UPPSALA
UNIVERSITET

Algorithms and Data Structures

Assignment 2

Rim Sleimi

October 6, 2021

Insertion Sort

Insertion sort is very simple to implement, is non-recursive, stable. Works by inserting elements from an unsorted list into a sorted subsection of the list, one item at a time.

Advantages

- **Intuitive and Naive sorting approach**
- **Fast** When the input array is already sorted, insertion sort runs in $O(n)$ time. It is also faster for small inputs since it can start sorting directly without having to split the array or rearrange it beforehand.
- **Space efficient** Insertion sort can be done in-place.

Disadvantages

- **Slow** Insertion sort usually takes $O(n^2)$ time. Too slow to be used on large data sets, making it useless for anything but small arrays.

Quick Sort

Quick sort is based on the divide-and-conquer principle. It recursively partitions the array of items into sublists based on a pivot element. Elements in the first sublist are arranged to be smaller than the pivot, while all elements in the second sublist are arranged to be larger than the pivot. Both partitioning and arranging are repeated until all items are sorted.

Advantages

- **Fast** On average, it runs in $O(n \log(n))$ time, which scales well as n grows. It also outperforms the other two algorithms for very large inputs.
- **Parallelizable** Quicksort divides the input into two sections, each of which can be sorted at the same time in parallel.

Disadvantages

- **Slow Worst-Case** Even though it is rare, the worst case can take $O(n^2)$ time. This makes quicksort undesirable in cases where any slow performance is unacceptable and heap sort is usually favoured in such cases.

- **Space inefficiency** Quick sort is in-place, but it requires a stack because it is a divide-and-conquer algorithm (i.e, its recursive nature). Making the space complexity bounded by $O(\log(n))$

Heap Sort

In heap sort the largest item is repeatedly selected and moved to the end of our array. Therefore, instead of scanning through the entire array to find the largest item, the array is converted into a max heap to speed things up.

Advantages

- **Fast** Heap sort is $O(n \log(n))$ because it does not care about the initial state nor the size of the array. Therefore it scales well as n grows. Unlike quick-sort, there's no worst-case $O(n^2)$ complexity.
- **Consistency** Heap sort exhibits consistent performance. It performs equally well in the best, average and worst cases, regardless of the size of the input array.
- **Space efficient** Its primary advantage is that it can work in-place and be implemented iteratively, with no additional memory requirements. In certain time-critical situations (e.g the worst case of quicksort), its fixed complexity can be useful.

Table 1: Summary of the time and space complexity of Insertion sort, quick sort, and heap sort

Algorithm	**Data Structure**	**Space**	**Best**	**Average**	**Worst**
Insertion Sort	Array	O(1)	O(n)	$O(n^2)$	$O(n^2)$
Quick Sort	Array	O(log(n))	O(n log(n))	O(n log(n))	$O(n^2)$
Heap Sort	Array	O(1)	O(n log(n)))	O(n log(n)))	O(n log(n)))