

Algorithms and datastructures Exercises

Kristoffer Klokke

2022

Contents

5	Week	4
5.1	For each function $f(n)$ and time t in the following table, determine the largest 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	4
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	4
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
5.4	Create an algorithm which can find cycles in a given puzzle . .	5
5.5	Use the algorithm implementation to calculate statistic over the amount of cycles in a 16 long permutation	5
5.6	Write insertion sort pseudo code	5
6	Week	6
6.1	What is the average and worst case run time og linear search aglorithm with the element placed randomly	6
6.2	Let an inversion be that in an array if $i < j$ and $A[i] > A[j]$.	6
6.2.a	Find inversion pairs in $\{2, 3, 8, 6, 1\}$	6
6.2.b	For which array will it have the most inverse pairs and how many in an array of length n	6
6.2.c	What is the relation between inversion pairs and insertion sort	6
6.3	Analyse the run time of insertion sort, in best case, worst case and random case	6
6.4	Find an algorithm which for a array with integers if there exists a pair which sum is equal to x	7
6.5	Illustrate merge sort using the array $A = \{3, 41, 52, 26, 38, 57, 9, 49\}$	7
6.6	Show that for $f(n) = 0.1n^2 + 5n + 25$ that $f(n) = \Theta(n^2)$ and $f(n) = o(n^3)$	7
6.7	Prove that $\max(f(n), g(n)) = \Theta(f(n) + g(n))$	8
6.8	Draw binary search, write pseudo code and then code	8
6.9	How can binary search be used to optimize linear search to $O(n \log_2 n)$?	9
6.10	Is $2^{n+1} = O(2^n)$? Is $2^{2n} = O(2^n)$?	9
6.11	Prove $\log(n!) = \Theta(n \log n)$	9
6.12	Prove $n! = \omega(2^n)$	9

6.13	Prove $n! = o(n^n)$	9
7	Week	9
7.1	Rank the function speed from fastest growing to slowing . . .	9
7.2	If $f_1(n) \in O(g_1(n))$ and $f_2(n) \in O(g_2(n))$ which statements is true	10
7.3	Describe an algorithm which find the number of tuples in an array which has a lower value than another element but higher index in the run time $n \log_2 n$	11
7.4	Which of the following statements are true	11
8	Week	11
8.1	Illustrate the partitioning in quick sort on the following array	11
8.2	In an array with only the same value, where will quick sort return the middle value	12
8.3	What is the run time of quick sort on the array of the same value	12
8.4	Is an sorted array a min-heap	12
8.5	Is the following array a max-heap?	13
8.6	Insert 9 into the following max-heap tree	13
8.7	Insert 2 into the following min-heap	13
8.8	Illustrate Max-Heapify($A, 2$) on the following array	13
8.9	Use Heap-Extract-Max(A) on the following array	14
8.10	Where in a max heap will the smallest element reside	14
8.11	Find all mini-heaps with the elements 1,2,3,4	14
8.12	Prove that the childrens index relative to the parent is index times two and index times two plus 1	15
8.13	Analysis if d-ary heap	15
8.13.a	What would be the array representation	15
8.13.b	What is the height of a tree with n elements with d children	15
8.13.c	Rewrite heap sort such it works with d-ary heaps . . .	15

5 Week

5.1 For each function $f(n)$ and time t in the following table, determine the largest 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 century
n	10^9	$6 \cdot 10^{10}$	$3.2 \cdot 10^{16}$	$3.2 \cdot 10^{18}$
$n \log_2 n$	$4 \cdot 10^7$	$10^{9\frac{1}{s}} \cdot 3600s = n \cdot \log_2(n) \rightarrow n = 9.8 \cdot 10^{10}$	$6.4 \cdot 10^{14}$	$5.6 \cdot 10^{16}$
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 pieces 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 not 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.

