Algorithms and Data Structures

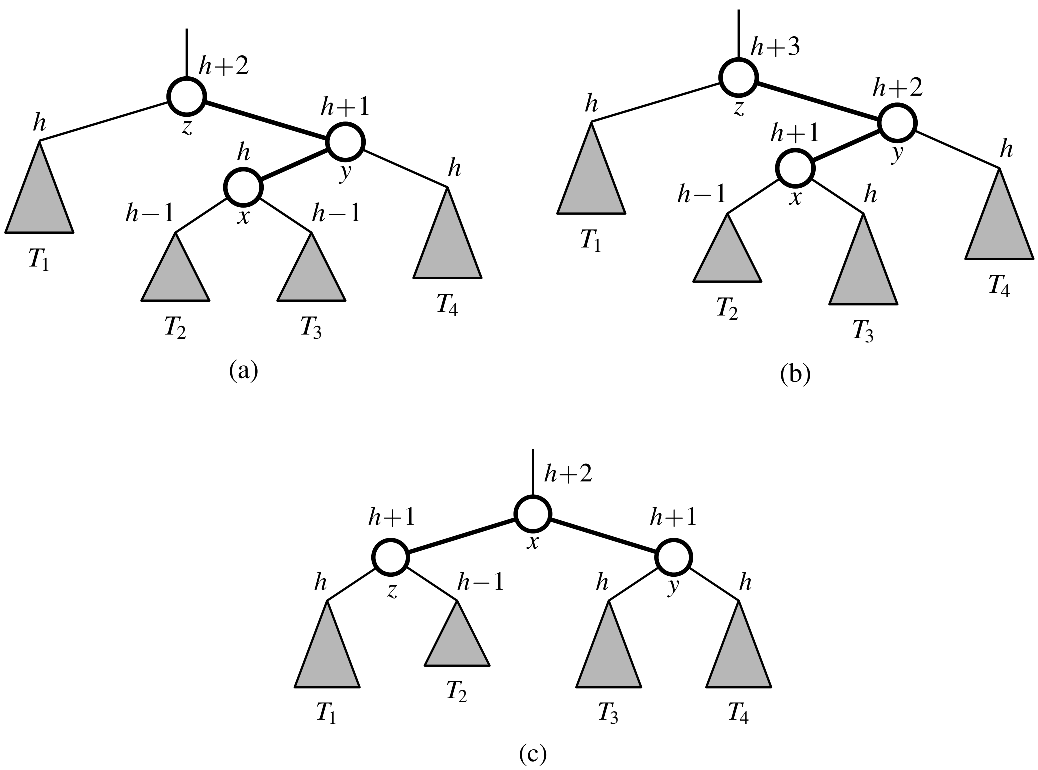
Makeup Exam

October 9, 2024

Last Name: ______________________________

First Name: ______________________________

Signature: ______________________________



(a) before insertion, (b) after insertion, (c) rebalanced

1. (a) Order the following functions by their asymptotic growth rate:

$$n, n + n^2 + n^3, 2^{(n^2)}, (2^n)^2, 2^{2\log n}, 2^{4\log n}$$

Note: $\log n$ *denotes the base-2 logarithm!*

(b) Al and Bob are arguing about their algorithms. Al claims his $O(n \log n)$-time method is always faster than Bob's $O(n^2)$-time method. To settle the issue, they perform a set of experiments. To Al's dismay, they find that if $n < 100$, the $O(n^2)$-time algorithm runs faster, and only when $n \geq 100$ is the $O(n \log n)$-time one better. Explain how this is possible.

(5+5)

(a) Note that: $(2^n)^2 = 2^{2n}$, $2^{2\log n} = n^2$, $2^{4\log n} = n^4$

So the order, from slowest to fastest asymptotic growth rate, is

$$n, \quad 2^{2\log n}, \quad n + n^2 + n^3, \quad 2^{4\log n}, \quad (2^n)^2, \quad 2^{(n^2)}$$

(b) Big-Oh notation does not say anything about the constants multiplying the fastest-growing terms, nor does it say anything about lower order terms (such as fixed set-up costs of an algorithm).

