CSCA48 Winter 2015

Week 10 - Sorting

Anna Bretscher

March 18/19, 2015

Why Sorting?

- A big part of computer science - happens all the time

Why Sorting?

- A big part of computer science - happens all the time
- Sorting algorithms have been well studied and therefore developed

Why Sorting?

- A big part of computer science - happens all the time
- Sorting algorithms have been well studied and therefore developed
- Sorting algorithms are interesting to analyze

Why Sorting?

- A big part of computer science - happens all the time
- Sorting algorithms have been well studied and therefore developed
- Sorting algorithms are interesting to analyze
- Good examples of recursion

Why Sorting?

- A big part of computer science - happens all the time
- Sorting algorithms have been well studied and therefore developed
- Sorting algorithms are interesting to analyze
- Good examples of recursion
- Non-recursive sorting algorithms have invariants:

Why Sorting?

- A big part of computer science - happens all the time
- Sorting algorithms have been well studied and therefore developed
- Sorting algorithms are interesting to analyze
- Good examples of recursion
- Non-recursive sorting algorithms have invariants:
    - Invariant: A statement that is true at the end of each iteration of a loop.

Why Sorting?

- A big part of computer science - happens all the time
- Sorting algorithms have been well studied and therefore developed
- Sorting algorithms are interesting to analyze
- Good examples of recursion
- Non-recursive sorting algorithms have invariants:
  - Invariant: A statement that is true at the end of each iteration of a loop.
  - Use invariants to write code/prove code works

Insertion Sort

- Very simple

Insertion Sort

- Very simple
- Takes O(??) time
- Done in-place

Insertion Sort

- Very simple
- Takes O(??) time
- Done in-place
- Idea:

```
for x in L:
    insert x into the correct position in sorted list S
```

Insertion Sort

- Very simple
- Takes O(??) time
- Done in-place
- Idea:

```
for x in L:
    insert x into the correct position in sorted list S
```

- Invariant: `S is sorted`

Insertion Sort

- Very simple
- Takes O(??) time
- Done in-place
- Idea:

```
for x in L:
    insert x into the correct position in sorted list S
```

- Invariant: `S is sorted`
- When is `Insertion Sort` fairly efficient?

Heap Sort

- Given a list, return it sorted as S
- What was the complexity again?

Heap Sort

- Given a list, return it sorted as S
- What was the complexity again?
- Build the heap `H` - O(n)

HEAP SORT

- Given a list, return it sorted as S
- What was the complexity again?
- Build the heap `H` - O(n)

```
while H not empty:
```

Heap Sort

- Given a list, return it sorted as S
- What was the complexity again?
- Build the heap `H` - O(n)

```
while H not empty:
    m = extract_min()
```

Heap Sort

- Given a list, return it sorted as S
- What was the complexity again?
- Build the heap `H` - O(n)

```
while H not empty:
    m = extract_min()
    append m to S
```

Heap Sort

- Given a list, return it sorted as S
- What was the complexity again?
- Build the heap `H` - O(n)

```
while H not empty:
    m = extract_min()
    append m to S
```

- Each `extract_min` - O(log n)

Heap Sort

- Given a list, return it sorted as S
- What was the complexity again?
- Build the heap `H` - O(n)

```
while H not empty:
    m = extract_min()
    append m to S
```

- Each `extract_min` - O(log n)
- We do this n times is for O(n log n)

Merge Sort

- Divide and Conquer algorithm

Merge Sort

- Divide and Conquer algorithm
- Split the list into L1 and L2

MERGE SORT

- Divide and Conquer algorithm
- Split the list into L1 and L2
- Recursively sort L1 and L2

Merge Sort

- Divide and Conquer algorithm
- Split the list into L1 and L2
- Recursively sort L1 and L2
- Merge L1 and L2 into a single sorted list

Merge Sort

- Divide and Conquer algorithm
- Split the list into L1 and L2
- Recursively sort L1 and L2
- Merge L1 and L2 into a single sorted list
- How do we merge two sorted lists?