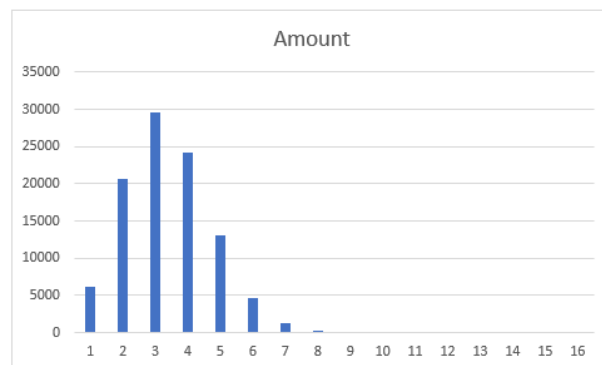
When the function returns it add 1 to the cycle.

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** $i < A.length$ && $A[i] \neq v$ **do**

```

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

```

6 Week

6.1 What is the average and worst case run time of linear search algorithm 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 uses 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 were used from which random length of arrays was used. Here 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 $A = \{3, 41, 52, 26, 38, 57, 9, 49\}$

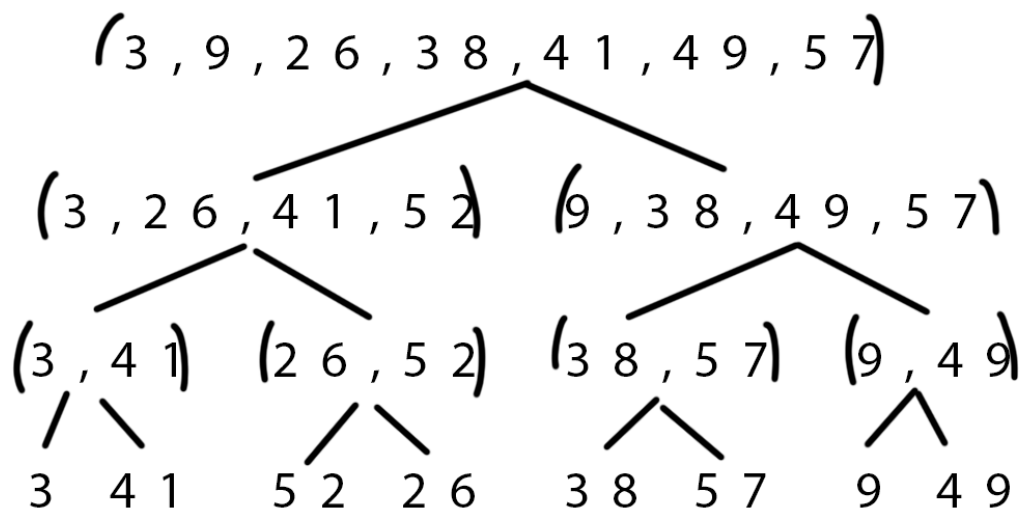


Figure 2: Illustration of merge sort

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

$$\lim_{n \rightarrow \infty} \frac{0.1n^2 + 5n + 25}{n^2} = 0.1$$

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

Due to the first being 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 result $g(n)$ 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

