Motivation

- sorting with insertion sort is $O(n^2)$
- we can divide the array into two halves and sort them separately
- each subproblem could be sorted in $O(n^2/4)$
- sorting both halves will require a total of $O(n^2/2)$
- we need an additional operation to combine both solutions

Divide and Conquer

- **Divide** the problem into smaller subproblems
- **Conquer** recursively
  - … each subproblem
- **Combine** Solutions
Divide and Conquer

- Divide the array into two halves
  - just need to calculate the mid point
- Conquer Recursively each half
  - call Merge Sort on each half (i.e. solve 2 smaller problems)
- Merge Solutions
  - after both calls are finished, proceed to merge the solutions

Merge Sort: pseudocode

```c
if (hi <= lo) return;

int mid = lo + (hi - lo) / 2;
mergesort(A, lo, mid);
mergesort(A, mid+1, hi);
merge(A, lo, mid, hi);
```

Merging two sorted arrays

A secondary array is necessary to guarantee a linear time operation
### Merge

```c
void merge(int *A, int *aux, int lo, int mid, int hi) {
    // copy array
    std::memcpy(aux+lo, A+lo, (hi-lo+1) * sizeof(int));
    // merge
    int i = lo, j = mid + 1;
    for (int k = lo ; k <= hi; k++) {
        if (i > mid) A[k] = aux[j++];
        else if (j > hi) A[k] = aux[i++];
        else if (aux[j] < aux[i]) A[k] = aux[j++];
        else A[k] = aux[i++];
    }
}
```

### Merge Sort

```c
void r_mergesort(int *A, int *aux, int lo, int hi) {
    // base case (single element or empty list)
    if (hi <= lo) return;
    // divide
    int mid = lo + (hi - lo) / 2;
    // recursively sort halves
    r_mergesort(A, aux, lo, mid);
    r_mergesort(A, aux, mid+1, hi);
    // merge results
    merge(A, aux, lo, mid, hi);
}
```

```c
void mergesort(int *A, int n) {
    int *aux = new int[n];
    r_mergesort(A, aux, 0, n-1);
    delete [] aux;
}
```

### Merge Sort: random array

https://www.toptal.com/developers/sorting-algorithms/merge-sort

### Merge Sort: reversed array

https://www.toptal.com/developers/sorting-algorithms/merge-sort
Comments on Merge Sort

- **Major disadvantage**
  - it is not **in-place**
  - in-place algorithm exists but it is complex and inefficient

- **Improvements**
  - use insertion sort for small arrays
    - avoid overhead on small instances (~10 elements)
  - stop if already sorted
    - avoids unnecessary merge
    - works well with partially sorted arrays

Sorting Algorithms

<table>
<thead>
<tr>
<th></th>
<th>Best-Case</th>
<th>Average-Case</th>
<th>Worst-Case</th>
<th>Stable?</th>
<th>In-place?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Selection Sort</strong></td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n^2)$</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Insertion Sort</strong></td>
<td>$\Theta(n)$</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n^2)$</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Merge Sort</strong></td>
<td>$\Theta(n\log n)$</td>
<td>$\Theta(n\log n)$</td>
<td>$\Theta(n\log n)$</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Next lecture!