Selection Sort and Insertion Sort

Arnab Ganguly, Assistant Professor
Department of Computer Science, University of Wisconsin – Whitewater
Data Structures (CS 223)

Simulation Weblink

https://visualgo.net/bn/sorting

1 Introduction

In the binary search lecture, we saw that the array has to be sorted, i.e., given an array A of n-integers, we should have $A[0] \leq A[1] \leq A[2] \leq \cdots \leq A[n-1]$. Therefore, given an unsorted data, we need to first sort it, in order to be able binary-search it. Besides binary search, sorting has plenty of other applications – finding whether there is any duplicate element in a collection, finding two elements in the collection that are closest to each other, etc.

2 Selection Sort

Main Observation: To sort an array A, position 0 should contain the minimum element in the array, position 1 should contain the second minimum in the array, and so on.

Algorithm: The selection sort algorithm is precisely an implementation of the above observation, and can be described as follows.

- For each position $i = 0, 1, 2, \ldots, n-2$, repeat the following step.
- Find a position $minIndex$ in the sub-array $A[i, n-1]$ that contains the minimum value in this sub-array. If $i \neq minIndex$, then swap the values at positions i and $minIndex$ of A.

Table 1: Illustration of Selection Sort. Shows the array at the start and at the end of iteration of outer loop that goes from $i = 0$ to $i = n-2$.

Iteration	i	**Array at start**	**Min value in** $A[i, n-1]$	**Array at end**
1	0	$[5, 6, 1, 2, 0]$	0	$[0, 6, 1, 2, 5]$
2	1	$[0, 6, 1, 2, 5]$	1	$[0, 1, 6, 2, 5]$
3	2	$[0, 1, 6, 2, 5]$	2	$[0, 1, 2, 6, 5]$
4	3	$[0, 1, 2, 6, 5]$	5	$[0, 1, 2, 5, 6]$

Analysis: The algorithm has two nested for-loops – outer one going from $i = 0$ to $i = n-2$, and the other inner one for finding the minimum value in the sub-array $A[i, n-1]$. Therefore, the worst-case complexity is $O(n^2)$. Note that it does not matter what our input array is, the two for-loops will always run till their upper limit, making the algorithm $O(n^2)$, even in best case.

3 Insertion Sort

Main Observation: Suppose, we have sorted $A[0, i]$, where $n-1 > i > 0$. Then, to sort $A[0, i+1]$, we have to simply place $A[i+1]$ correctly within $A[0, i]$.

Algorithm: The insertion sort algorithm is precisely an implementation of the above observation, and can be described as follows.

- For each position $i = 1, 2, \dots, n-1$, repeat the following steps.
- Let $j = i$ and $temp = A[j]$.
- While $j > 0$ and $temp < A[j-1]$, assign $A[j] = A[j-1]$ and decrement j by one.
- Assign $A[j] = temp$.

Table 2: Illustration of Insertion Sort. Shows the array at the start and at the end of iteration of outer loop that goes from $i = 0$ to $i = n-2$.

Iteration	i	**Array at start**	$j = i$ at beginning	**Array at end**
1	1	$[5, 6, 1, 2, 0]$	1	$[5, 6, 1, 2, 0]$
2	2	$[5, 6, 1, 2, 0]$	2	$[1, 5, 6, 2, 0]$
3	3	$[1, 5, 6, 2, 0]$	3	$[1, 2, 5, 6, 0]$
4	4	$[1, 2, 5, 6, 0]$	4	$[0, 1, 2, 5, 6]$

Analysis: The algorithm has two loops – a for loop running from $i = 1$ to $i = n-1$, and a nested while loop running from $j = i$ to $j = 1$. In the worst case, therefore, insertion sort has $O(n^2)$ complexity, same as selection sort. However, if we provide a sorted array as input, observe that $temp < A[j-1]$ is always false (since $A[0] \leq A[1] \leq \cdots A[j-1] \leq A[j] \leq \cdots \leq A[n-1]$). Therefore, for a sorted array, the while loop never executes, leading to a best case complexity of $O(n)$.