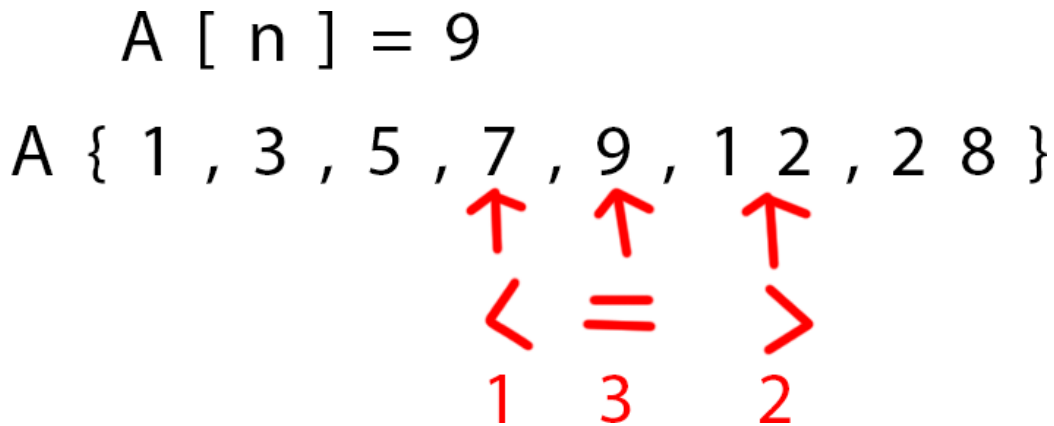


Figure 3: Illustration of binary search

```

1: Binary search(A, v)
2: i = A.length/2
3: j = A.length/2 + A.length%2
4: while iA[i] != v || j == 0 do
5:     j = j/2 + j%2
6:     if A[i] < v then
7:         i = i + j
8:     else
9:         i = i - j
10:    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 $2^{n+1} = O(2^n)$

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

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

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

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

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

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

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

7 Week

7.1 Rank the function speed from fastest growing to slowing

$$\sqrt{n}, 2^n, \log_{10}^2 n, \log_2 n$$

$$\lim_{n \rightarrow \infty} \frac{f(x)}{g(x)}$$

$g(n) \setminus f(n)$	\sqrt{n}	2^n	$\log_{10}^2 n$	n	$\log_2 n$
\sqrt{n}	1	∞	0	∞	0
2^n	0	1	0	0	0
$\log_{10}^2 n$	∞	∞	1	∞	0
n	0	∞	0	1	0
$\log_2 n$	∞	∞	∞	∞	1

This can be rearranged from a clear order is made.
This is therefore the order of fastest growing to slowest $2^n, n, \sqrt{n}, \log_{10}^2 n, \log_2 n$

$g(n) \setminus f(n)$	2^n	n	\sqrt{n}	$\log_{10}^2 n$	$\log_2 n$
2^n	1	0	0	0	0
n	∞	1	0	0	0
\sqrt{n}	∞	∞	1	0	0
$\log_{10}^2 n$	∞	∞	∞	1	0
$\log_2 n$	∞	∞	∞	∞	1

7.2 If $f_1(n) \in O(g_1(n))$ and $f_2(n) \in O(g_2(n))$ which statements is true

1. $f_1(n) + f_2(n) \in O(g_1(n) + g_2(n))$
2. $g_1(n) + g_2(n) \in \Omega(f_1(n) + f_2(n))$
3. $\frac{f_1(n)}{f_2(n)} \in O(\frac{g_1(n)}{g_2(n)})$

The first statement is true due to if $f_1(n) \Omega f_2(n)$ then $g_2(n)$ will simply be a constant when added to $g_2(n)$, which also account for the other way around.
The second is true due to being the inverse of the first statement.
The third is not true, in the following assignment $f_1(n) = n^2, f_2(n) = n^2, g_1(n) = n^2, g_2(n) = n^n$ in this scenario the left will go towards 1 and the other will to 0.

7.3 Describe an algorithm which find the number of tuples in an array which has a lower value than another element but higher index in the run time $n \log_2 n$

This could be done with merge sort. Here when comparing two elements if an element is chosen from the array which comes first then a tuple exists with every element left in the other array.

7.4 Which of the following statements are true

1. $n^2 \in \Omega(n)$
2. $n \in \Theta(n^2)$
3. $n \log n \in o(n^2)$
