## CSE 142
## Programming I

### Sorting



## Problem:

- Put an array of items in order
  - Either ascending or descending
- Given an array a[0]…a[n-1], reorder the elements so that a[0] <= a[1] <= a[2] etc.



## Applications

- Why would we want to do this?
- This is used all over the place to organize data
  - Can easily merge data
  - Order logs
  - Prepare output
  - etc.



## Algorithms

- What is an algorithm?
- When we design an algorithm, what things should we think about?



## Back to Our Problem

- How might we approach sorting?
- What basic operation(s) do we need?
- How do we put the basic pieces together?



## Simple Idea

- We can put things “more” in order by swapping two elements that are out of order
- Go through the array, and swap neighboring elements if they’re out of order
  - Repeat until done

## Bubble Sort

```
void bubblesort(int nums[], int first, int last){
   int i, changed;
   do {
      changed = FALSE;
      for (i=first; i<last; i++){
         if (nums[i] > nums[i+1]) {
            swap(&nums[i], &nums[i+1]);
            changed = TRUE;
         }
      }
   } while (changed);
}
```



## Example

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 0 | 4 | 2 | 100 | 6 | 5 | -8 |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |



## Example Continued (Ugh!)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |



## That Hurt

- Why did this take so long?
- How many times do we have to loop?
- How much work each time?



## How Can We Improve This?

- Big problem: each out of order bar can only move 1 step towards its destination each time
- Can we improve this?
  - (Of course, silly!)



## Idea!

- Instead of just comparing neighboring bars, compare things farther away
- We have a lot of options…

## New Algorithm

- Find the smallest element, and swap it with the first element
- Find the next smallest element, and swap it with the second element
- Proceed until done



## Selection Sort

```
void selectionsort(int nums[], int first, int last){
   int i, j, smallest, smallindex;
   for (i=first; i<last; i++){
      smallest = nums[i];
      smallindex = i;
      for (j=i+1; j<=last;j++){
         if (nums[j] < smallest){
            smallest = nums[j];
            smallindex = j; }
      }
      swap(&nums[i], &nums[smallindex]);
   }
}
```



## Example

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 0 | 4 | 2 | 100 | 6 | 5 | -8 |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |



## Example Continued (Ugh!)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |



## Is this any better?

- How many times do we have to loop over the array?
- How much work every time?



## Sorts Compared?

- In practice, selection sort is much faster than bubble sort
- In fact, bubble sort is particularly bad

## Can We Do Better?

- Maybe if we…



## Some Further Insights

- Selection sort improved on bubble sort but, with selection sort, we have to scan through the remainder of the array once for each element we pick out

- Why can't we do more work on each scan?



## Quicksort

- Idea: pick an element—any element—and call it the pivot
- Go through the array and put everything smaller than the pivot at the front, and everything larger than the pivot at the end
- Put the pivot in the middle
- Recursively Quicksort each half



## Example

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 0 | 4 | 2 | 100 | 6 | 5 | -8 |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |



## Quicksort Code

```
void quicksort(int nums[],
  int first, int last){
  int i, j;
  int pivot;

  if (first >= last) return;

  pivot = nums[first];
  i = first;
  j = last+1;
  while (i<j){
    do {
      i++;
    } while (pivot > nums[i]);
    do {
      j--;
    } while (nums[j] > pivot);

    if (i < j)
      swap(&nums[i], &nums[j]);
  }

  swap(&nums[first], &nums[j]);

  quicksort(nums, first, j-1);
  quicksort(nums, j+1, last);
  return;
}
```



## How Long Does This Take?

- This is harder to analyze than the other two.

- In the worst case, it's not too hard to see it takes $n^2$ time

- What about the average time?

## Quicksort Best Case

- Each time, if we're lucky, we split the list in two.  Each "level" takes a total of n time to do
- There are log(n) "levels", so the total time is n*log(n)
  - We say "n log n"

14. August, 2000 CSE 142 Summer 2000 — Isaac Kunen L-25

## Quicksort Average Case

- It turns out that the average case is n log(n) as well.
  - (This is not easy to see.)
- Can we do better?

14. August, 2000 CSE 142 Summer 2000 — Isaac Kunen L-26

## Optimal Sorting

- It turns out that the best one can do with a general sorting algorithm is n log(n).
  - For special cases we can do better

14. August, 2000 CSE 142 Summer 2000 — Isaac Kunen L-27

## Algorithms We've Seen:

- Searching:

| Name | Speed |
|---|---|
| Linear Search | n |
| Binary Search | log(n) |

- Sorting:

| Name | Speed |
|---|---|
| Bubble Sort | $n^2$ |
| Selection Sort | $n^2$ |
| Quicksort | n log(n) |

14. August, 2000 CSE 142 Summer 2000 — Isaac Kunen L-28

## The Message:

- Algorithm design is very important!
- We've examined algorithm speed, but there are other considerations:
  - Space
  - Correctness
  - Even ease of programming may be important

14. August, 2000 CSE 142 Summer 2000 — Isaac Kunen L-29