Algorithms and datastructures
Exercises

Kristoffer Klokker

2022

Contents

5 Week

5.1 For each function $f(n)$ and time t in the following table, determine the alrgest size n of a problem that can be solved in time t, assuming that the algorithm to solve the problem takes $f(n)$ 1 nanosecond

	1s	1hour	1year	1 centuray
n	10^9	$6 \cdot 10^{10}$	$3.2 \cdot 10^{16}$	$3.2 \cdot 10^{18}$
$n \log_2 n$	$4 \cdot 10^7$	$10^9 \frac{1}{\text{s}} \cdot 3600\text{s} = n \cdot \log_2(n) \rightarrow n = 9.8 \cdot 10^{10}$	$6.4 \cdot 10^{14}$	$5.6 \cdot 10^{16}$
n^2	31622	$1.8 \cdot 10^6$	$1.7 \cdot 10^8$	$1.8 \cdot 10^9$
n^3	10^3	15326	316010	$1.4 \cdot 10^6$
2^n	30	41.7	54.8	61.5

5.2 Show that in a puzzle where two peices is switched with n pieces in all wrong positions, it requires at minimum of $n/2$ switches to solve the puzzle

For a puzzle with no correct positions in advance, the lowest amounts of move will be in the scenario where every piece's correct position has to piece of its current position. Which therefore will result in $n/2$ amounts of moves is needed.

5.3 Create a puzzle with 4 pieces, and find a sequence of switches, but where not every switch moves at least one piece to its correct position

4	1
2	3

$4 \rightarrow 2, 3 \rightarrow 2, 1 \rightarrow 2$

As seen here this method does no use the greedy method but it still use the same amount of moves.

5.4 Create an algorithm which can find cycles in a given puzzle

The algorithm takes a list, and creates a variable counter for the amount of cycles.
It then goes through every entry, if the entry is not -1 then it calls a recursive function with the entry index and list.
The list then check if the given entry is -1 if not then set the entry to -1 and then calls itself with the entries last index and the list.
When the function returns it add 1 to the cycle.
Then it returns the amount of cycles
This algorithm will run at $O(2n)$ if the cycle is 1 and it then has to move every entry and go through the rest of the list.

5.5 Use the algorithm implementation to calculate statistic over the amount of cycles in a 16 long permutation

Cycles	Amount	Chance of occurrence
1	6249	6%
2	20716	21%
3	29600	30%
4	24131	24%
5	13038	13%
6	4721	5%
7	1252	1%
8	256	0%
9	29	0%
10	7	0%
11	1	0%
12	0	0%
13	0	0%
14	0	0%
15	0	0%
16	0	0%

Average	3.37675

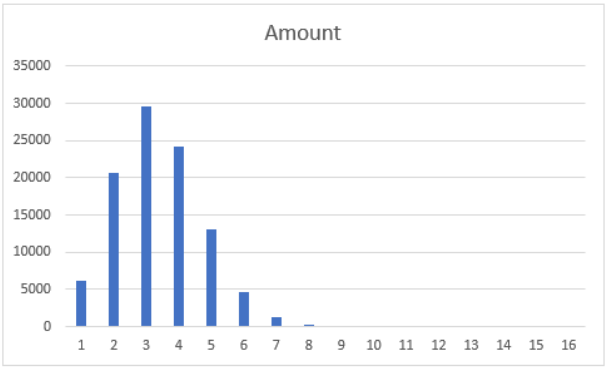

Figure 1: Statistic from puzzleSolve/data.csv

5.6 Write insertion sort pseudo code

1: Linear search(A, v)
2: $i = 0$
3: **while** $i < A.length$ && $A[i]! = v$ **do**

4: $i++$
5: **end while**
6: return i

6 Week

6.1 What is the average and worst case run time og linear search aglorithm with the element placed randomly

Average: on average the run time will be $n/2$
Worst: if the element is at the end of the list it will be n

6.2 Let an inversion be that in an array if $i < j$ and $A[i] > A[j]$

6.2.a Find inversion pairs in $\{2, 3, 8, 6, 1\}$

$(2, 1), (3, 1), (8, 1), (8, 6), (6, 1)$

6.2.b For which array will it have the most inverse pairs and how many in an array of length n

The backwards sorted array, which will have $\frac{n^2-n}{2}$ pairs.

6.2.c What is the relation between inversion pairs and insertion sort

The relation is that insertion sort use the same amount of operations in the worst case scenario as inverse pairs.

6.3 Analyse the run time of insertion sort, in best case, worst case and random case

Here 1000 arrays was used from which random length of arrays was used. Here the are the results of the time divided by length average.

- Best - $4.31 \cdot 10^{-6}$
- Worst - 0.016

- Random - 0.008

As seen the random is closest to the worst case.