4. $\log n \in O(\sqrt{n})$
5. $n! \in \omega(2^n)$

$$\lim_{n \rightarrow \infty} \frac{n^2}{n} = \infty \quad (1)$$

$$\lim_{n \rightarrow \infty} \frac{n^2}{n \log_2 n} = \infty \quad (2)$$

$$\lim_{n \rightarrow \infty} \frac{\sqrt{n}}{\log_2 n} = \infty \quad (3)$$

$$\lim_{n \rightarrow \infty} \frac{n!}{2^n} = \infty \quad (4)$$

The first statement is true according to (1) and the second is false due to (1) not being equal 1.

The third is true according to (2), likewise the fourth is true according to (3)

The fifth is also true according to (4).

8 Week

8.1 Illustrate the partitioning in quick sort on the following array

$$A = \{13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11\}$$

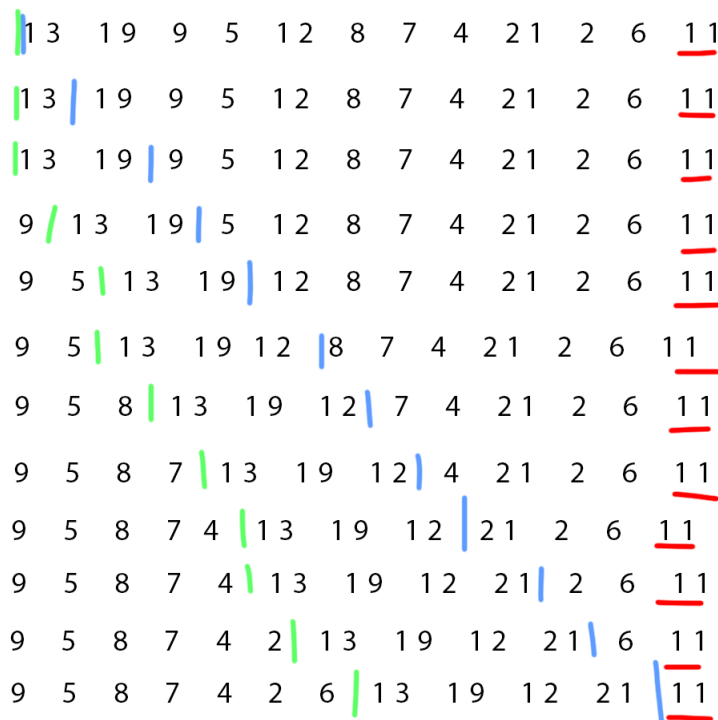


Figure 4: Quick sort partitioning on an array

8.2 In an array with only the same value, where will quick sort return the middle value

The returned placement will then be the length of the array -1 / the last element in the array.

8.3 What is the run time of quick sort on the array of the same value

This will either be the best case or worst case for the algorithm. This is due to if left partition is \leq nothing is moved. Whereas if the right partition is \geq it will have to move every element.

8.4 Is an sorted array a min-heap

Yes due to in a sorted array $2i$ and $2i + 1$ will always be lower or equal

8.5 Is the following array a max-heap?

$\langle 23, 17, 14, 6, 13, 10, 1, 5, 7, 12 \rangle$

It is not a valid max-heap due to 6's children is 5 and 7 which is not smaller.

8.6 Insert 9 into the following max-heap tree

$\langle 10, 8, 6, 3, 7, 4, 5, 1, 2 \rangle$

by inserting it on index 3 the following tree is made

$\langle 10, 8, 9, 3, 7, 4, 5, 1, 2 \rangle$

This can then be checked

$10 > 8, 9$

$8 > 3, 7$

$9 > 4, 5$

$3 > 1, 2$

It is therefore a valid max-heap

8.7 Insert 2 into the following min-heap

$\langle 1, 3, 5, 4, 10, 13, 7, 6, 17 \rangle$

$\langle 1, 2, 3, 5, 4, 10, 13, 7, 6, 17 \rangle$

$1 < 2, 3$

$2 < 5, 4$

$3 < 10, 13$

$5 < 7, 6$

$10 < 17$

8.8 Illustrate Max-Heapify($A, 2$) on the following array

$\langle 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0 \rangle$

$\langle 27, 17, \textcolor{red}{2}, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0 \rangle$

$2 > 13, 10$

$\langle 27, 17, 13, 16, 13, \textcolor{red}{2}, 10, 1, 5, 7, 12, 4, 8, 9, 0 \rangle$

$2 > 4, 8$

$\langle 27, 17, 3, 16, 13, 8, 10, 1, 5, 7, 12, 4, \textcolor{red}{2}, 9, 0 \rangle$

8.9 Use Heap-Extract-Max(A) on the following array

