Sorting Algorithms and Complexity

Dr. Robert Lowe

Division of Mathematics and Computer Science
Maryville College

Maryville COLLEGE

Outline

Maryville COLLEGE

Activity: Sort your cards

1. Layout the cards in front of you.
2. Using some technique which looks at one or two cards at once, put the cards into sorted order.
3. Repeat this a few times, until you are conscious of how you do it.

Activity: How did you sort?

1. Think about how you sorted your cards.
2. Try to write down the general idea about how it was done.

Activity: Produce Pseudocode of your technique

1. Create a directory `labs/week10`
2. In this directory, using your favorite editor, create a file named `mysort.txt`
3. Write pseudocode to describe the sorting method that you used.
4. Compare sorting methods with your neighbors. Who's sort looks better?

Sorting Algorithms

- Sorting is one of the most commonly studied tasks in computer science.
- Some of the first algorithms to be studied in terms of complexity were sorting algorithms.
- Many “official” sorting algorithms exist.
- These include:
  - Selection Sort
  - Bubble Sort
  - Merge Sort

Selection Sort

```
selction_sort(ar)
    for i = 0 to ar
        min=i
        for j = i+1 to ar.size()-1
            if ar[j] < ar[min]
                min = j
            end if
        end for
        swap ar[i] and ar[min]
    end for
```

- Carry out selection sort with your cards.
- Intuitively, what is selection sort doing?

Bubble Sort

```
bubble_sort(ar)
    do
        swapped = false
        for i=0 to ar.size()-2
            if ar[i+1] < ar[i]
                swap ar[i+1] and ar[i]
                swapped = true
            end if
        end for
    while swapped
```

- Carry out selection sort with your cards.
- Intuitively, what is bubble sort doing?

Merge Sort

```
merge_sort(ar)
    if ar.size() <= 1
        retun
    end if

    mid = ar.size() / 2
    merge_sort(ar[0..mid])
    merge_sort(ar[mid+1 .. ar.size()-1)
    merge(ar[0..mid], ar[mid+1 .. ar.size()-1])

merge(left, right)
    While left and right are not empty
        take the smallest of the first element in
        left and right
    end while
```

Activity: Compare Sorting Algorithms

1. Which sorting method seemed closer to what you did?
2. Which sorting method seemed more efficient?

Maryville COLLEGE

Which algorithm is better?

- We can always buy more memory.
- We can never buy more time.
- We typically evaluate an algorithm based on how long it will take to execute.
- The standard is to rate the algorithm by the number of steps necessary to solve a problem of n size.

Asymptotic Notation

- Suppose we want to determine the number of steps as a function $f(n)$
- Usually, we can't find the exact $f(n)$, and even if we could,this would not be all that meaningful.
- We instead compute an asymptotic bound.
- An algorithm is $O(g(n))$ if $f(n) \leq cg(n)$ for some constant c for all n above some threshold.
- Intuitively, $O(g(n))$ gives us the "worst case" run time.

Maryville COLLEGE

How different can they be?

- Some common runtime include:
  - Logarithmic - $O(\lg(n))$
  - Linear - $O(n)$
  - n log n - $O(n\lg(n))$
  - Quadratic - $O(n^2)$
  - Exponential - $O(2^n)$
- Using a spreadsheet, let's see how these functions relate to each other.

Selection Sort Complexity

Find the time complexity of the following:

```
selction_sort(ar)
    for i = 0 to ar
        min=i
        for j = i+1 to ar.size()-1
            if ar[j] < ar[min]
                min = j
            end if
        end for
        swap ar[i] and ar[min]
    end for
```

Bubble Sort Complexity

Find the time complexity of the following:

```
bubble_sort(ar)
    do
        swapped = false
        for i=0 to ar.size()-2
            if ar[i+1] < ar[i]
                swap ar[i+1] and ar[i]
                swapped = true
            end if
        end for
    while swapped
```

Merge Sort Complexity

Find the time complexity of the following:

```
merge_sort(ar)
    if ar.size() <= 1
        retun
    end if

    mid = ar.size() / 2
    merge_sort(ar[0..mid])
    merge_sort(ar[mid+1 .. ar.size()-1)
    merge(ar[0..mid], ar[mid+1 .. ar.size()-1])

merge(left, right)
    While left and right are not empty
        take the smallest of the first element in
        left and right
```

Activity: What is the complexity of your search algorithm?

1. Compute the time complexity of your own sorting algorithm.
2. Include your runtime and justification of your complexity in your text file.

Activity: Code a Sorting Algorithm

1. Code your favorite sorting algorithm.
2. Your program should do the following:
    1. Ask for 10 numbers, which it adds to a vector.
    2. Run your sorting algorithm.
    3. Print the sorted vector.