Computing Statistics (6)

1 COMPUTING STATISTICS

1.1 Class Rank

Every student in a graduating class at Duke (1.5k) has a grade-point average (GPA) between 0.0 and 4.0.
The registrar uses these GPAs for a number of different tasks:

- assigning Latin honors,
- identifying the valedictorian,
- computing ranking for specific students,
- characterizing the effects of grade inflation.

1.2 Some Formal Problem Statements

Let n be the number of students, and $L[i]$ be the GPA for student i.

- Given a student $0 \leq i \leq n-1$, compute the rank of i; that is, what position would $L[i]$ occupy in a sorted list?
- Given a rank k, what is the GPA that receives that rank?

1.3 Statistics

Statistics are just numbers that summarize a bunch of other numbers. For example,

- the largest GPA,
- the number of GPAs between 2 and 3,
- the sum of the digits in all n GPAs.

1.4 Mmmm, Some Useful Statistics

Given a list $L[0]$ through $L[n-1]$ of numbers, there are a number of statistics we could make use of concerning L:

- maximum: largest number in L.
- minimum: smallest number in L.
- midpoint: average of max and min.
- mean: average of numbers in L.
- mode: most common number in L.
- median: middlest number in L (half bigger, half smaller).

1.5 Facts About Statistics

One use of statistics is winning bets. I'm going to pick one of the elements of L (uniformly at random). You get to pick one number.

- Lose if your number is smaller than mine.
- Lose if your number is bigger than mine.
- Lose if you get the wrong number.
- Lose proportional to the square of the difference between the numbers.
- Lose proportional to the *absolute* difference between the numbers.

maximum, minimum, mode, mean, median.

1.6 Solutions by Sorting

Several of these statistics can be computed easily (constant time) if L is already sorted.

- minimum.
- maximum.
- midpoint.
- median.

$L[0]$, $L[n-1]$, $(L[0]+L[n-1])/2$, $L[\lceil n/2 \rceil]$.
We'll talk more about all of these in a moment, and you'll look at mode and rank in the homework.

1.7 Solutions if Unsorted

If L is not sorted, we can sort it in $O(n \log n)$... but for max, min, and midpoint we'd expect $O(n)$ to suffice.
We'll see that $O(n)$ is all we need for median as well!

