upGrad

Full Stack Software Development

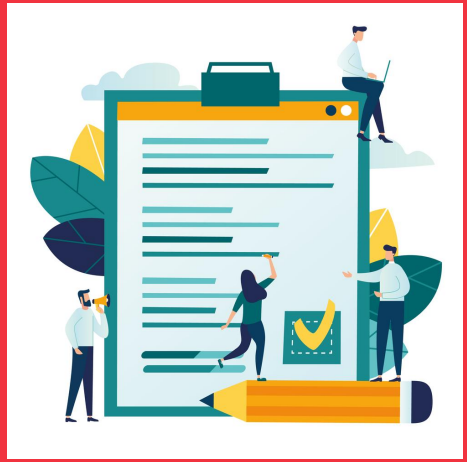
upGrad

Course: Data Structures and Algorithm

Lecture On: Basic Sorting Algorithms

Today's Agenda

1 Basic Sorting Algorithms



Sorting

upGrad

Sorting

What would we do if the dictionary was just a collection of all the English words that exist today, scattered in some random order? It would be almost impossible to search for anything, right?

Sorting, especially efficient sorting, is a prerequisite to a large number of algorithms and doing so efficiently can help us come up with several elegant solutions.

Can anyone think of any other real-world problem where we might need to sort data in order to speed up other work?

Poll 1 (15 Sec.)

Arranging elements in a proper order is known as:

1. Searching
2. Saving
3. Sorting
4. None of these

Poll 1 (Answer)

Arranging elements in a proper order is known as:

1. Searching
2. Saving
3. **Sorting**
4. None of these

Sorting

As always, we are constantly going to worry about the time and space complexity of the sorting algorithm that we choose to use. As a rule of thumb,

- O(n log n) is considered to be good sorting behaviour.
- $O(n^2)$ is considered to be bad sorting behaviour.
- O(n) is considered to be the ideal behaviour.

Along with these, we will also be worrying about another aspect when it comes to sorting, which is known as the **stability** of a sorting algorithm.

Sorting

Sorting algorithms can either be **stable** or **unstable**.

Stable sorting algorithms always maintain the relative order of the data elements in the original array when the 'key' that it is being sorted upon is the same.

Let's say we are sorting the students of a class based on their score in a test. If a student named Rahul appears ahead of Rakesh in the unsorted data set and they have both have the same score in the test, Rahul's name would appear ahead of Rakesh's name in the sorted data set if we use a stable sorting algorithm.

Sorting

When data elements are indistinguishable from one another and the ‘key’ to be sorted upon is the same, stability of the sorting algorithm does not hold much value. However, in a few other use cases, sorting stability might be highly important.

One simple example of the necessity of stable sorting would be when a data set needs to be sorted multiple times on different keys.

Let’s say we want to sort the census data of a country in ascending order first by age and then by weight (to find the youngest and lightest baby, weirdly enough), we need to use a stable sorting algorithm. Can you figure out why?

In-place Sorting

In-place sorting algorithms occupy the same memory space after sorting as in the original unsorted array and which requires a small constant amount of extra space for manipulating the input array.

Poll 2 (15 Sec.)

Can an algorithm be both stable as well inplace?

1. Yes
2. No

Poll 2 (Answer)

Can an algorithm be both stable as well inplace?

1. **Yes**
2. No

Poll 3 (15 Sec.)

Can an algorithm be both unstable as well inplace ?

1. Yes
2. No

Poll 3 (Answer)

Can an algorithm be both unstable as well inplace?

1. **Yes**
2. No

Sorting

What would you do if you forgot the lock combination of your suitcase, and breaking or resetting the lock was not an option? Since there are just three digits in the combination, we can actually go through all the possible combinations one by one.

Algorithms that do something similar are aptly named '**brute force**' algorithms, where we go through all the possible solutions in order to fix our problem. First, let's try to sort our array by sheer brute force.

