selecting sorts/sorting details

- **$O(n^2)$ sorts are reasonable in certain situations**
  - advantages/disadvantages
  - selection sort: easy to code
  - insertion sort: stable, fast on nearly-sorted inputs
  - bubble sort: nearly worthless

- **$O(n \log n)$ sorts are the fastest comparison-based sorts**
  - quick sort is quick, degrades in the worse case
  - merge sort is good in the worst case, uses extra storage
  - heap sort is good in the worst case, no extra storage

- **Shell sort: in between, fast, straightforward to code**
Partition code for quicksort

- Quicksort partition in use

```cpp
void quick(Vector<string> & a, int first, int last)
{
    if (first < last)   {     int piv = partition(a,first,last);     quick(a,first,piv-1);     quick(a,piv+1,last);
    }
}
```

- Loop invariant:
  - statement true each time loop test is evaluated

- Class invariant
  - properties about state of class true after certain functions have executed
Loop invariant practice:

- Remove all zeroes from an array leaving order of other elements unchanged

- number of elements stored in extra variable

RemoveZeroes(a,num); // a is vector above, num == 6

void RemoveZeroes(Vector<int> & list, int & numElts)
// pre: numElts = # elements in list
// post: numElts = # elements in list, all zeros removed
Mergesort

- **Divide and conquer,** $O(n \log n)$ **algorithm**
  - divide array/list into two equal halves
  - sort each half (recursively)
  - merge sorted subarrays/sublists together
- **merging uses extra storage (when arrays/vectors used)**
  - how much extra storage? what about linked lists?

```cpp
void mergesort(Vector<string> & a, int first, int last)
{  if (first < last)
    { int middle = (first + last)/2; // (last-first)/2 + first
        mergesort(a,first,middle);
        mergesort(a,middle+1,last);
        merge(a,first,middle,last);
    }
}
```
non-comparison based sorts

- lower bound: $\Omega(n \log n)$ for comparison based sorts (like searching lower bound)
- bucket sort/radix sort are not-comparison based, faster asymptotically and in practice

sort a vector of ints, all ints in the range 1..100, how?

radix: examine each digit of numbers being sorted