2 MEAN

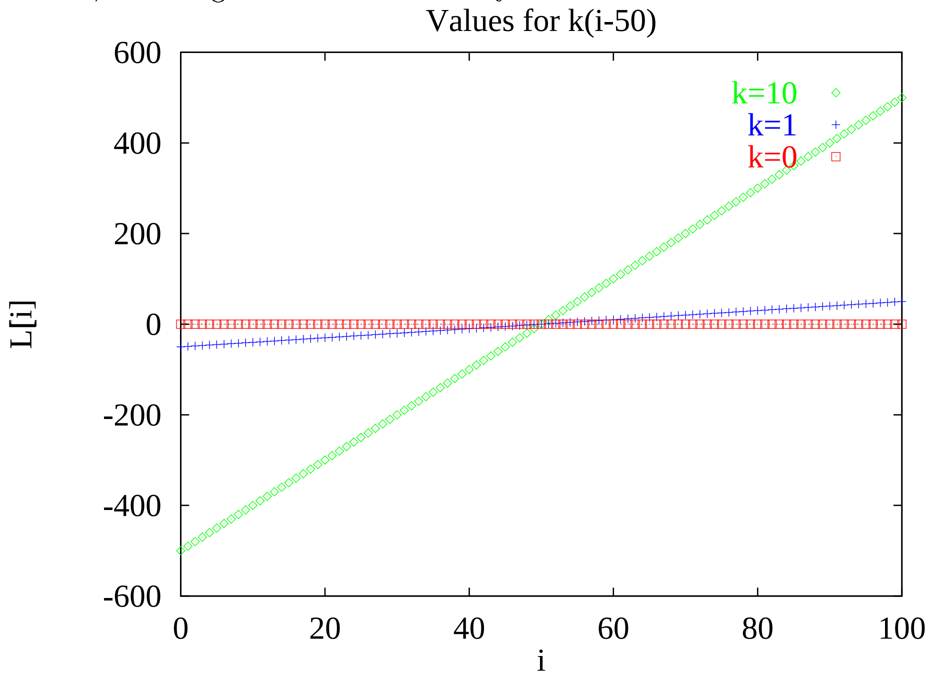
2.1 Computing the Mean

Let's start with an easy one.
The mean (average) of L is $\mu = \sum_{i=0}^{n-1} L[i]/n$.
We can compute this directly:

```
int mean( vector<int> & data )
{
  int sum = data[0];
  for( int i = 1; i < data.size(); i++ )
    sum += data[i];
  return sum / data.size();
}
```

2.2 Standard Deviation

Just from the mean, we don't get a sense of how spread out the numbers are.
For example, let $L[i] = k(i - 50)$ for $0 \leq i \leq 100$ and constant k. The mean of L is zero for any k. But, the range varies substantially.



Standard deviation is a measure of “spread” around the mean.

- sample variance, $\sigma^2 = \sum_{i=0}^{n-1}(L[i] - \mu)^2/(n-1)$,
- standard deviation $= \sqrt{\sigma^2}$.

Standard devision of above list is $58.16786|k|$ (grows with range).

2.3 Computing the Standard Deviation

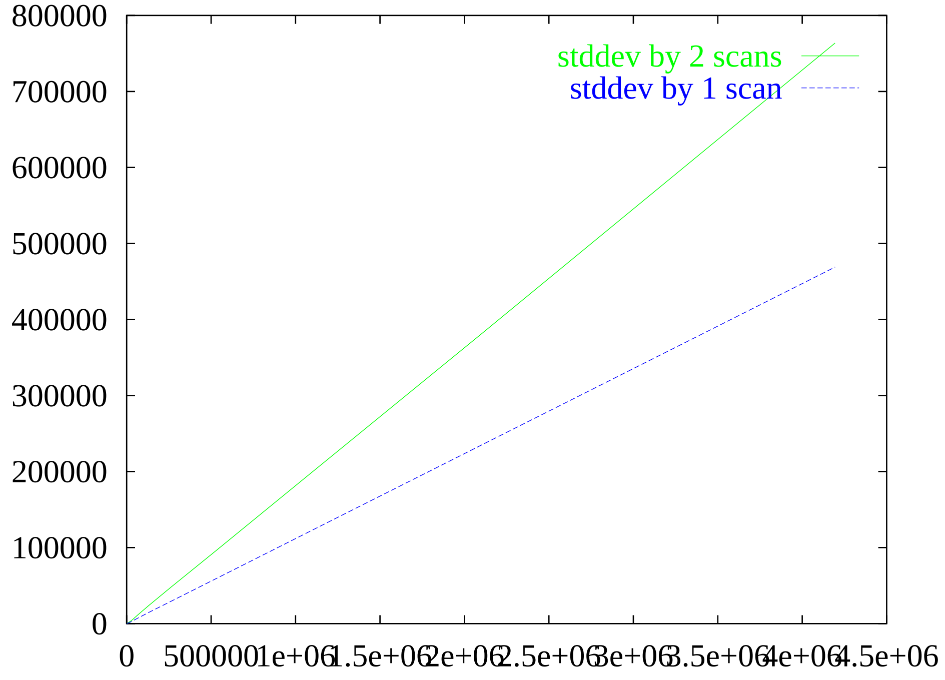
Obvious how to do it in multiple passes through the list.
But...