We usually use two brute force sorting algorithms:

- **Bubble sort**, and
- **Selection sort.**

Bubble Sort

upGrad

6 5 3 1 8 7 2 4

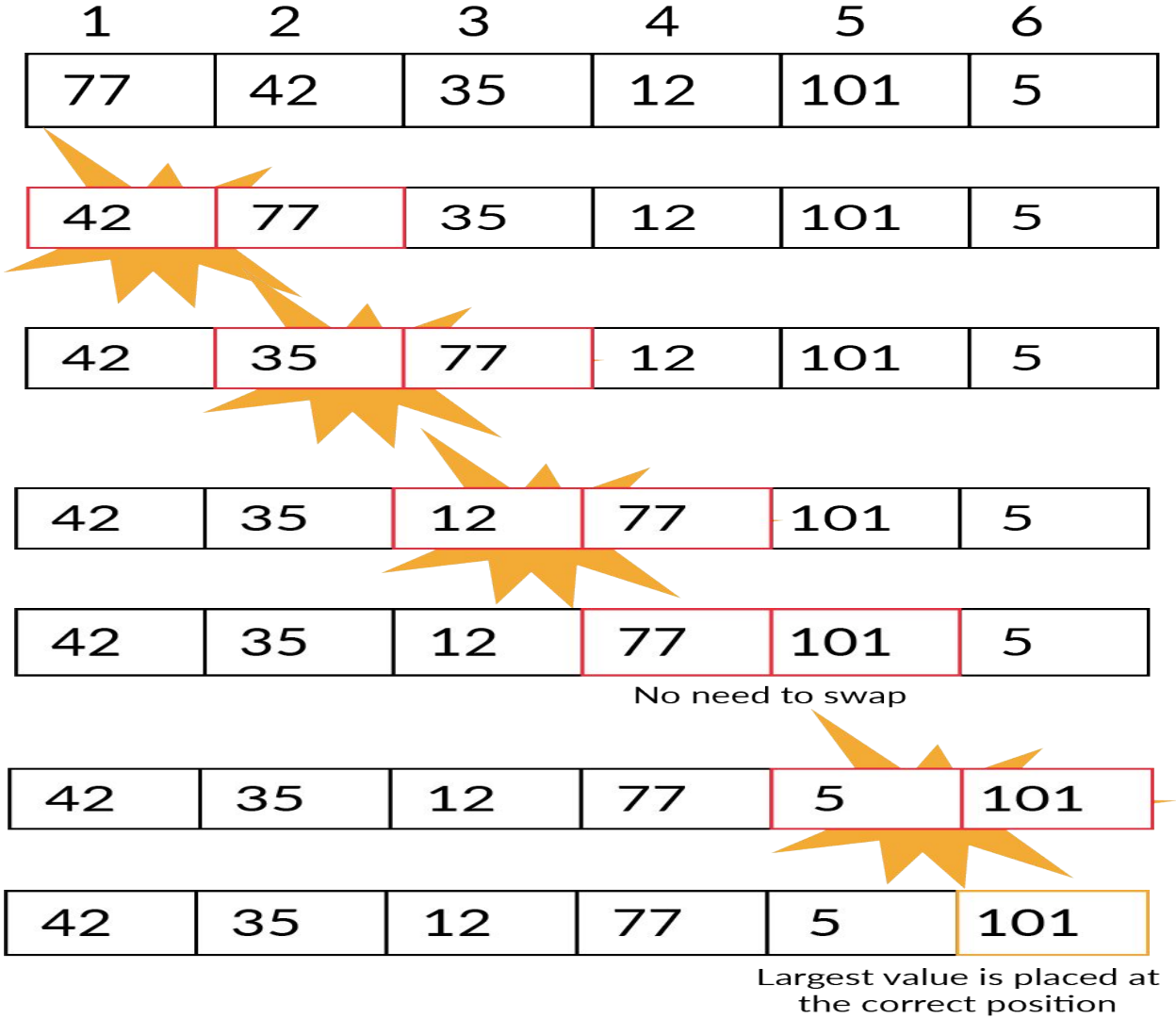
Bubble Sort

upGrad

Bubble Sort is one of the simplest sorting algorithms. It sorts by a simple exchange in the data elements and has been called ‘the generic **bad** algorithm’ by Donald E. Knuth.

The implementation is simple. We iterate over all the elements, comparing and swapping consecutive elements to ensure that **the data element that are supposed to come at the end are placed at the end of one traversal**.

One traversal thus places each element in its sorted position. For an array of size ‘n’, we must traverse this array ‘n - 1’ times in order to place all the elements in their correct position.


1
2
3
4
5
6
77
42
35
12
101
5
42
77
35
12
101
5
42
35
77
12
101
5
42
35
12
77
101
5
42
35
12
77
101
5
No need to swap
42
35
12
77
5
101
42
35
12
77
5
101
Largest value is placed at the correct position

Hands-on Coding

- **Implement bubble sort in Java.**

- Given an array of natural numbers of size x and an integer variable “i”. Write a bubble sort algorithm for the given array in such a way that the state of the array after “i” iterations gets printed.

- Consider you have an array of natural numbers of size x. You first need to sort the given array in ascending order using bubble sort and later print out the indices of the sorted elements as per their original indices (Note: The index starts from 0)

Poll 4 (15 Sec.)

