Suggested solution for DAT495 (and re-exam for DIT181, DIT182, LET375, DAT038, DAT037, TDA417, TDA416, and DAT525)

Datastrukturer och algoritmer

Saturday, 2022-06-04, 8:30–12:30, Lindholmen

Section 1: Complexity

Question 1A

a)
```
for i from 12 to n:
    // some statement with O(1) complexity
```

$O(n - 12) = O(n)$

b)
```
for i from 0 to n:
    j = 1
    while j < n:
        // some statement with O(n) complexity
        j = j * 2
```

Outer loop runs n times, inner loop $\log_2 n$ times and executes statements in $O(n)$ complexity for a total of $O(n) \cdot O(\log n) \cdot O(n) = O(n^2 \log n)$.

c)
```
i = n
while i > 0:
    for j from 0 to i:
        // some statement with O(1) complexity
    i = i - 1
```

We have $n + (n - 1) + (n - 2) + \ldots + 1 = n(n + 1) / 2 \in O(n^2)$.

Question 1B

A solution in $O(k \log n)$ is sufficient:

```
def check(n, xs) -> bool:
    seen = new set implemented as a balanced BST.
    unique = 0

    for each x in xs:
       if x is not in seen:
                Add x to seen
                unique = unique + 1

    return unique == n
```

If one uses an array of booleans instead of a balanced BST the complexity will be $O(k + n)$. The array will have to be initialised which takes $O(n)$ time but "insertions" and lookups will be constant time.

Note that a set implemented with a hash table will not work: its *worst-case* time complexity with regards to add/contains is in $O(n)$.

Section 2: Sorting

Question 2A

42	36	33	33	94	40	41	73	43	15	34
0	1	2	3	4	5	6	7	8	9	10

Median-of-three would in this case give us `median(42, 40, 34) = 40`.

We swap the element at index 0 with the median element (our pivot), at 5:

40	36	33	33	94	42	41	73	43	15	34
0	1	2	3	4	5	6	7	8	9	10

We initialise *lo* = 1 and *hi* = 10 and follow the procedure for moving *lo* to the right and *hi* to the left, swapping elements that are in the wrong partition. Eventually, *lo* and *hi* cross at *lo* = 6 and *hi* = 5 and the array is in the following state:

40	36	33	33	34	15	41	73	43	42	94
0	1	2	3	4	5	6	7	8	9	10

Finally, we move the pivot into its proper place by exchanging it with the element at *hi*:

15	36	33	33	34	40	41	73	43	42	94
0	1	2	3	4	5	6	7	8	9	10
Left					Pivot	Right				

Question 2B

Regardless of strategy for choosing the pivot, there are always inputs that lead to a maximally uneven partitioning, yielding an empty left or right side. We only need to show that there is one such partitioning strategy and that there is a category of inputs for which it always will have a complexity greater than $O(n \log n)$.
Example: Assume a pivot strategy where the first element always is chosen and an array that is already sorted in ascending order:

21	22	23	25	28	213	221	...			

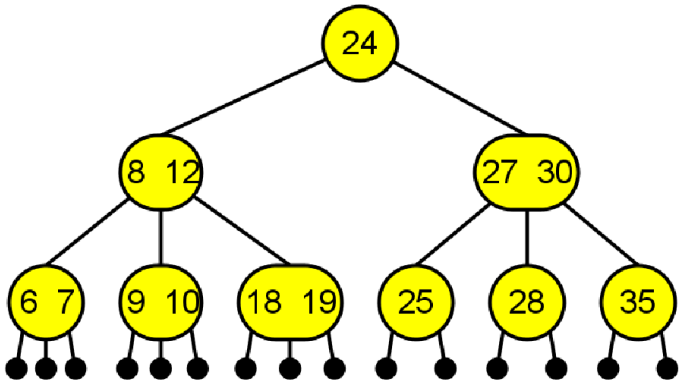
After the first partitioning, we will have sorted a single element (21), leaving us with $n - 1$ remaining elements to be sorted in the next partitioning step. This will result in n partitionings, each doing one less comparison than the previous one for a sum of $(n - 1) + (n - 2) + \ldots + 1 = n\,(n - 1) / 2 \notin O(n \log n)$ comparisons.

