Software Systems Lab: OutLab
LaTeX

Name: Ashwin Abraham
Roll no: 210050023

September 6, 2022

Contents

Chapter 1

Abstract

1.1 About this report

This report was made by me as a part of my CS 251 course (Software Systems Lab). This particular document was made as a part of the Outlab for the module on LaTeX.

1.2 Introduction

In this report, I was supposed to make a report in LaTeX showcasing my learning from previous labs in this report. The following features were mandatory in this Outlab:

1. **Frontpage**
2. **Table of Contents**
3. **Sections and Subsections**
4. **Table of Contents**
5. **Sections and Subsections**
6. **Use of Itemization**
7. **Images with Captions**
8. **A Table with Captions**
9. **Algorithm and Code**
10. **A minipage**

11. **A Bibliography**

The following points were also to be noted:

- The report should be of **at least 5 pages** (10 points), including the front page and table of contents.
- You can use the article or report document class, and font size: 10-12pt.
- You can use the Overleaf tool (An online LaTeX editor).
- You can use online resources but don't forget to mention it in the readme and bibliography.
- Use **bold**, *italic* and underlined words wherever needed.
  Also use new lines/paragraphs, proper spacing, etc..
  You've to be creative with your work.

1.3 Background in LaTeX

1.3.1 Work done before this Module

In the **PH 107** (*Quantum Physics and Applications*) and **CH 107** (*Quantum Chemistry*) courses, I used LaTeX in order to submit assignments. These assignments can be found in my GitHub: here (PH 107), and here (CH 107) [1]. LaTeX was aso used by me to prepare my *Summer of Science* plan of action, mid-term and end-term reports on **Machine Learning**, which can be found here. It was also used for the **CS 215** (*Data Analysis and Interpretation*) course assignments.

1.3.2 Work done in this module

As a part of the inlab for this module, we were asked to make a presentation using the *Beamer* package of LaTeX, which is used for making slideshows and presentations. Then in the outlab, we were asked to make this report.

Chapter 2

Use of Itemization[4]

2.1 Itemization in Ordered Lists

1. Comparison based sorting algorithms
   These algorithms will always have $\Omega(n \log n)$ comparisons.
   (a) Sorting algorithms that are $O(n^2)$ on average
      i. Bubble Sort
      ii. Insertion Sort
      iii. Selection Sort
      iv. Comb Sort
      v. Exchange Sort
      vi. Shell Sort
   (b) Sorting algorithms that are $O(n \log n)$ on average
      i. Merge Sort
      ii. Heap Sort
      iii. Quick Sort
2. Distribution based sorting algorithms
   These algorithms can be more efficient than $n \log n$ and are all $\Omega(n)$.
   However they make some assumptions about the distribution of the elements of the array.
   (a) Counting Sort
      Here it is assumed that the elements of the array belong to a set S of possibilities.
      It's time complexity is then $O(n + |S|)$.

(b) Bucket Sort
Here we divide the array into k buckets, and sort the buckets, and then merge them.
It's time complexity is $O(n + \frac{n^2}{k} + k)$.

(c) Radix Sort
Here we assume that the elements are to be sorted lexicographically.
It's time complexity is $O(wn)$ where w is the length of each element.

2.2 Itemization in Unordered Lists

The same thing, but as an unordered list.

- Comparison based sorting algorithms
  These algorithms will always have $\Omega(n \log n)$ comparisons.
  - Sorting algorithms that are $O(n^2)$ on average
    * Bubble Sort
    * Insertion Sort
    * Selection Sort
    * Comb Sort
    * Exchange Sort
    * Shell Sort
  - Sorting algorithms that are $O(n \log n)$ on average
    * Merge Sort
    * Heap Sort
    * Quick Sort
- Distribution based sorting algorithms
  These algorithms can be more efficient than $n \log n$ and are all $\Omega(n)$.
  However they make some assumptions about the distribution of the elements of the array.
  - Counting Sort
    Here it is assumed that the elements of the array belong to a set S of possibilities.
    It's time complexity is then $O(n + |S|)$.

– Bucket Sort
  Here we divide the array into k buckets, and sort the buckets, and then merge them.
  It's time complexity is $O(n + \frac{n^2}{k} + k)$.
– Radix Sort
  Here we assume that the elements are to be sorted lexicographically.
  It's time complexity is $O(wn)$ where w is the length of each element.

Chapter 3

Images with Captions

Have a look at these images with captions related to sorting.