What is the worst case time complexity of bubble sort?

1. $O(n^2)$
2. $O(n^3)$
3. $O(n)$
4. $O(\log n)$

Poll 4 (Answer)

What is the worst case time complexity of bubble sort?

1. **$O(n^2)$**
2. $O(n^3)$
3. $O(n)$
4. $O(\log n)$

Poll 5 (15 Sec.)

Is bubble sort stable?

1. Yes
2. No

Poll 5 (Answer)

Is bubble sort stable?

1. **Yes**
2. No

Selection Sort

Selection sort is a slightly better sorting algorithm than bubble sort, which follows a very simple algorithm.

It starts off by looking for the ‘smallest element’ in the array and swaps it with the first element. After this, it repeats the same process, starting from the second element of the array, advancing one element in each iteration.

Essentially, the array here can be thought of as being in two parts:

- The **left portion that is sorted**, which keeps on growing in size, and
- The **right portion that is unsorted**, which keeps shrinking with each iteration.

Selection Sort

upGrad

64	25	12	22	11
11	25	12	22	64
11	12	25	22	64
11	12	22	25	64
11	12	22	25	64

Hands-on Coding

- **Implement selection sort in Java.**

- Consider you have an array of natural numbers of size x. You also have an integer variable “i”. Write a selection sort algorithm for the given array in such a way that the state of the array after “i” iterations gets printed.

- Consider you have an array of natural numbers of size x. You first need to sort the given array in ascending order using selection sort and later print out the indices of the sorted elements as per their original indices (Note: The index starts from 0).

Please look at the animation carefully and figure out if it is Selection sort?

Poll 6 (15 Sec.)

What is the worst case time complexity of selection sort?

1. O(n)
2. O(logn)
3. $O(n^2)$
4. $O(n^3)$

Poll 6 (Answer)

What is the worst case time complexity of selection sort?

1. O(n)
2. O(logn)
3. **$O(n^2)$**
4. $O(n^3)$

Poll 7 (15 Sec.)

Is selection sort a stable sort?

1. Yes
2. No

Poll 7 (Answer)

Is selection sort a stable sort?

1. Yes
2. **No**

Poll 8 (15 Sec.)

Is selection sort an in-place sorting algorithm?

1. Yes
2. No

Poll 8 (Answer)

Is selection sort an in-place sorting algorithm?

1. **Yes**
2. No

upGrad

Insertion Sort

Insertion sort is another trivial sorting algorithm that is inspired by the way we arrange cards in our hands.

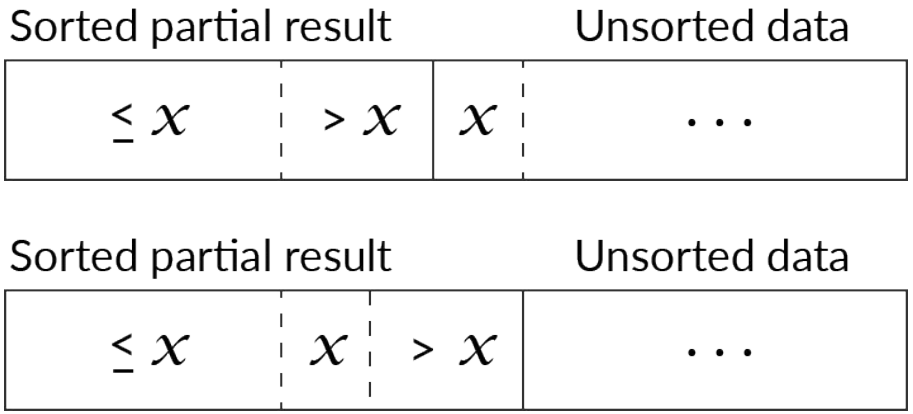
If we are to pause the execution of insertion sort and take a snapshot of the array, we will be able to see three distinct portions (from left to right):

- The sorted portion of the array,
- The data element under consideration, and
- The unsorted portion of the array.