Section 3: Search trees

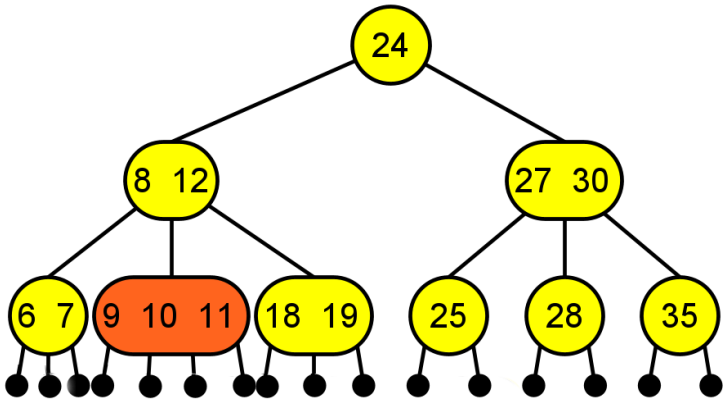
In this section, the key ordering of search trees is the natural ordering of integers.

Question 3A

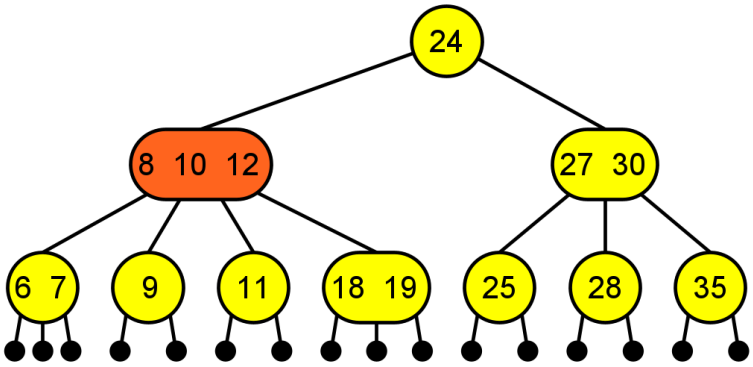
a) When inserting 9, the 2-node "10" becomes a 3-node, "9 | 10".



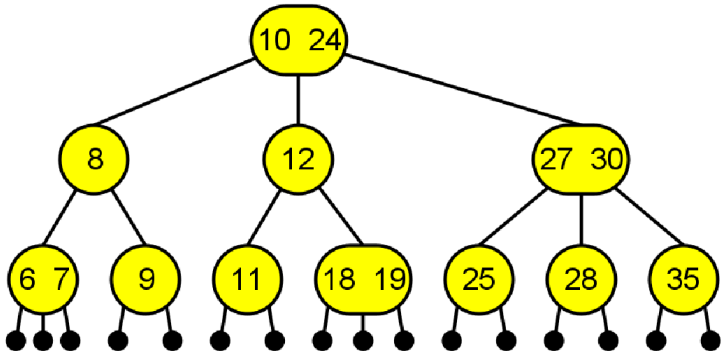
b) Inserting "11" creates a 4-node, "9 | 10 | 11".



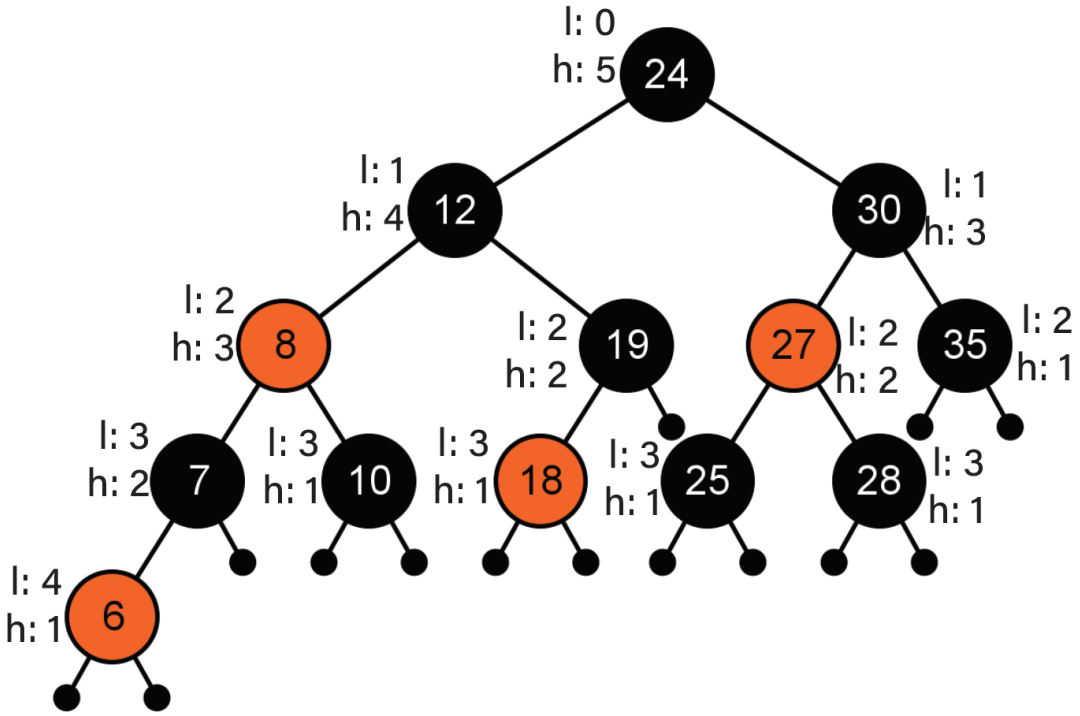
We split and absorb "10" into "8 | 12", turning it into yet another 4-node, "8 | 10 | 12".