Merge Sort

- Divide and Conquer algorithm
- Split the list into L1 and L2
- Recursively sort L1 and L2
- Merge L1 and L2 into a single sorted list
- How do we merge two sorted lists?

```
while (neither L1 nor L2 are empty):
```

Merge Sort

- Divide and Conquer algorithm
- Split the list into L1 and L2
- Recursively sort L1 and L2
- Merge L1 and L2 into a single sorted list
- How do we merge two sorted lists?

```
while (neither L1 nor L2 are empty):
     move min(L1[0], L2[0]) to S
```

MERGE SORT

- Divide and Conquer algorithm
- Split the list into L1 and L2
- Recursively sort L1 and L2
- Merge L1 and L2 into a single sorted list
- How do we merge two sorted lists?

```
while (neither L1 nor L2 are empty):
     move min(L1[0], L2[0]) to S
Append rest of non-empty list to S
```

MERGE SORT

```
mergesort(L):
    if len(L) < 2, return L
    split L into L1 and L2
    S1 = mergesort(L1)
    S2 = mergesort(L2)
    S = merge(S1, S2)
    return S
```

Complexity of Merge Sort

- How many times do we ”divide”? how many levels of recursion are there?
- Start with length n, then $\frac{n}{2}$, then $\frac{n}{4}, \ldots, 1$

Complexity of Merge Sort

- How many times do we "divide"? how many levels of recursion are there?
- Start with length n, then $\frac{n}{2}$, then $\frac{n}{4}, \ldots, 1$
- So $\lceil log(n) \rceil$ levels

COMPLEXITY OF MERGE SORT

- How many times do we "divide"? how many levels of recursion are there?
- Start with length n, then $\frac{n}{2}$, then $\frac{n}{4}, \ldots, 1$
- So $\lceil log(n) \rceil$ levels
- How much work do we do per level?

Complexity of Merge Sort

- How many times do we ”divide”? how many levels of recursion are there?
- Start with length n, then $\frac{n}{2}$, then $\frac{n}{4}, \ldots, 1$
- So $\lceil log(n) \rceil$ levels
- How much work do we do per level?
- Visit each item in the list, so n steps

Complexity of Merge Sort

- How many times do we "divide"? how many levels of recursion are there?
- Start with length n, then $\frac{n}{2}$, then $\frac{n}{4}, \ldots, 1$
- So $\lceil log(n) \rceil$ levels
- How much work do we do per level?
- Visit each item in the list, so n steps
- Total: O(n log n)

QUICKSORT

- Divide and Conquer like Mergesort

QUICKSORT

- Divide and Conquer like Mergesort
- Idea:

Quicksort

- Divide and Conquer like Mergesort
- Idea:

```
Select an item from L to be the pivot
```

Quicksort

- Divide and Conquer like Mergesort
- Idea:

```
Select an item from L to be the pivot
Split L into three sublists:  L1, L2, L3
```

QUICKSORT

- Divide and Conquer like Mergesort
- Idea:

```
Select an item from L to be the pivot
Split L into three sublists:  L1, L2, L3
L1 ← values in L smaller than the pivot
```

QUICKSORT

- Divide and Conquer like Mergesort
- Idea:

```
Select an item from L to be the pivot
Split L into three sublists:  L1, L2, L3
L1 ← values in L smaller than the pivot
L2 ← values in L equal to the pivot
```

QUICKSORT

- Divide and Conquer like Mergesort
- Idea:

```
Select an item from L to be the pivot
Split L into three sublists:  L1, L2, L3
L1 ← values in L smaller than the pivot
L2 ← values in L equal to the pivot
L3 ← values in L larger than the pivot
```

Quicksort

- Divide and Conquer like Mergesort
- Idea:

```
Select an item from L to be the pivot
Split L into three sublists:  L1, L2, L3
L1 ← values in L smaller than the pivot
L2 ← values in L equal to the pivot
L3 ← values in L larger than the pivot
S1 = quicksort(L1)
S3 = quicksort(L3)
```

QUICKSORT

- Divide and Conquer like Mergesort
- Idea:

```
Select an item from L to be the pivot
Split L into three sublists:  L1, L2, L3
L1 ← values in L smaller than the pivot
L2 ← values in L equal to the pivot
L3 ← values in L larger than the pivot
S1 = quicksort(L1)
S3 = quicksort(L3)
Return S1 + L2 + S3
```