- $\sum_{i=0}^{n-1}(L[i] - \mu)^2$
- $= \sum_{i=0}^{n-1}(L[i]^2 - 2\mu L[i] + \mu^2)$
- $= \sum_{i=0}^{n-1} L[i]^2 - 2\mu \sum_{i=0}^{n-1} L[i] + \mu^2 \sum_{i=0}^{n-1} 1$
- $= \sum_{i=0}^{n-1} L[i]^2 - 2n\mu^2 + \mu^2 n$
- $= \sum_{i=0}^{n-1} L[i]^2 - n\mu^2$

Standard deviation is easily computed from this.
So, we can compute the sum of the list elements and the sum of the *squares* of the list elements in a single pass.

2.4 Empirical Data

I was curious as to whether this actually helps, and it seems to.



Plot of list size (n) vs. running time (in microseconds) for two algorithms for calculating the standard deviation of a list.

3 EXTREMALS

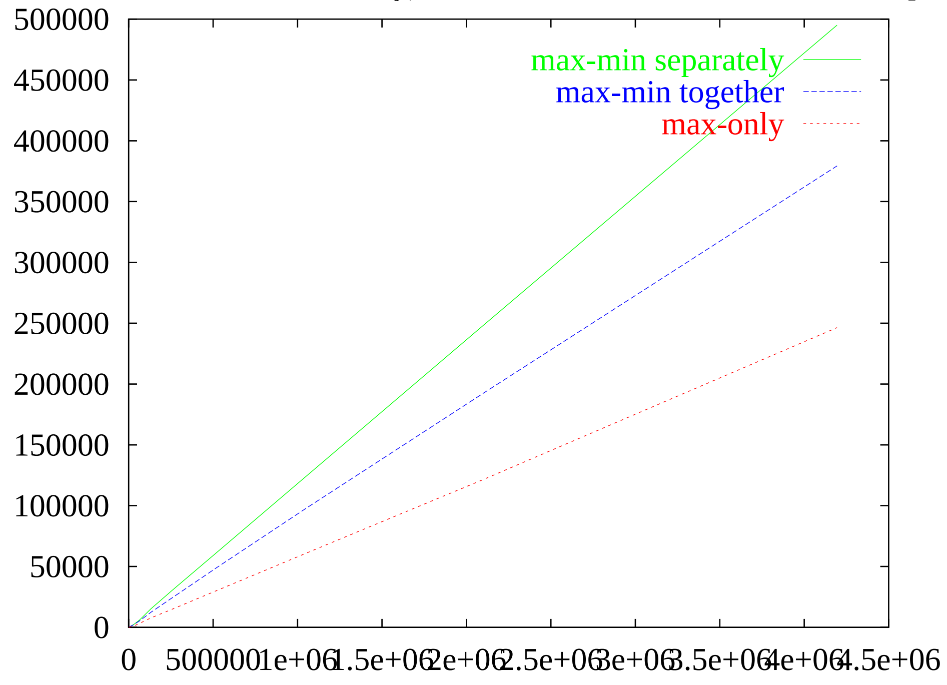
3.1 Computing the Max

Another simple calculation is that of finding the largest element in a list.
This is an obvious $\Theta(n)$ operation... in fact, the total number of comparisons is just $n-1$.

```
int max( vector<int> & data )
{
  int max = data[0];
  for( int i = 1; i < data.size(); i++ )
    if( data[i] > max )
      max = data[i];

  return max;
}
```

3.2 Computing Max and Min Together

Finding the minimum is similar to finding the max.
If we want the midpoint (or the range), we need both the min and max together.
We can do this the obvious way, or a clever one that uses fewer comparisons.