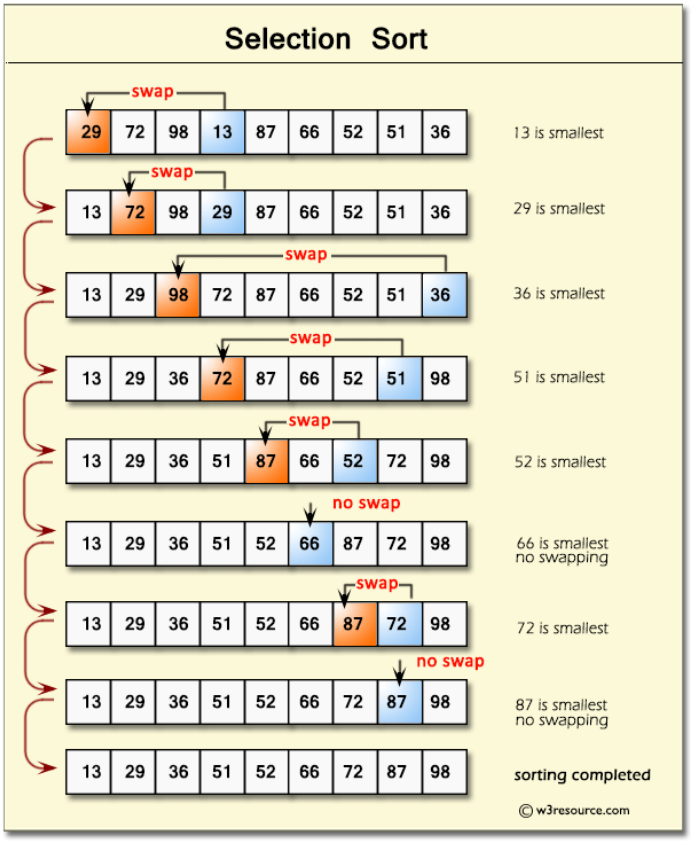

Figure 3.1: Selection Sort[6]

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)

Figure 3.2: Time Complexities of some Sorting Algorithms[5]

Chapter 4

A Table with Captions

Sorting Algorithms				
Algorithm	Best Case Complexity	Average Case Complexity	Worst Case Complexity	Memory Complexity
Bubblesort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Insertion Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Comb Sort	$\Omega(n \log n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Exchange Sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Shell Sort	$\Omega(n \log n)$	$\Theta(n^{\frac{4}{3}})$	$O(n^{\frac{3}{2}})$	$O(1)$
Merge Sort	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n \log n)$	$O(n)$
Heap Sort	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n \log n)$	$O(1)$
Quick Sort	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n^2)$	$O(\log n)$
Counting Sort	$\Omega(n + r)$	$\Theta(n + r)$	$O(n + r)$	$O(n + r)$
Bucket Sort	$\Omega(n + r)$	$\Theta(n + r)$	$O(n + r)$	$O(n + r)$
Radix Sort	$\Omega(n)$	$\Theta(n\frac{k}{d})$	$O(n\frac{k}{d})$	$O(n + 2^d)$

Table 4.1: Some popular sorting algorithms and their complexities[4]

Chapter 5

Algorithm and Code

Here we will discuss an algorithm to print the first n Fibonacci numbers. This was part of the second inlab of **CS 251**. The pseudocode of this algorithm is as follows:

1: $x \leftarrow 0$
2: $y \leftarrow 1$
3: **for** $k \leftarrow 1$ to n **do**
4: **print** x
5: $temp \leftarrow x$
6: $x \leftarrow y$
7: $y \leftarrow y + temp$
8: **end for**

The implementation of this algorithm as a *Bash* Script [2] is as follows:

Listing 5.1: Calculating the Fibonacci Numbers

```
1  #! /bin/bash
2
3  x=0
4  y=1
5  str=""
6  num=$(($1))
7  for ((i=0; i<$num; i++))
8  do
9      str+="$x "
10     temp=$(($x))
11     x=$(($y))
12     y=$(($y+$temp))
13 done
14 echo $str
```

Chapter 6

Minipage

The following is a boxed minipage.

Lorem ipsum[3] dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Bibliography

[1] Ashwin Abraham. My GitHub, 2021.

[2] Ashwin Abraham. Cs 251 - previous labs, 2022.

[3] Marcus Tullius Cicero. *De finibus bonorum et malorum*. Loeb Classical Library, 50s BC.

[4] Wikipedia Editors. The wikipedia article on sorting algorithms.

[5] Matteo Kimura Leonardo Galler. Sorting algorithms. *Laboratório de Aprendizado de Máquina em Finanças e Organizações*, 2019.

[6] w3 resources. Selection sort.