We split again and absorb “10” into “24”, giving the root node “10 | 24” and this final tree:



Question 3B



Section 4: Priority queues, binary heaps

Question 4A

Which array out of A, B and C represents a binary min-heap? Only one answer is right.

A

11	38	33	70	76	94	69	97	88	85	52		

B

32	38	40	65	40	92	48	97	55	61	55		

C

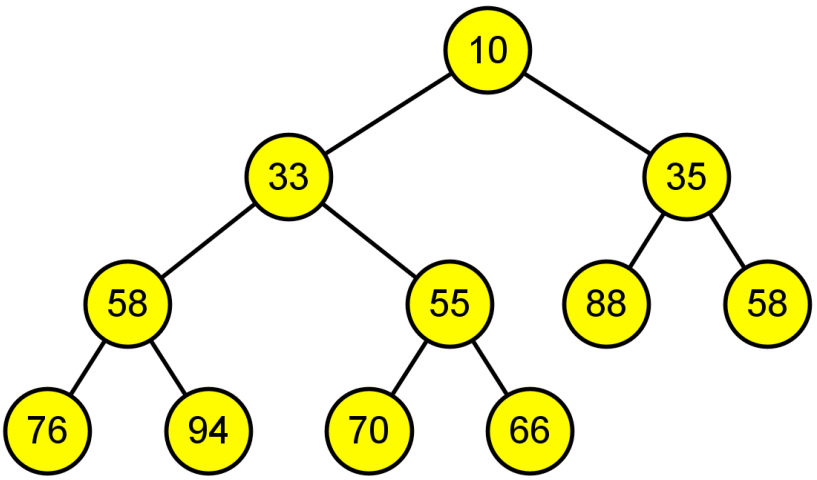
10	33	35	58	55	88	58	76	94	70	66		

For A, 76 is the parent of 52, violating the heap invariant.

For B, 65 is the parent of 55, violating the heap invariant.

For C, the heap invariant holds: every parent is lesser or equal to all its children.

a) Write out that heap as a binary tree.



b) Add 5 to the heap, making sure to restore the heap invariant. How does the array look now?

5	33	10	58	55	35	58	76	94	70	66	88	

Question 4B

In this particular case, removal of 5 gives the same heap again:

10	33	35	58	55	88	58	76	94	70	66		

Section 5: Hash tables

Question 5A

	31	12	21		25	45	55	18	37
0	1	2	3	4	5	6	7	8	9

a) In which order could the elements have been added to the hash table, assuming no deletions occurred? If there is more than one possible insertion order, you should select **all of them**.

(A) 12, 31, 45, 25, 37, 55, 18, 21 -- Impossible, if 45 is inserted before 25 and cell 4 is empty, 45 must be at cell 5.
(B) 25, 18, 31, 45, 55, 12, 21, 37 -- Possible.
(C) 25, 45, 12, 21, 31, 18, 37, 55 -- Impossible, if 21 is inserted before 31 and cell 0 is empty, 21 must be at cell 1.
(D) 18, 25, 12, 45, 31, 21, 55, 37 -- Possible.

b) Add 79 to the table. Assuming no rehashing, what does the table look like now?

Since cell 9 is occupied with 37, 79 will end up in cell 0:

79	31	12	21		25	45	55	18	37
0	1	2	3	4	5	6	7	8	9

Question 5B

Grow (rehash) the hash table from 5A, **after** addition of 79, to array size 13. Insert the elements starting in the order they occur in the original table, i.e. element at 0, then at 1 and so forth.