3.3 Algorithm

Here is the algorithm (assumes n even):

```
void minMax( vector<int> & data, pair<int, int> & result )
{
  int i = 0;
  int curmax = data[i];
  int curmin = data[i+1];

  if( curmax < curmin )
  {
    curmax = data[i+1];
    curmin = data[i];
  }

  int max = curmax;
  int min = curmin;

  for( i = 2; i < data.size(); i += 2 )
  {
    curmax = data[i];
    curmin = data[i+1];

    if( curmax < curmin )
    {
      curmax = data[i+1];
      curmin = data[i];
    }

    if( curmax > max )
      max = curmax;

    if( curmin < min )
      min = curmin;
  }

  result.first = min;
  result.second = max;
}
```

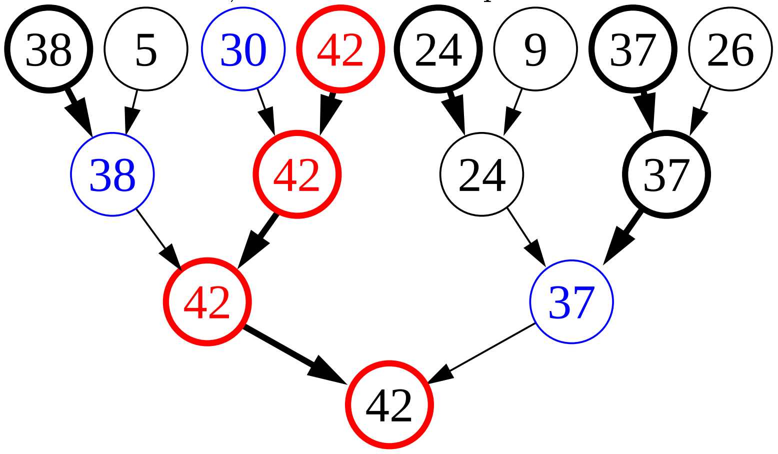
Computes min-max in pairs. Only need to compare current min to min and current max to max. How many comparisons all together?

3.4 Finding the Salutatorian

One other cute trick, before we look at a more general and useful algorithm.
How many comparisons does it take to find the *second* largest element in a list?
Easy to solve using $2n - 3$ comparisons by pulling out the largest, then taking the max of the remaining $n - 1$ elements.
Can solve this in $n + \log n - 2$ comparisons. (It only took 5 times longer in my experiments.)

3.5 Comparison Tree

In a tournament, where is the 2nd place team?



4 ORDER STATISTICS

4.1 Order Statistics

Finding the largest, second largest, and smallest elements are examples of “order statistics.”
The ith order statistic of a list is the value that ends up being in position i in a sorted order.

- Min: order statistic 1.
- Second smallest: order statistic 2.
- Median: order statistic $n/2$.
- Second largest: order statistic $n - 1$.
- Max: order statistic n.

4.2 Class Rank Problem

Let’s say we wanted to honor the students with the top 10 percent GPAs.

1. Find the GPA with rank $n/10$.

2. List all GPAs, return those that are at least as good as the threshold.

Homework: How do you compute the rank for a given student?

4.3 Randomized Select

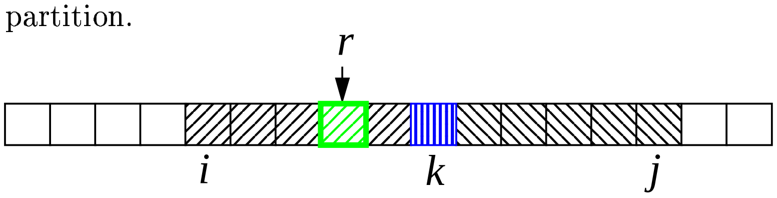
If r is not between i and j, there's no point in sorting...

```
int select( vector<int> & data, int i, int j, int r )
{
  if( i == j )
    return data[r];

  int l = rando.RandInt( i, j );
  int k = partition( data, i, j, l );

  if( k == r)
    return data[r];

 if( r < k )
   return select( data, i, k-1, r );
 else
   return select( data, k+1, j, r );
}
```

Show that the expected running time is $O(n)$ [HW].

