Algorithms and Data Structures 1
Assignment 2

1DL210

Uppsala University

Daniel Brown and Yashas Koppa Ramesh

October 6, 2021

UPPSALA
UNIVERSITET

1 Comparison

Algorithm	Best-case	Average-case	Worst-case	Already-sorted case
Insertion sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Quick sort	$\Theta(n^2)$	$\Theta(n\log(n))$	$\Theta(n\log(n))$	$\Theta(n^2)$
Heap sort	$\Theta(n\log(n))$	$\Theta(n\log(n))$	$\Theta(n\log(n))$	$\Theta(n\log(n))$

The three algorithms whose tight-bound execution times are summarised above, namely Insertion Sort, Quick Sort and Heap Sort, are all in-place algorithms. Heap Sort requires a heap data structure, a type of binary tree, whereas Insertion and Quick Sort can execute their functionalities of replacement and swap without one. Each algorithm has pros and cons over each other, and some of these are listed below:

1. Insertion Sort
   - Pros
     (a) Simple to implement and does not have recursive loops.
     (b) The Best-Case Complexity is $\theta(n)$ which implies a linear runtime.
     (c) Low overhead which makes it suitable for small-sized arrays.
   - Cons
     (a) The Worst-Case Complexity is $\theta(n^2)$ which means that this algorithm is not suited for large-sized arrays.
   - Most advantegeous for small-sized arrays.
2. Quick Sort
   - Pros
     (a) Fast in its implementation for most inputs.
     (b) For sorting n elements, the Best-Case Complexity is $\theta(n\log(n))$.
     (c) Very efficient on the average with small constant factors.
     (d) Works well in virtual-memory environments.
   - Cons
     (a) The Worst-Case Complexity is quadratic, that is, $\theta(n^2)$ which means that this algorithm with recursion has a complex implementation.
     (b) Asymptotic speed dependent upon whether partitioning is balanced or unbalanced.
     (c) Not efficient for small-sized arrays.

- Most advantageous when a good average speed is needed (sorting multiple arrays).

3. Heap Sort

   - Pros
     (a) Efficient in the way it works
     (b) With a $\theta(n \log(n))$ guaranteed speed for worst, average and best-case scenarios, this algorithm is consistent.
     (c) Works well with large-sized inputs.

   - Cons
     (a) Complex memory management can result in slower execution times in comparison to other sorting algorithms.
     (b) Large constant factors mean that it is usually beaten by quicksort in practice.

   - Most advantageous when response time is critical and the algorithm must sort with a guaranteed speed.