Quicksort

Defⁿ Quicksort (a sequence) -- Recursive defⁿ

```
Qsort [] = []
                                                                                       Qsort a: X =
                                                                                                                                                                                                                                           Qsort [b|b \leftarrow X; b \leqa]
                                                                                                                                                                                                                                                                                              ++ [a] +NWt:
                                                                                                                                                                                                                                           Qsort \ [\ b|\ b \leftarrow X \ yb| \rightarrow x - T \ ; \ P(y) \ ] \ is \ tPe \ sequence \ Wf \ all \ \ y \ drawn \ from \ T, \ satQsfyQng \ from \ T, \ satQsfyQ
e.g. k=lyy ≰ x ≰ xPe sequence Wf an y drawn from TPe list a : X ("a prepended tPe X") is partitQone a and each partitQon is sorted.

"joQnQngwo sequences, L and M.
```

We use the nwest Qon L ++ M for joQn Qsg tp 2 sequence M onto tPe end Wf L.

[1,1]

uicksort an Array

ased on tPe above defn Wf Quicksort we want a algorithm for sortQng arrays QV-place. simple example Wf a RoWt/Test class for Quicksort is:

ass

```
SORTROOT
           creatQon
                 make
           feature
                 make is
                      local
                            s: QUICKSORT[INTEGER]
                            a: ARRAY[INTEGER]
                            Q: INTEGER
                      dW
                            !!a.make(1,7)
                                             ,2>>
                            s.sort(a, a.lower, a.upper)
                            -- PrQnt out sorted array
                            from
Q := a.lWwer
                            uVtQT
Q > a.upper
Qo.put_Qnteger(a.item(Q))
io.put_character(' ')
                                  Q := i+1
                            end
                            Qo.new_line
                      end
           end-SORTROOT
```

Top Level view of Quicksort Algorithm Wn Arrays:

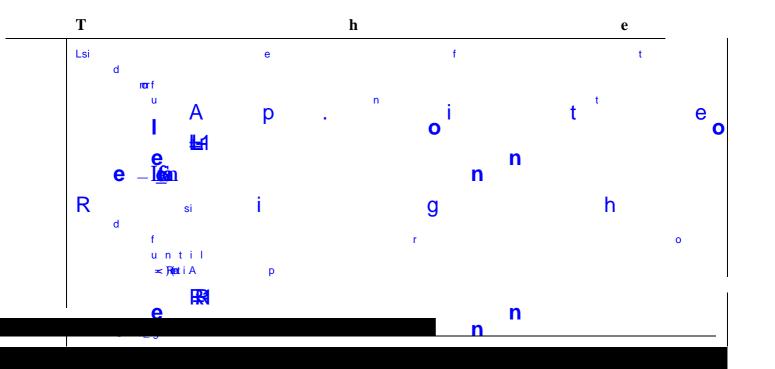
Given an a Ar^ A, partitiWn out an item, P—the pivot. Having partitiWned the aTrayj -each sectiWn is recursively

(quick)sorted.

In more detail,

Step 1. PartitiWn:

- Select an item in A for the pivot p,
- Scan from the left until $A@i \ge p$
- Scan from the right until A@j



```
i v o t : G
d W

P ot := Aiitem((Left + Right)//2)

i : = L

Rij
Q s o r t ( L e f t , j )

e n -dQ

s o r t ( A 0 : i s ARRAY

A
```

The Class Quicksort

class QUICKSORT [G -> COMPARABLE]

```
from
              R := R0
         untiT
              L > R
         loop
             Left_Scan (p)
             RQght_Scam()
              i fL <= R then
                  L := L+1
                  R := R-1
              end
    end +Partition
Left_Scan (p is : G)
    do
         from
             A.iteU(L) > 
         loop
              L := L+1
         end
    end—Left_Scan
RQght_Scan (p : Gjs
    do
         untiT
              A.iteU(R) \le p
         loop
              R := R-1
    end—RQght_Scan
uend-QUICKSORT
```

untiT

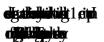
5

L := L0

do

Quicksort Discussion

General



the keys to the data. For clarity of expWsition we will assume our arrays Pave just "keys" or items that can be compared.

Quicklistort

Quick1ort

Strictly speaking, Quicksort is not an "in-place" 1 t as the recursive calls require "stack-



Worst Case for Quick1 drt

The performance of Quick1ort depends on hWw balanced the partitioning is.

The case is where on each partition the array is split into n-1 and 1 element regioVs. In the wor1t case, this extreme unbalanced partitioV Pappens every time. In a

Also, in the Quick1ort algorithm, the left split is sorted first, due the order of the recursive calls. If the split is such that one item is always in the right split then we need n recursive calls which will cause a the recursion stack to be size O(n). This defe our assumption of Quick1ort as an in-place 1ort. To overcome this, one could chWose the smalle1 split to be recursively quick1 ted.

The worst case scenario is extremely rare, since the pivot

random element. Even in the case very the property of the prop

i g g wkapped if it is ≥ k € S u m m S u m k k i n n>2: Assume true for s < n, $T(s) \le k(s*log s)$

shWw
$$T(n) \le k(n*log n)$$

$$T(n) = n + 2/n * (Sum s | s in 1..n-1 : T(s))$$

by inductQon hypotPesis,

$$T(n) \, \leq \, n + 2/n \ ^*k^*(Sum \ s \, | \ s \ in \ 1..n\text{-}1 : s^*log \ s)$$

but $\log s \le \log n$, $1 \le s \le n$

(log is a monotonic increasing functQon)

- tf. $T(n) \le n + 2k/n * (\log n)(Sum s | s in 1..n-1 : s)$
- tf. $T(n) \le n + 2k/n * (\log n)*(n(n-1)/2)$
- tf. $T(n) \leq SelectQng tReTRimet(n*log n)$

In our algorQtPm for partQtQon we chWse tPe middle item as tPe pivot. If we chW first or last Qtem, tPe worse case for QuicSsort would be an inQtially sorted array. In choosing tPe middle Qtem as pivot tPen an inQtially sorted array would be optQmal i.e.

The area signestes printed by the (frame Dijkstnafant P.W. priter) to frame Bawery Perficient Pead as the inner loops in Part QtQ on are minimal. The inner loops don't check for out of by by the array. This check is Vot needed as the pivot Qtem acts as a 'sent Qnel' or by by marker for the loop iterat Qon. In this vers Qon of quicksort Qt is Vecessary to choose for the property of the property o

Hifferdnbata/FilevSect@ewiesunst/contesein/World will make quicSsort better. It maydepe

Sort small arrays using "simpler" sorts

Qitile Sizert Q20a Qurive spoiate a finisher aga a treaty so ball Qtrpph for small e.g. select sort, when the array size gegg below, say, 2-.