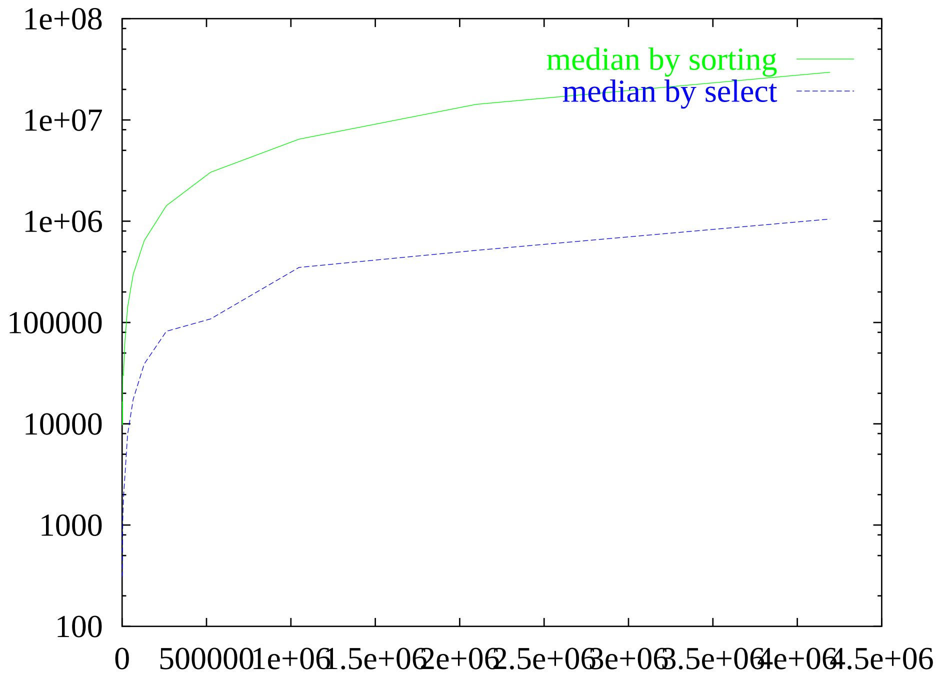
4.4 Illustration

We know that the element with rank r is somewhere between index i and index j. We partition and find the pivot element at position k. We only need to recurse on the left partition.



4.5 Experimental Results

On a log scale... for 4 million elements, 30 seconds vs. 1 second.



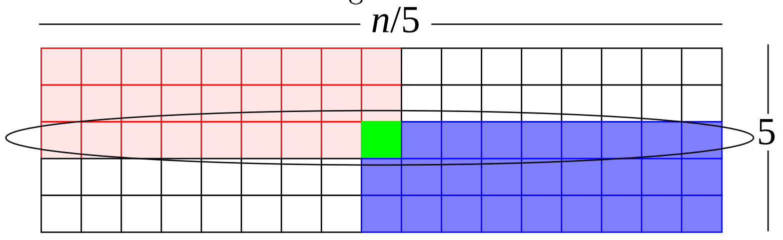
4.6 Deterministic Select

It is interesting to note that there *is* a way to get rid of the randomness here without losing the nice linear running time bound. After all, the worst-case running time here is $O(n^2)$, whereas we know we can do it in $O(n \log n)$ deterministically.
The resulting algorithm is not really practical, but it's clever.
Idea: Replace with choice of pivot element with something that guarantees we get a sufficiently good split.

- Step 1: Break n elements into blocks of 5 each.
- Step 2: Compute the median of each 5-element block in constant time.
- Step 3: Collect together the $n/5$ medians and recursively compute *their* median.
- Step 4: Chose the resulting element as a pivot.

4.7 Analysis

What is the recurrence? Clearly, we compute the median of the $n/5$ medians. Then, what's the worst case for running select on what's left?



$T(n) \leq T(n/5) + T(3n/4) + n$.
This is $O(n)$ because $T(n) \leq T(n/5) + T(3n/4) + n = T(19n/20) + n$ (recursion tree).