6.4 Find an algorithm which for a array with integers if there exists a pair which sum is equal to x

This is done by using a sort like merge sort which takes $n \cdot \log_2 n$ then two pointers where one is at the start and one at the end.
If the two pointers integer sum exceeds x the end moves to the left if less than x the start pointer moves to the right.
This will take n time and therefore the time will still just be $O(n \cdot \log_2 n)$

6.5 Illustrate merge sort using the array $A = \{3, 41, 52, 26, 38, 57, 9, 49\}$

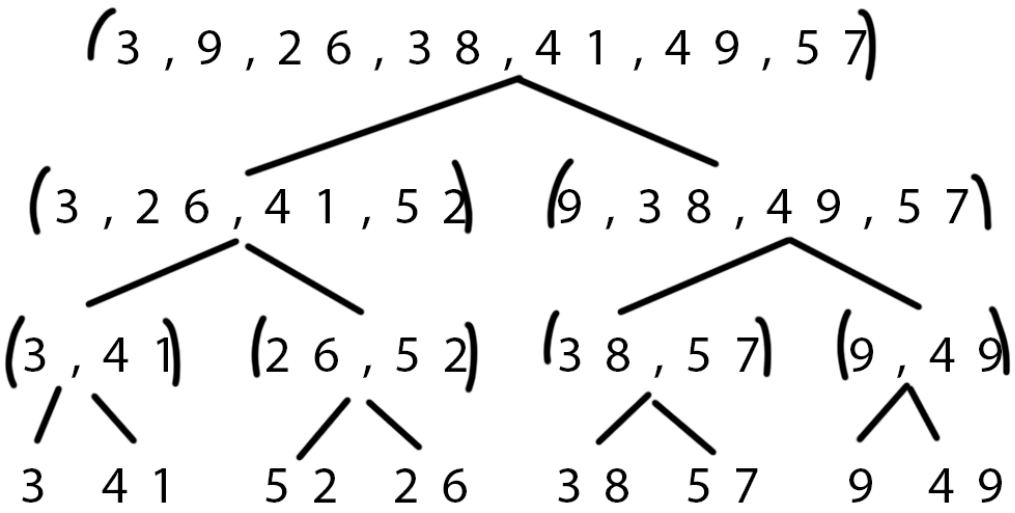

Figure 2: Illustration of merge sort

6.6 Show that for $f(n) = 0.1n^2+5n+25$ that $f(n) = \Theta(n^2)$ and $f(n) = o(n^3)$

$$\lim_{n\to\infty} \frac{0.1n^2 + 5n + 25}{n^2} = 0.1$$
$$\lim_{n\to\infty} \frac{0.1n^2 + 5n + 25}{n^3} = 0$$

Due to the first begin bigger than 0 it means that $f(n) = \Theta(n^2)$ and due to the other being zero means that $f(n) = o(n^3)$

6.7 Prove that $max(f(n), g(n)) = \Theta(f(n) + g(n))$

Due to the max function the highest resullt $g(n)$ can maximally be equal to $f(n)$.
Therefore if $f(n) = n^2$ $g(n)$ can maximally be n^2 and therefore the addition will result in the same run time.

6.8 Draw binary search, write pseudo code and then code

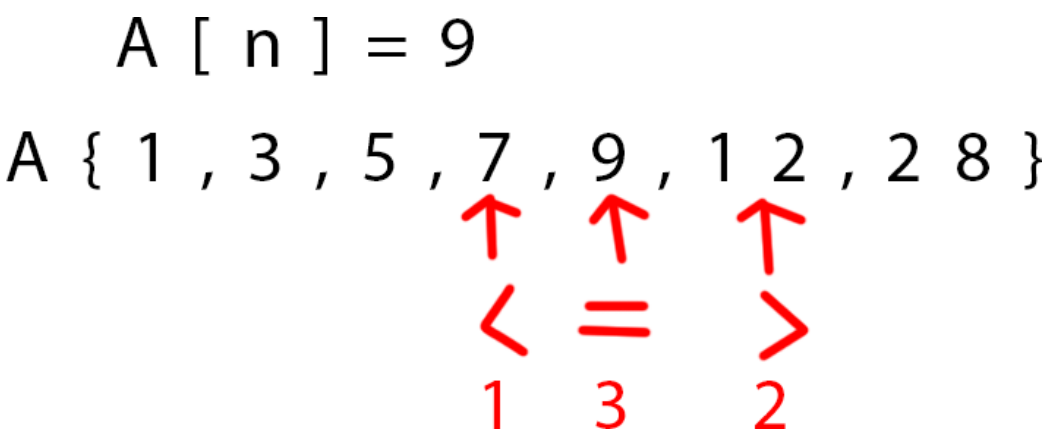

Figure 3: Illustration of binary search

1: Binary search(A, v)
2: $i = A.length/2$
3: $j = A.length/2 + A.length\%2$
4: **while** $iA[i]! = v \mid\mid j == 0$ **do**
5: $\quad j = j/2 + j\%2$
6: $\quad$ **if** $A[i] < v$ **then**
7: $\quad\quad i = i + j$
8: $\quad$ **else**
9: $\quad\quad i = i - j$
10: $\quad$ **end if**
11: **end while**
12: return i

6.9 How can binary search be used to optimize linear search to $O(n \log_2 n)$?

By instead of linearly going down and finding a number which the current is lower than, a binary search can be done to a number which is lower.
By this every entry will be performed a binary search upon an such the run time will be $n \log_2 n$

6.10 Is $2^{n+1} = O(2^n)$? Is $2^{2n} = O(2^n)$?

$2^{n+1} = 2^n \cdot 2^1$ therefore $2^{n+1} = O(2^n)$
$\lim_{n\to\infty} \frac{2^n}{2^{2n}} = 0$ therefore $2^n = o(2^{2n})$

6.11 Prove $\log(n!) = \Theta(n \log n)$

$\lim_{n\to\infty} \frac{\log(n!)}{n \log(n)} = 1$ therefore $\log(n!) = \Theta(n \log n)$

6.12 Prove $n! = \omega(2^n)$

$\lim_{n\to\infty} \frac{2^n}{n!} = 0$ therefore $2^n = o(n!) \to n! = \omega(2^n)$

6.13 Prove $n! = o(n^n)$

$\lim_{n\to\infty} \frac{n!}{n^n} = 0$ therefore $n! = o(n^n)$