Admin

- Today's topics
  - Sorting, sorting, and more sorting!
- Reading
  - Ch 7
- Midterm next Tuesday evening
  - Terman Aud 7-9pm
- Boggle and late days

Selection sort code

```
void SelectionSort(Vector<int> &v)
{
   for (int i = 0; i < v.size()-1; i++) {
      int minIndex = i;  // find index of min in range i to end
      for (int j = i+1; j < v.size(); j++)  {
         if (v[j] < v[minIndex])
            minIndex = j;
       }
       Swap(v[i], v[minIndex]); // swap min to front
  }
```

Selection sort analysis

- Count work inside loops
  - First iteration does N-1 compares, second does N-2, and so on
    - one swap per iteration

$$N-1 + N-2 + N-3 + \ldots + 3 + 2 + 1$$

"Gaussian sum"

Add sum to self

$$\begin{aligned} & N-1 + N-2 + N-3 + \ldots + 3 + 2 + 1 \\ & \underline{+ 1 + 2 + 3 + \ldots + N-2 + N-1} \\ = \; & N + N + N + \ldots + N + N \\ = \; & (N-1)N \end{aligned}$$

Sum = $1/2 * (N-1)N = O(N^2)$

Insertion sort algorithm

- How you might sort hand of just-dealt cards…
  - Each subsequent element inserted into proper place
    - Start with first element (already sorted)
    - Insert next element relative to first
    - Repeat for third, fourth, etc.
    - Slide elements over to make space during insert

Insertion sort code

```
void InsertionSort(Vector<int> &v)
{
  for (int i = 1; i < v.size(); i++) {
    int cur = v[i]; // slide cur down into position to left
    for (int j=i-1; j >= 0 && v[j] > cur; j--)
        v[j+1] = v[j];
    v[j+1] = cur;
  }
}
```

Insertion sort analysis

- Count work inside loops
  - First time inner loop does 1 compare/move
  - Second iteration does <= 2 compare/move, third <= 3, and so on
  - Last iteration potentially N-1 comparisons
- Cases
  - What is best case? Worst case?
  - Average (expected) case?

Insertion vs Selection

- Big O?
- Mix of operations?
  - Number of comparisons vs moves
- Best/worst inputs?
- Ease of coding?

- Why do we need multiple algorithms?

Quadratic growth

- In clock time
  - 10,000 3 sec
  - 20,000 13 sec
  - 50,000 77 sec
  - 100,000 5 min
- Double input -> 4X time
  - Feasible for small inputs, quickly unmanagable
- Halve input -> 1/4 time
  - Hmm… can recursion save the day?
  - If have two sorted halves, how to produce sorted full result?

Mergesort idea

- "Divide and conquer" algorithm
  - Divide input in half
  - Recursively sort each half
  - Merge two halves together
- "Easy-split hard-join"
  - No complex decision about which goes where, just divide in middle
  - Merge step preserves ordering from each half

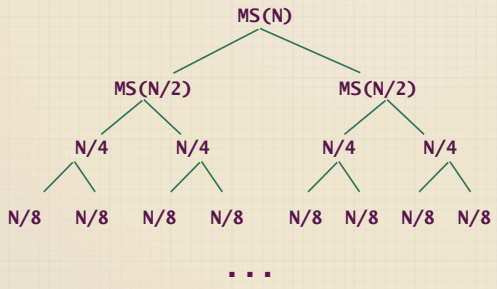
Merge sort code

```
void MergeSort(Vector<int> &v)
{
   if (v.size() > 1) {
     int n1 = v.size()/2;                          } O(1)
     int n2 = v.size() - n1;
     Vector<int> left = Copy(v, 0, n1);            } O(N)
     Vector<int> right = Copy(v, n1, n2);
     MergeSort(left);
     MergeSort(right);
     Merge(v, left, right);                        } O(N)
   }

}
```

$$T(N) = N + 2T(N/2)$$

Mergesort analysis



= N

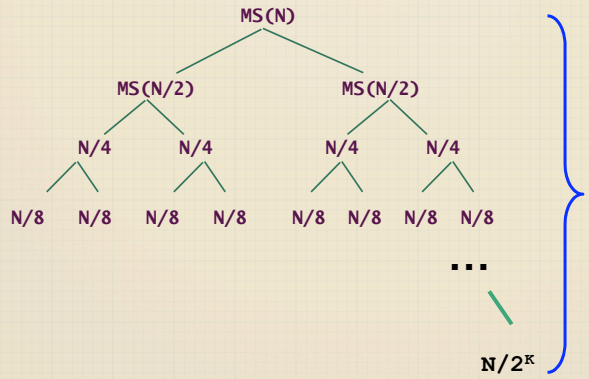
= N/2 + N/2

= 4*N/4

= 8*N/8

Each level contributes N

Mergesort analysis



K levels

$N/2^K = 1$

$N = 2^K$

$\log N = K$

log N levels * N per level= O(NlogN)

Quadratic vs linearithmic

- Compare SelectionSort to MergeSort
  - 10,000 3 sec .05 sec
  - 20,000 13 sec .15 sec
  - 50,000 78 sec .38 sec
  - 100,000 5 min .81 sec
  - 200,000 20 min 1.7 sec
  - 1,000,000 8 hrs (est) 9 sec
- O(NlogN) is pretty good, can we do better?
  - Theoretical result (beyond scope of 106B) no general sort algorithm better than NlogN
  - But a better NlogN in practice?

Quicksort idea

- "Divide and conquer" algorithm
  - Divide input into low half and high half
  - Recursively sort each half
  - Join two halves together
- "Hard-split easy-join"
  - Each element examined and placed in correct half
  - Join step is trivial