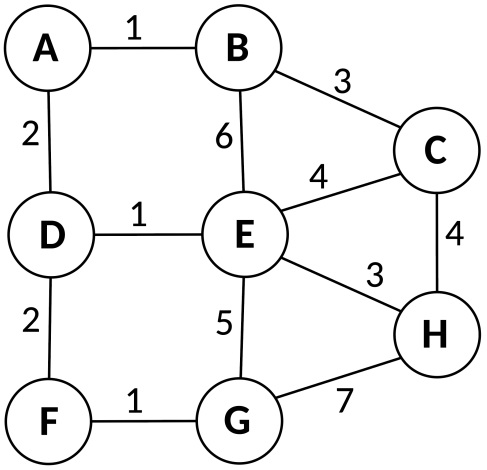
0: 79 mod 13 = 1, cell 1 empty: 79 → cell 1.
1: 31 mod 13 = 5, cell 5 empty: 31 → cell 5.
2: 12 mod 13 = 12, cell 12 empty: 12 → cell 12
3: 21 mod 13 = 8, cell 8 empty: 21 → cell 8
5: 25 mod 13 = 12, cell 12 full, cell 0 empty: 25 → cell 0
6: 45 mod 13 = 6, cell 6 empty: 45 → cell 6
7: 55 mod 13 = 3, cell 3 empty: 55 → cell 3
8: 18 mod 13 = 5, cell 5 full, cell 6 full, cell 7 empty: 18 → cell 7
9: 37 mod 13 = 11, cell 11 empty: 37 → cell 11

Resulting hash table:

25	79		55		31	45	18	21			37	12
0	1	2	3	4	5	6	7	8	9	10	11	12

Section 6: Graphs

You are given the following undirected weighted graph:

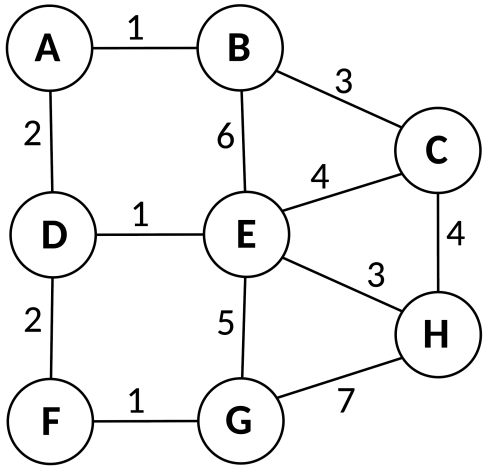


Question 6A:

– Compute a minimal spanning tree for the graph by manually performing Prim's algorithm using H as the starting node.

We ignore edges that go to nodes that are marked as visited. Edges on the queue that have a destination in one of the visited nodes are removed for the priority queue.

Visited	Edge → MST	PQ
H		HE: 3, HC: 4, HG: 7
E	HE	ED: 1, EC: 4, HC: 4, EG: 5, EB: 6, HG: 7
D	ED	DA: 2, DF: 2, EC: 4, HC: 4, EG: 5, EB: 6, HG: 7
A (or F)	DA	AB: 1, DF: 2, EC: 4, HC: 4, EG: 5, EB: 6, HG: 7
B	AB	DF: 2, BC: 3, EC: 4, HC: 4, EG: 5, EB: 6, HG: 7
F	DF	FG:1, BC: 3, EC: 4, HC: 4, EG: 5, EB: 6, HG: 7
G	FG	BC: 3, EC: 4, HC: 4, EG: 5, EB: 6, HG: 7
C	BC	EC: 4, HC: 4, EG: 5, EB: 6, HG: 7
		Early termination – all nodes visited.



Question 6B:

– Perform Dijkstra's algorithm/UCS starting from node A.

In which order does the algorithm visit the nodes and what is the computed distance to each of them?

Explain your answer by showing the data structure(s) involved, making it possible to trace your steps.

If there is a way to assign integer values to nodes, an array of booleans can be used to keep track of what nodes have been visited. In the first column below, "Visited" should be interpreted as the union of the node at the current row and the previous rows.

Visited	Distance	Edge → SPT	PQ
A	0		AB: 1, AD: 2
B	1	AB	AD: 2, BC: 4, BE: 7
D	2	AD	DE: 3, DF: 4, BC: 4, BE: 7
E	3	DE	DF: 4, BC: 4, EH: 6, EC: 7, BE: 7, EG: 8
F	4	DF	BC: 4, FG: 5, EH: 6, EC: 7, BE: 7, EG: 8
C	4	BC	FG: 5, EH: 6, EC: 7, BE: 7, EG: 8, CH: 8
G	5	FG	EH: 6, EC: 7, BE: 7, EG: 8, CH: 8, GH: 12
H	6	EH	EC: 7, BE: 7, EG: 8, CH: 8, GE: 10, GH: 12
			Early termination – all nodes visited.