So we only know that there exists an n_0 such that for all $n \geq n_0$, the $O(n \log n)$-algorithm will be faster than an $\Omega(n^2)$ algorithm. In this case, $n_0 = 100$ (assuming that the informal statement "$O(n^2)$-algorithm" actually means that the algorithm is $\Omega(n^2)$ at least for some classes of data).

2. Consider the following algorithm, which takes as input a list L containing integer entries:

```python
def f(L):
    n = len(L)
    for i in range(n):
        for j in range (i):
            for k in range(n):
                if abs(L[i]-L[j]) == L[k]:
                    return True
    return False
```

(a) Describe what this algorithm does and state its asymptotic running time in the best case and in the worst case.

(b) Describe an alternative algorithm for the same problem that uses hashing and completes in $O(n^2)$ time. You may assume that inserting a key into the hash table or searching for a key always completes in $O(1)$ time

(5+5)

(a) The algorithm returns true if and only if the absolute value of the difference of two numbers in the array matches a number in the array.

In the best case, it finds a match immediately (e.g. if all entries are zero), thus completing in $O(1)$ time. In the worst case, when no match is found, the three nested loops are executed fully at $O(n^3)$ time.

(b) Put all members of L as keys into a hash table at $O(n)$ cost. Then replace the inner loop by a check whether abs(L[i]-L[j]) is a key in the hash table. The two outer loops then run at $O(n^2)$ as the check is assumed to incur $O(1)$ cost.

3. Are the following statements true or false? Explain your answer in 1–2 sentences.

   (a) There exists a sort algorithm that can sort a list of length n in $O(n)$ best case.

   (b) There exists a sort algorithm that can sort a list of length n in $O(n)$ worst case.

   (c) Searching for an element in a splay tree with n elements always completes in $O(\log n)$ time.

   (d) Searching for the same element in a splay tree with n elements n times in direct succession takes $O(n)$ time.

   (e) When implementing Dijkstra's algorithm to find the shortest path between vertices in a connected graph, we should always use a heap-based adaptable priority queue to get optimal performance.

(2+2+2+2+2)

(a) True. E.g., insertion sort will take $O(n)$-time on an already sorted list.

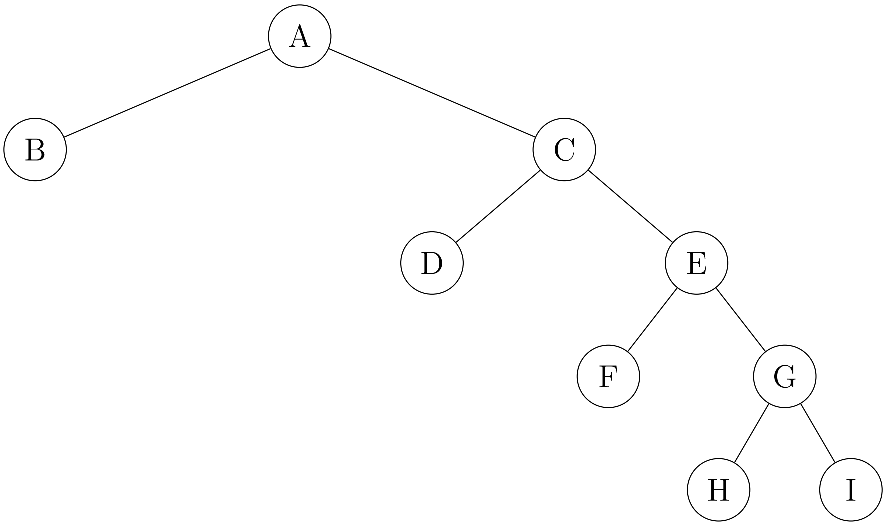
(b) False. It is a theorem, proved in class, that any comparison-based sorting algorithm has a worst-case performance of $\Omega(n \log n)$. (Proof not required – it is based on estimating the height of the associated decision tree, see, e.g., GTG p. 563.)

(c) A splay tree can be maximally unbalanced (i.e., degenerate to a linked list), e.g. when constructing the tree from sorted data. Thus, the statement is false as a search in such a tree takes $O(n)$ worst case.

