COMPSCI 105

Principles of Computer Science

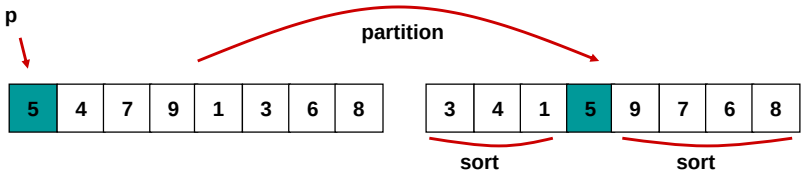
QuickSort

Quicksort

Divide and Conquer

Imagine a set of cards

- Pick a single element 'p'
- Shift the cards so that p is in its final position:
  - all elements less than p are below p
  - all elements greater than p are above p
- Get two friends to sort each half (recursive case)



Instructions

QuickSort

- Accept a pile of cards
- If you were given < 2 cards, then return them (if any), else
  - Partition the cards
  - Keep the selected card, low cards and high cards separate
  - Give low cards to neighbour and wait for it to be returned
  - Add selected card to top of the low cards
  - Give high cards to neighbour and wait for it to be returned
  - Return the sorted pile

Partition

- Accept a pile of cards
- Pick the top card and put aside
  - Divide cards into two piles… cards lower and cards higher than the selected card.
- Return the low cards, selected card, and high cards

Quicksort code

Quicksort

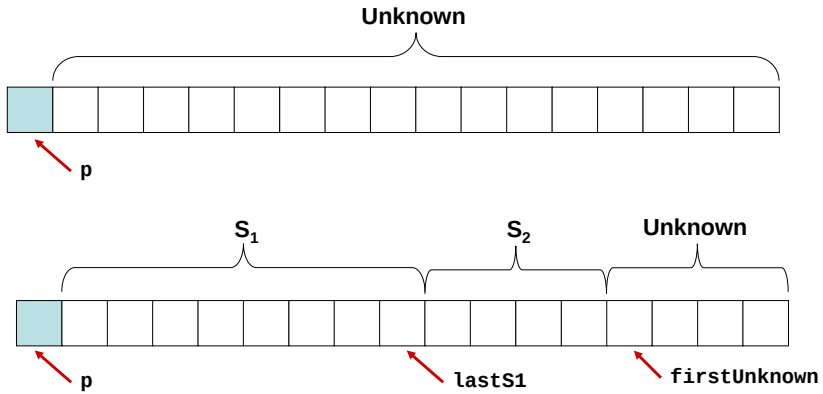
```
if number of elements to sort > 1
    choose a pivot p
    partition elements around pivot p
    quicksort all elements below pivot
    quicksort all elements above pivot
```

```
public void quicksort( int[] data, int first, int last ) {
    if (first < last) {
        int p = partition( data, first, last );
        quicksort( data, first, p - 1 );
        quicksort( data, p + 1, last );
    }
}
```

Partitioning the data

Pivot value starts in the left-most array element

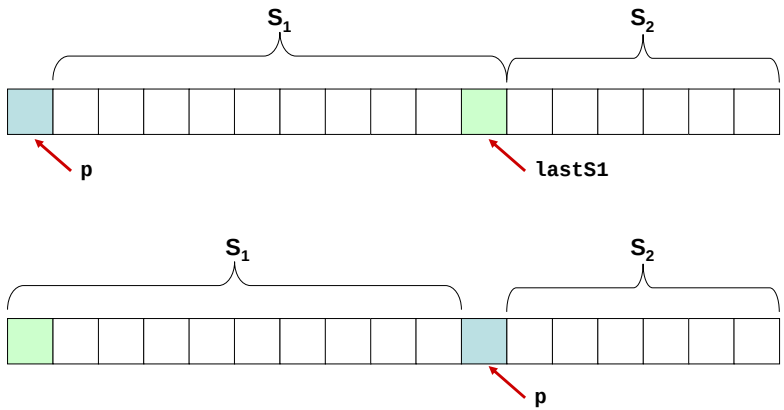
- Take each element from the unknown section
- Put the element in S_1 (elements < p) or S_2 (elements >= p)



