CPS 100 - Program Design and Analysis II
Dr. Susan Rodger
Section: Sorting (handout)

Read Chap 8

Why sort?

Selection Sort

A is an array of size N.

IndexOfSmall(A,k,j) returns the index of the smallest element in array A between (and including) positions k through j.

```
SelectionSort(A,N)
 {
    for (pos=0; pos < N-1; pos++)
    {
      smallIndex = IndexOfSmall(A,pos+1,N-1);
      Swap(A[pos], A[smallIndex]);
    }
 }
```

Analysis: N elements

- Comparisons:
- Swaps:
- Running Time:

Review:

BubbleSort

- Comparisons:
- Swaps:
- Running Time:

InsertionSort - (see algorithm in text)

- Comparisons:
- Swaps:
- Running Time:

Why use?

Shellsort

```
Shellsort

  while (groupsize < N)
  {
    partition into groups
    InsertionSort each group
    increase group size
  }
  InsertionSort N elements
```

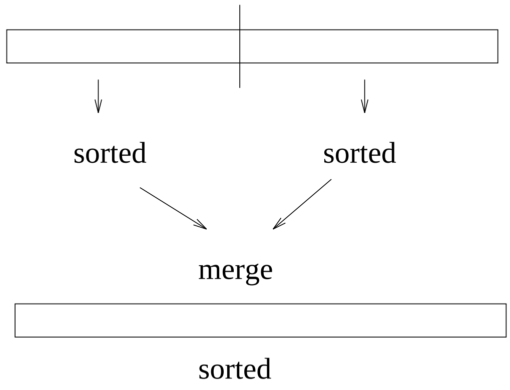
Analysis depends on group sizes chosen.

To see this and other animations, type the following on acpub (see the README file for more info):

```
    cd ~rodger/bin/anims
    setup
    shellsort < shellsort.in      // shellsort demo given in class
    bsort < bsort.in              // bubblesort, also try bsort2.in
    selectsort < selectsort.in    // just the merge part of mergesort
    quicksort < quicksort.in      // also try quicksort2.in, quicksort3.in
    merge < merge.in             // just the merge part of mergesort
    hanoi < hanoi.in             // Towers of hanoi
    queens                       // 8 queens, no data file
```

Mergesort

```
MergeSort(A, left, right)
{
  if (left < right)
  {
    mid = (left+right)/2
    MergeSort(A, left, mid)                // recursively sort leftside of A
    MergeSort(A, mid+1, right)             // recursively sort rightside of A
    Merge(A, left, mid+1,  right)          // merge two sorted sides
  }
}
```



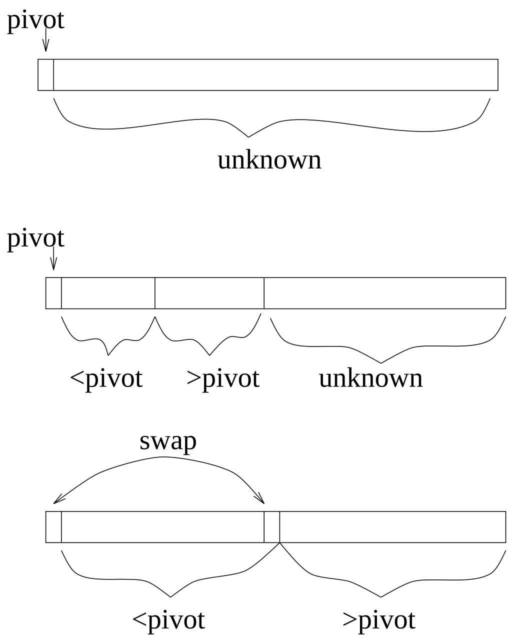
Example

```
3   6   8   4   7   1   5   2
```

Analysis:

Quicksort

Trace of Partition



The Partition function returns pos (by reference), the index of the partition element after it has been put into its place in sorted order.

```
QuickSort(A, first, last)
{
  if (first < last)
  {
    Partition(A, first, last, pos)     // partitions and returns pos
    Quicksort(A, first, pos-1)         // Quicksort left side of partition
    Quicksort(A, pos+1, last)          // Quicksort right side of partition
  }
}
```

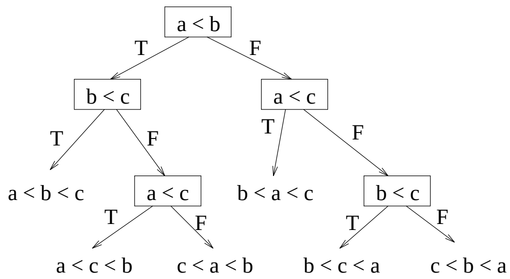
```
Partition(A, F, L, pos)
{
  pos = A[F]                                  // pivot is in first position
  lastlt = F                                  // position of last item in <pivot section
  unknown = F+1                               // position of first unknown
  for ( ;unknown <= L; unknown++)             // process each unknown element
  {
    if (A[unknown] < pivot)                   // swap unknown with leftmost item
    {                                         //  in >pivot section
       lastlt++
       Swap(A[unknown],A[lastlt])
    }
  }
  Swap(A[F], A[lastlt])                       // put pivot in place
  pos = lastlt                                // return pivot position
}
```

Analysis:

What is the Fastest Sort?

Decision Tree (Comparision Tree)

Sort a, b, and c



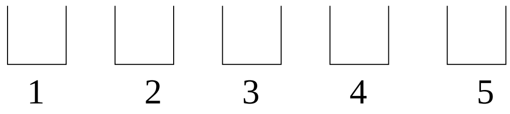
Bucket Sort

Sort N numbers in the range 1 to M

```
Create M buckets
Put numbers in buckets
Pull numbers out of buckets in order
```

Example

Sort: 4 5 2 3 2 5



Analysis:

Radix Sort

This is similar to bucket sort, but applies to larger integers.

Create 10 buckets, 0-9.

Sort numbers by last digit.

Then sort numbers by next to last digit.

etc.