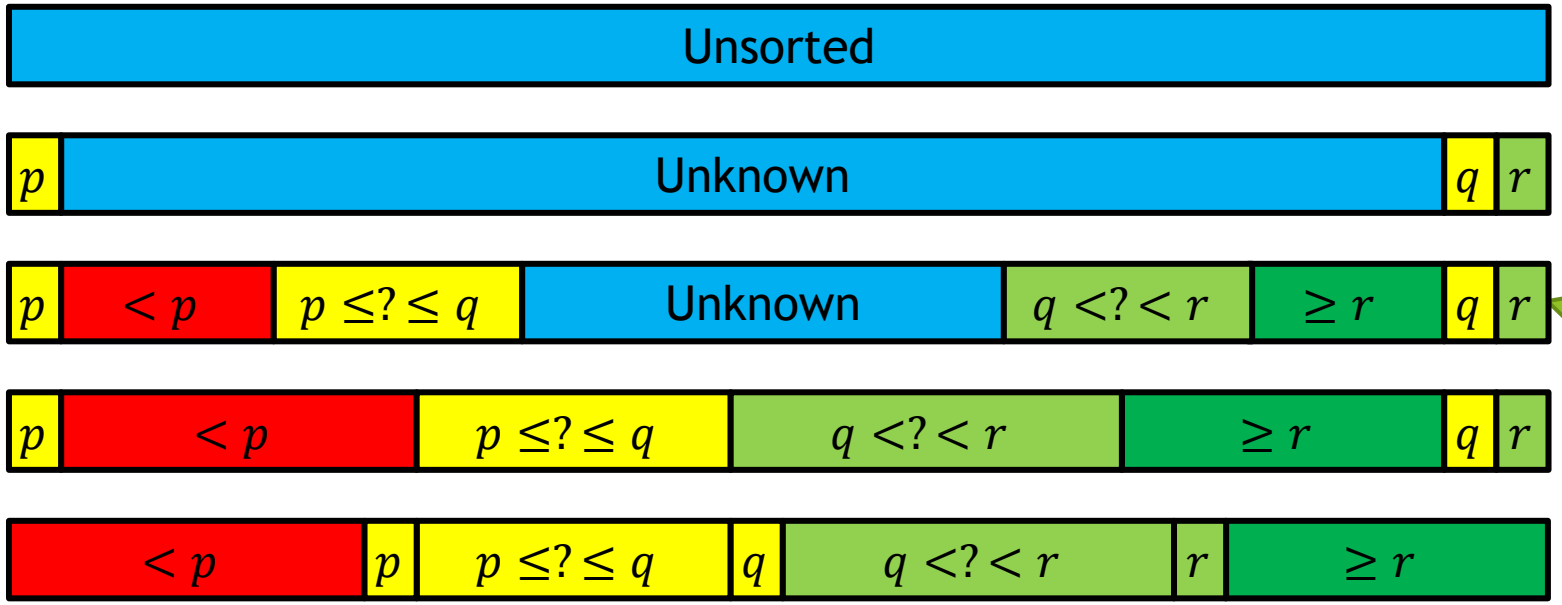


- There are more variants where p and q are differently located on either end.
- The choice of variant is often determined by opportunities to squeeze the speed with simplifications of index manipulations.
- Yaroslavskiy, Bentley and Bloch have developed a super-fast quicksort with both these variants (the second picture is what they use in the newest version, from January 2018).

Yaroslavskiy, Bentley & Bloch, newer version:

```
/*
 * Backwards 3-interval partitioning
 *
 *   left part                  central part          right part
 * +------------------------------------------------------------+
 * |  < pivot1  |   ?   |  pivot1 <= && <= pivot2  |  > pivot2  |
 * +------------------------------------------------------------+
 *              ^       ^                          ^
 *              |       |                          |
 *            lower     k                        upper
 *
 * Invariants:
 *
 *                all in (low, lower] < pivot1
 *    pivot1 <=   all in (k, upper)  <= pivot2
 *                all in [upper, end) > pivot2
 *
 * Pointer k is the last index of ?-part
 */
```

A method with three pivots

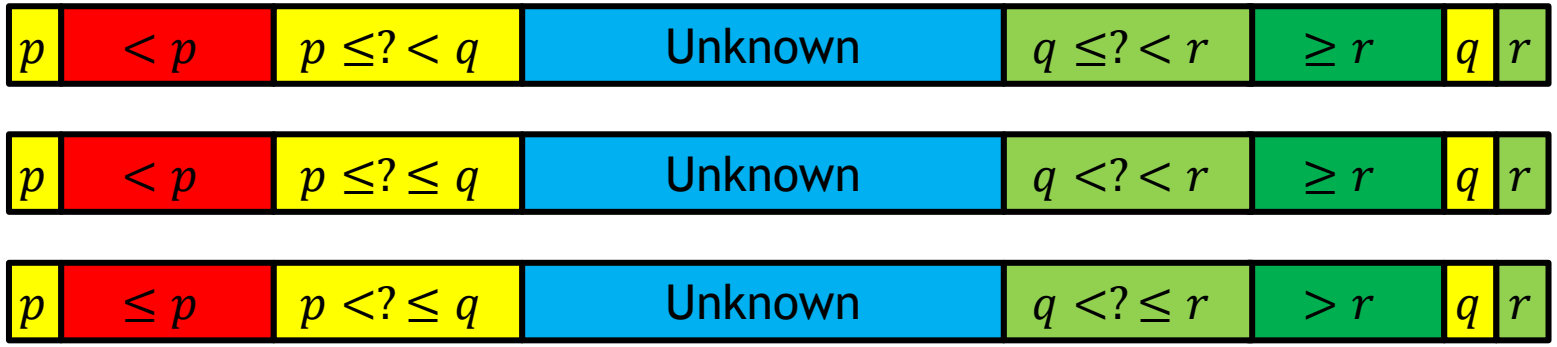


Loop invariant.

We need at most two comparisons per pass through the loop to maintain the invariant if we start by comparing to q.

Three pivots have so far not given a super-fast sorting function. Still, we should not write the idea off because speed us often achieved by squeezing the manipulations of indexes that define the areas. Also, it is cache-friendly.

Three pivot variants



More variants are possible. The variants can behave very differently in cases where $p = q$ or $q = r$. Note for example that the first and last pictures give a very bad partition if all values in the array are equal. Then $p = q = r$ and all the values go to the first or the last area, depending on which picture is being used. The middle picture, on the other hand, is likely to give a good partition if we end by checking whether $p = q$ and refrain from sorting the yellow area if so.