Partitioning the data

When there is no remaining unknown elements

- Swap the pivot p with the last element of S_1



Partitioning Pseudocode v2

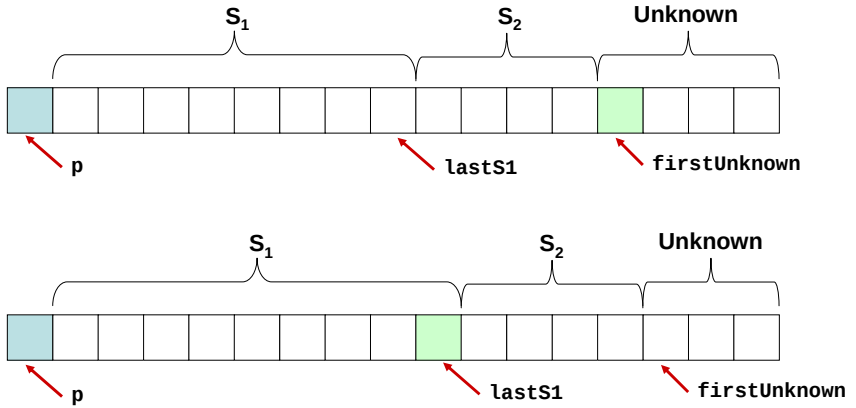
Partition (array, first, last)

```
Choose a pivot value and swap it with the first element
for (firstUnknown = first + 1, firstUnknown <= last, firstUnknown++ ) {
        if ( array[firstUnknown] < p )
                move the element into S1
        else
                move the element into S2
}
swap the pivot with the last element of S1
return the index of the pivot
```


Move the element into S_1

`Move array[firstUnknown] into S1`

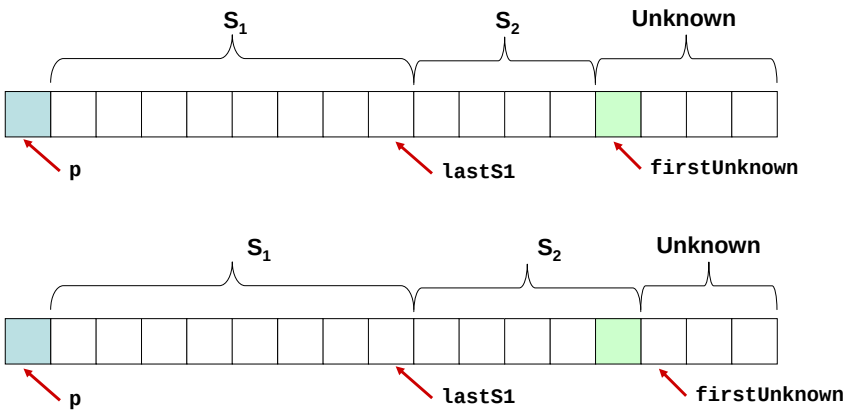
- `swap with array[lastS1 + 1] //swap with first element of S2`
- `lastS1++`
- `firstUnknown++`



Move the element into S_2

Move `array[firstUnknown]` into S_2

- `firstUnknown++`



Partitioning code

```
private int partition(int[] theArray, int first, int last) {
   choosePivot(theArray, first, last);

   int pivot = theArray[first];
   int lastS1 = first;

   for (int firstUnknown = first + 1; firstUnknown <= last; firstUnknown++) {
      if (theArray[firstUnknown] < pivot) {
         lastS1++;
         swap(theArray, firstUnknown, lastS1);
      }
   }

   swap(theArray, first, lastS1);
   return lastS1;
}
```

```
public void quicksort( int[] data, int first, int last ) {
    if (first < last) {
         int p = partition( data, first, last );
         quicksort( data, first, p - 1 );
         quicksort( data, p + 1, last );
    }
}
```


Problems?

Normally executes in O(nlog(n)) time

- Worst case is $O(n^2)$
- Why?

Most important part of quicksort is choosing good pivot

- Evenly divide the problem
- Ideal choice is element which will end up in the middle of the sorted array

Methods of choosing pivot

- Randomly choose it
- Take sample of elements and choose median value
- Choose center value.

Generalised Sorting

Implement the Comparable interface.

- Defines an ordering for that object

```
public interface Comparable {
       public int compareTo(Object o)
}
```

`String a = ...`

`String b = ...`

`a.compareTo(b)`

- return a negative value if a is ordered before b
- return zero if a and b are ordered the same
- return a positive value if a is ordered after b

Sorting objects

Object must implement the Comparable interface

- Use the compareTo method

```
public void sort(Comparable[] data){
       for(int i=1; i<data.length; i++)
              for(int j=i; j>0; j--)
                     if (data[j].compareTo(data[j-1]) < 0)
                            swap(data, j, j-1);
}
```


Questions?

Multi-dimensional Arrays

Exceptions

File IO using `java.io` character streams

Recursion

Performance Analysis

Sorting