Meditations, sort of, on Sorting

C. Seshadhri

University of California, Santa Cruz
sesh@ucsc.edu

April 1, 2016
Selection Sort

(Let length of array $A$ be $n$.)

Selection-Sort($A$)

1. $\text{for } i = 1 $ to $n$
2. \hspace{0.5cm} $min = A[i]$ \hspace{0.5cm} // Initialize min
3. \hspace{0.5cm} $minInd = i$ \hspace{0.5cm} // Initialize min index
4. $\text{for } j = i + 1 $ to $n$ \hspace{0.5cm} // Loop runs for $n - i$ steps
5. \hspace{1cm} if $A[j] < min$ \hspace{0.5cm} // Aha! Min needs to be changed
6. \hspace{1.5cm} \hspace{0.5cm} $min = A[j]$ \hspace{0.5cm} // So change min and min index
7. \hspace{1.5cm} \hspace{0.5cm} $minInd = j$
8. \hspace{0.5cm} // Swap $A[i]$ with $A[minInd]$
9. \hspace{0.5cm} $A[minInd] = A[i]$
10. \hspace{0.5cm} $A[i] = min$
Number of comparisons is

\[ \sum_{i=1}^{n} (n - i) = \sum_{j=1}^{n-1} j = (n - 1)n/2 = \Theta(n^2) \]
Number of comparisons

Number of comparisons is

\[ \sum_{i=1}^{n} (n - i) = \sum_{j=1}^{n-1} j = (n - 1)n/2 = \Theta(n^2) \]

Mea culpa. Not \((n - 1)(n - 2)/2\) as said in the first lecture.
“Time complexity of Selection-Sort is $O(n^2)$.”

There exists some $n_0$ and $c$ such that:
for every $n > n_0$, and every input $A$ of length $n$, the running time of Selection-Sort on $A$ is $\leq cn^2$. 

“Time complexity of Selection-Sort is $\Omega(n^2)$.”

There exists some $n'_0$ and $c'$ such that:
for every $n > n'_0$, there exists an input $A$ of length $n$, the running time of Selection-Sort on $A$ is $\geq c'n^2$. 

C. Seshadhri (UCSC)
“Time complexity of Selection-Sort is $O(n^2)$.”
There exists some $n_0$ and $c$ such that:
for every $n > n_0$, and every input $A$ of length $n$, the running time of Selection-Sort on $A$ is $\leq cn^2$.

“Time complexity of Selection-Sort is $\Omega(n^2)$.”
There exists some $n'_0$ and $c'$ such that:
for every $n > n'_0$, there exists an input $A$ of length $n$, the running time of Selection-Sort on $A$ is $\geq c'n^2$. 
Which is true?

- (R) For every input of size $n$, Selection-Sort has running time $\Omega(n^2)$.
- (B) For some (but not every) input of size $n$, Selection-Sort has running time $\Omega(n^2)$. 
Which is true?

- (R) For every input of size $n$, Selection-Sort has running time $\Omega(n^2)$.
- (B) For some (but not every) input of size $n$, Selection-Sort has running time $\Omega(n^2)$. 
Length of $A$ is $n$.

**Simple-Sort**

1. **for** $i = 2$ **to** $n$
2. $j = i$
5. $j = j - 1$
Simple-Sort in action

Take out paper. Write down the inputs $8 \ 4 \ 3 \ 7 \ 1 \ 2 \ 6 \ 5$ and $3 \ 4 \ 6 \ 2 \ 8 \ 7 \ 5 \ 1$. 
(R) I ran it and it worked!
(B) I ran it and it didn't work.
(G) I couldn't run it. What do you think I am, a computer?
Now, let’s think about it

Suppose the input was already sorted (say it was 1 2 \ldots n). What is the running time of Simple-Sort?

- (R) $O(n)$
- (B) Not $O(n)$, but definitely $O(n^2)$
- (G) Not $O(n^2)$
Now, let’s think about it

Suppose the input was already sorted (say it was $1 \ 2 \ \ldots \ n$). What is the running time of Simple-Sort?

- (R) $O(n)$
- (B) Not $O(n)$, but definitely $O(n^2)$
- (G) Not $O(n^2)$

Interesting! Simple-Sort “automatically” checks if input is sorted. That’s better than Selection-Sort
What is the time-complexity of Simple-Sort?

- (R) $O(n)$
- (B) Not $O(n)$, but definitely $O(n^2)$
- (G) Not $O(n^2)$
It never hurts to think more

What is the time-complexity of Simple-Sort?

- (R) $O(n)$
- (B) Not $O(n)$, but definitely $O(n^2)$
- (G) Not $O(n^2)$
Time-complexity of Simple-Sort

**Simple-Sort**

1. **for** $i = 2$ **to** $n$  // Runs at most $n - 1$ times
2. \hspace{1cm} $j = i$
4. \hspace{1cm} Swap $A[j - 1]$ and $A[j]$
5. \hspace{1cm} $j = j - 1$

Number of times innermost statements are run is

\[
\sum_{i=2}^{n} (i - 1) = n - 1 \sum_{i=1}^{n} i = \frac{(n - 1)n}{2} = O(n^2)
\]

Total running time is $O(n^2)$. 