(d) True. Searching once takes $O(n)$ worst case, see (c). The element matching the search key, if found, or the leaf where the search terminates unsuccessfully is splayed to root. Thus, the $n-1$ next searches take $O(1)$-time each, so $O(n)$ altogether.

(e) False. An unsorted sequence implementation updates in $O(1)$-time as opposed to $O(\log n)$ for the heap. Thus, for near-complete graphs, so $O(n^2)$ updates, this is better than using a heap.

4. Give the pre-order, in-order, and post-order traversals of the following binary tree:



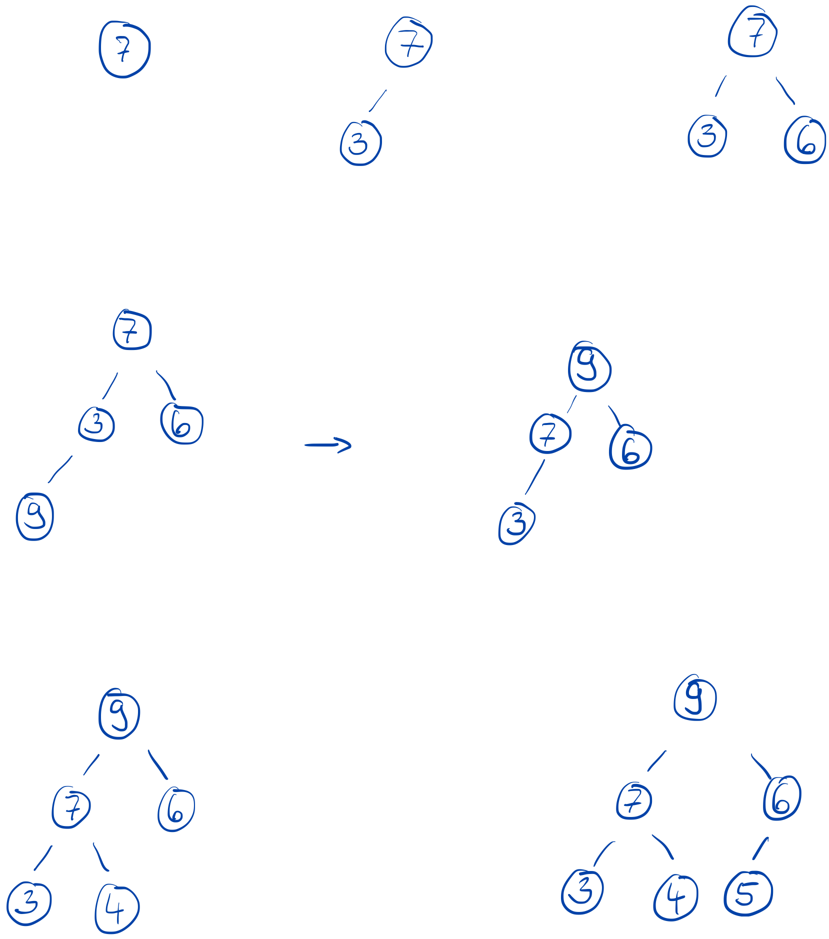
(5)