< 21, 18, 10, 12, 8, 9, 4, 7, 5, 2 >

< 2, 18, 10, 12, 8, 9, 4, 7, 5, 21 >
< 18, 2, 10, 12, 8, 9, 4, 7, 5, 21 >
< 18, 12, 10, 2, 8, 9, 4, 7, 5, 21 >
< 18, 12, 10, 7, 8, 9, 4, 2, 5, 21 >
< 5, 12, 10, 7, 8, 9, 4, 2, 18, 21 >
< 12, 5, 10, 7, 8, 9, 4, 2, 18, 21 >
< 12, 8, 10, 7, 5, 9, 4, 2, 18, 21 >
< 2, 8, 10, 7, 5, 9, 4, 12, 18, 21 >
< 10, 8, 2, 7, 5, 9, 4, 12, 18, 21 >
< 10, 8, 9, 7, 5, 2, 4, 12, 18, 21 >
< 4, 8, 9, 7, 5, 2, 10, 12, 18, 21 >
< 9, 8, 4, 7, 5, 2, 10, 12, 18, 21 >
< 2, 8, 4, 7, 5, 9, 10, 12, 18, 21 >
< 8, 2, 4, 7, 5, 9, 10, 12, 18, 21 >
< 8, 7, 4, 2, 5, 9, 10, 12, 18, 21 >
< 5, 7, 4, 28, 9, 10, 12, 18, 21 >
< 7, 5, 4, 28, 9, 10, 12, 18, 21 >
< 2, 5, 47, 8, 9, 10, 12, 18, 21 >
< 5, 2, 47, 8, 9, 10, 12, 18, 21 >
< 4, 25, 7, 8, 9, 10, 12, 18, 21 >
< 24, 5, 7, 8, 9, 10, 12, 18, 21 >
< 2, 4, 5, 7, 8, 9, 10, 12, 18, 21 >

8.10 Where in a max heap will the smallest element reside

Due to the trees nature, a specific index is not known rather just it is at a leaf in the tree

8.11 Find all mini-heaps with the elements 1,2,3,4

< 4, 3, 2, 1 >
< 4, 2, 3, 1 >
< 4, 3, 1, 2 >

8.12 Prove that the childrens index relative to the parent is index times two and index times two plus 1

For a parent the position can be divided to the sum of the current height and the index of the level.

Ex the parent 13 can be written as $(8+5)$ where 8 is the height index and 5 is the offset.

The child index will then be on the next height therefore the double height index, and the offset will then be the double due to every sibling to the parent having 2 children.

Therefore in the example the parents child will be $2(8+5)$ therefore two times index, and the other child will have the offset of plus 1.

8.13 Analysis if d-ary heap

d-ary heaps have d children instead of two

8.13.a What would be the array representation

This would be the same but instead of children being at $2i$ and $2i+1$ it would be $3i$ and $3i+1$

8.13.b What is the height of a tree with n elements with d children

A level consist of 3^n where n is the height.

Therefore for a level it will be the sum of all previous level.

That series can be condensed to $\frac{3^n-1}{2}$

8.13.c Rewrite heap sort such it works with d-ary heaps

First of when referencing the children it will be di and $di + 1$

When moving keys around there will then be d checks for the largest value.

It can here be noted that the height of the tree was $\frac{3^n-1}{2}$ therefore making the run time $O(3^n \cdot n)$ due n exchanges being done at each level.