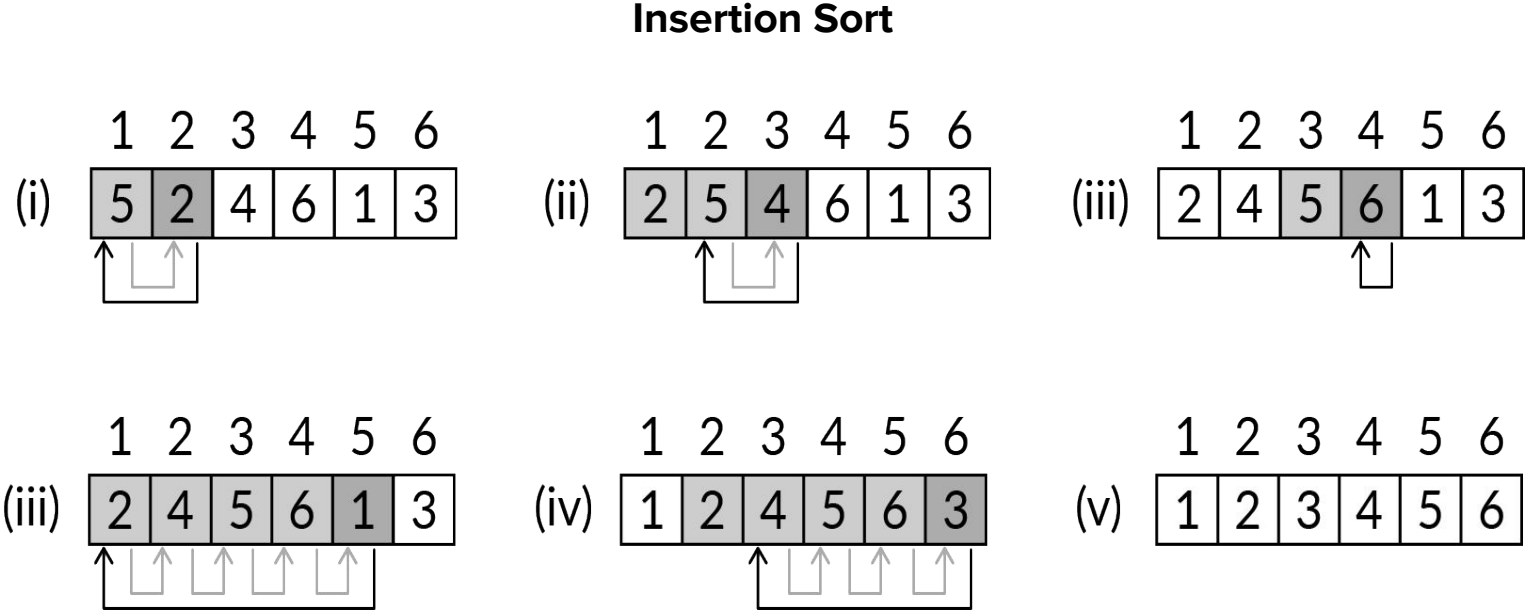
Insertion Sort

In each iteration, we pick up the left-most data element from the unsorted portion of the array and then place it in its correct place in the sorted portion of the array.

We keep on repeating this operation until the size of the unsorted portion of the array shrinks to 0.



Is insertion sort a brute force algorithm?


Insertion Sort
1 2 3 4 5 6
(i) 5 2 4 6 1 3
(ii) 2 5 4 6 1 3
(iii) 2 4 5 6 1 3
(iii) 2 4 5 6 1 3
(iv) 1 2 4 5 6 3
(v) 1 2 3 4 5 6

Hands-on Coding

- **Implement insertion sort in Java.**

- Consider you have an array of natural numbers of size x. You also have an integer variable “i”. Write a selection sort algorithm for the given array in such a way that the state of the array after “i” iterations gets printed.

- Consider an array of natural numbers of size x. Write an insertion sort algorithm for the given array and print out an array containing the original indices of the elements in the sorted array. (Note: The index starts from 0)

Poll 9 (30 Sec.)

What is the worst-case time complexity of insertion sort?

1. O(n)
2. O(logn)
3. $O(n^2)$
4. $O(n^3)$

Poll 9 (Answer)

What is the worst-case time complexity of insertion sort?

1. O(n)
2. O(logn)
3. **$O(n^2)$**
4. $O(n^3)$

Poll 10 (30 Sec.)

Is insertion sort a stable sorting algorithm?

1. Yes
2. No

Poll 10 (Answer)

Is insertion sort a stable sorting algorithm?

1. **Yes**
2. No

Poll 11 (15 Sec.)

Is insertion sort an in-place sorting algorithm?

1. Yes
2. No

Poll 11 (Answer)

Is insertion sort an in-place sorting algorithm?

1. **Yes**
2. No

Homework

1. Write a program to sort an array in descending order using bubble sort.

1. Write a program to sort an array in descending order using selection sort.

Tasks to complete after the session

Homework
MCQs
Coding Questions

upGrad

#RahoAmbitious

Thank You!