pre-order: A, B, C, D, E, F, G, H, I

in-order: B, A, D, C, F, E, H, G, I

post-order: B, D, F, H, I, G, E, C, A

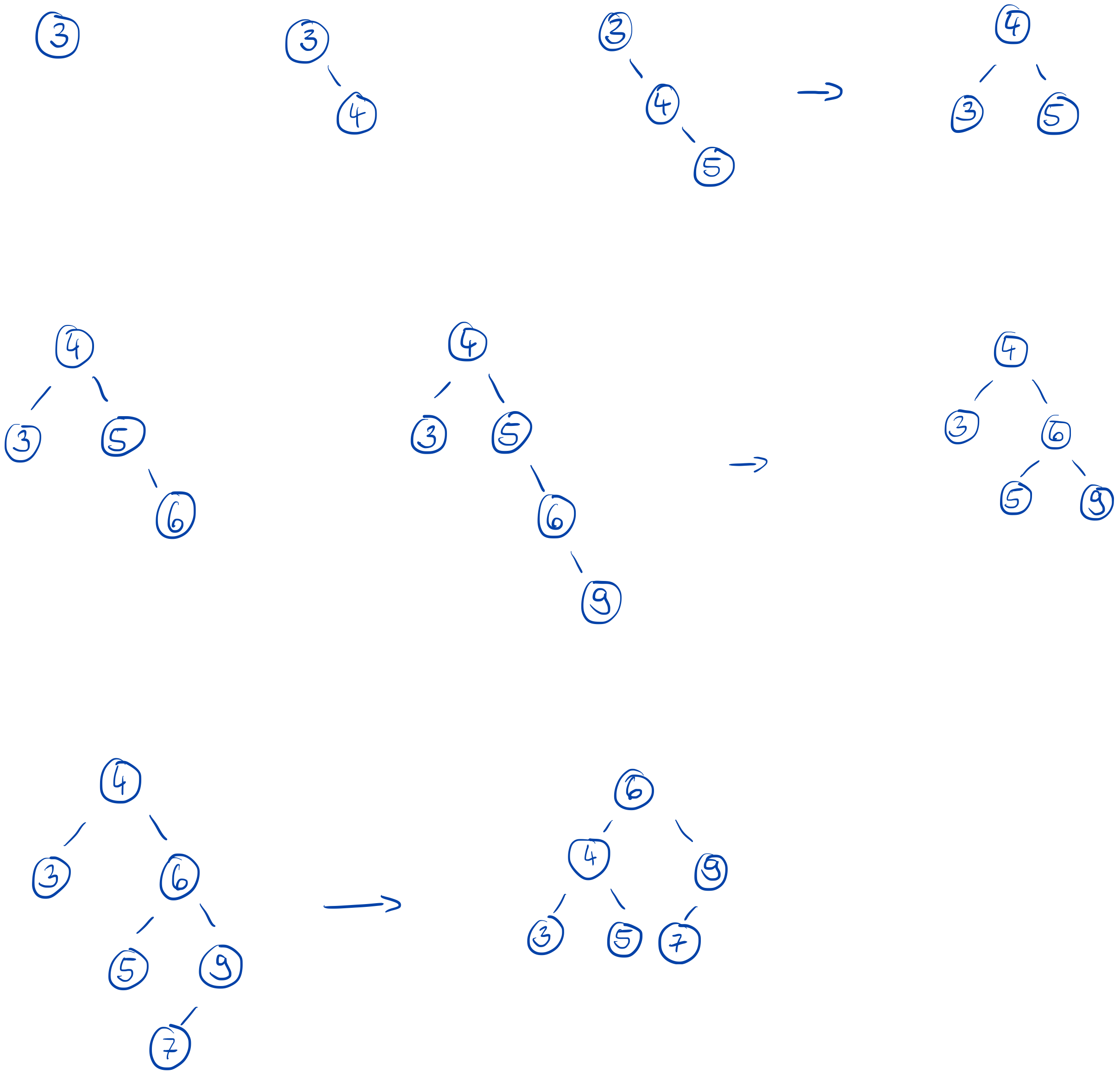
5. Insert the keys 7, 3, 6, 9, 4, 5 into an initially empty max-heap. Show the max-heap at each step of insertion. (5)



(restore heap order)

6. Insert the keys 3, 4, 5, 6, 9, 7 into an initially empty AVL tree tree. Show the AVL tree at each step of insertion.

   (You may consult the attached AVL tree "cheat sheet".) (5)



7. Give a precise and complete description of Kruskal's algorithm for computing the minimum spanning tree of a graph. (5)

Initialization:

Start with empty MST

Every vertex of the graph is a separate cluster.

Put all edges into a min-heap.

while heap is not empty:

remove minimum edge e from heap

if e connects two clusters:

put the edge into the MST

merge the two clusters

Note: loop can be terminated when MST has $|V|-1$ edges. This avoids unnecessary checking of additional edges.

(Solution ctd. or scratch.)