C. Seshadhri (UCSC) CMPS101
Time-complexity of Simple-Sort

**Simple-Sort**(\(A\))

1. **for** \(i = 2\) **to** \(n\) // Runs at most \(n - 1\) times
2. \(j = i\)
3. **while** \(j > 1\) and \(A[j - 1] > A[j]\) // Runs at most \(i - 1\) times
4. Swap \(A[j - 1]\) and \(A[j]\)
5. \(j = j - 1\)

Number of times innermost statements are run is

\[
\leq \sum_{i=2}^{n} (i - 1) = \sum_{i=1}^{n-1} i = (n - 1)n/2 = O(n^2)
\]

Total running time is \(O(n^2)\).
Time-complexity of Simple-Sort

**Simple-Sort**($A$)

1.   \textbf{for} $i = 2$ \textbf{to} $n$ // Runs at most $n$ times  
2. \hspace{1cm} $j = i$  
3. \hspace{1cm} \textbf{while} $j > 1$ and $A[j-1] > A[j]$ // Runs at most $n$ times  
4. \hspace{1cm} Swap $A[j-1]$ and $A[j]$  
5. \hspace{1cm} $j = j - 1$

Number of times innermost statements are run is

\[ \leq n \times n = n^2 \]

Total running time is $O(n^2)$. 
Now, let’s think about the reverse

Suppose the input was in reverse sorted order (say it was $n \ n - 1 \ \ldots \ 1$). What is the running time of Simple-Sort?

- (R) $O(n)$
- (B) Not $O(n)$, but definitely $O(n^2)$
- (G) Not $O(n^2)$
Now, let’s think about the reverse

Suppose the input was in reverse sorted order (say it was $n \ n - 1 \ \ldots \ 1$). What is the running time of Simple-Sort?

- (R) $O(n)$
- (B) Not $O(n)$, but definitely $O(n^2)$
- (G) Not $O(n^2)$
Time complexity of Simple-Sort is $\Theta(n^2)$

- “Time complexity of Simple-Sort is $O(n^2)$.”
  Running time on any input is $O(n^2)$.
- “Time complexity of Simple-Sort is $\Omega(n^2)$.”
  For all $n$, there is an input where running time is $\Omega(n^2)$. 
Does Simple-Sort really sort?

Guiseppe Peano: Prove it by induction!
Does Simple-Sort really sort?

Guiseppe Peano: Prove it by induction!
Loop Invariants

**Simple-Sort**($A$)

1. **for** $i = 1$ to $n$
2. $j = i$
5. $j = j - 1$

A loop invariant is a statement (parametrized by $i$) such that:
the statement is true when the $i$th loop finishes.
Using Loop Invariants

- If the loop invariant holds when the algorithm finishes (statement for $n$), then it solves your problem
Using Loop Invariants

- If the loop invariant holds when the algorithm finishes (statement for \( n \)), then it solves your problem
- Prove the loop invariant by induction
  1. The loop invariant holds at the beginning/first loop (Base Case/Initialization)
  2. The algorithm maintains the loop invariant in the intermediate stages (Induction Step/Maintanence)
Proving Correctness

- Loop Invariant: At the end of each iteration $i$ of the $for$ loop, the subarray $A[1..i]$ consists of the elements originally in $A[1..i]$, but in sorted order.
Loop Invariants

**Simple-Sort(A)**

1. for $i = 1$ to $n$
2. \hspace{1em} $j = i$
4. \hspace{1em} Swap $A[j - 1]$ and $A[j]$
5. \hspace{1em} $j = j - 1$
6.
7. // $A[1 \ldots i]$ is in sorted order
Proving Correctness

Loop Invariant: At the end of each iteration $i$ of the for loop, the subarray $A[1..i]$ consists of the elements originally in $A[1..i]$, but in sorted order.

If we can prove the Loop Invariant, we are done. At the end of iteration $n$, $A$ will be fully sorted.
Proof

Base case/Initialization

Proof

**Base case/Initialization**


**Induction/Maintenance**

Set $i > 2$. Assume Loop Invariant is true for $i - 1$. Thus, at the end of the $(i - 1)$ loop (which is the beginning of the $i$th loop), $A[1..(i - 1)]$ is the sorted version of the original first $(i - 1)$ elements.

- Observe that the index $j$ tracks the position of the original $A[i]$.
- The while loop effectively inserts $A[i]$ in this sorted array $A[1..(i - 1)]$.
- So $A[1..i]$ becomes sorted.
Proof

**Base case/Initialization**
- \( i = 1 \): Algorithm does nothing in the first loop. \( A[1] \) is trivially in sorted order.

**Induction/Maintenance**
Set \( i > 2 \). Assume Loop Invariant is true for \( i - 1 \). Thus, at the end of the \((i - 1)\)th loop (which is the beginning of the \(i\)th loop), \( A[1..(i - 1)] \) is the sorted version of the original first \((i - 1)\) elements.

- Observe that the index \( j \) tracks the position of the original \( A[i] \).
- The while loop effectively inserts \( A[i] \) in this sorted array \( A[1..(i - 1)] \).
- So \( A[1..i] \) becomes sorted.

*Quod Erat Demonstrandum*
Insertion Sort

**Insertion-Sort** \((A)\)

1. \textbf{for} \(i = 1\) \textbf{to} \(n\)
2. \(j = i\)
3. \textbf{while} \(j > 1\) and \(A[j - 1] > A[j]\)
4. \hspace{1em} Swap \(A[j - 1]\) and \(A[j]\)
5. \hspace{1em} \(j = j - 1\)
There are many $\Theta(n^2)$ sorting algorithms. But insertion sort is one of the best among these.

- Easy to code
- $O(n)$ time on sorted (or even near-sorted) inputs
- Online! You can add more elements as you go along.
A question!

How many possible comparisons can you make among the $n$ elements of array $A$?

- (R) $n$
- (B) $n/2$
- (G) $n(n-1)/2$
- (O) $n^2$
A question!

How many possible comparisons can you make among the $n$ elements of array $A$?

- (R) $n$
- (B) $n/2$
- (G) $n(n - 1)/2 = \binom{n}{2} = \Theta(n^2)$
- (O) $n^2$

This means that any $\Theta(n^2)$ sorting algorithm is making a constant fraction of all comparisons!

Can we do any better?