Data Structures and Algorithms — Block 1, 2018

Final Exam - Answer sheet

1 Problem 1 (35pt)

In each of the following question, please specify if the statement is true or false. If the statement is true, explain why it is true. If it is false, explain what the correct answer is and why. (For each question, 2 points for the true/false answer and 3 points for the explanations.)

(a) (5pt) If $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, then we have $f(n) = g(n)$.

Answer: *False.* $f(n) = \Theta(g(n))$.

(b) (5pt) If $f_1(n) = \Omega(g_1(n))$ and $f_2(n) = \Omega(g_2(n))$, then $f_1(n) \times f_2(n) = \Omega(g_1(n) \times g_2(n))$

Answer: *True.*
$f_1(n) = \Omega(g_1(n)) \implies$ *There exists positive constants* n_1 *and* c_1 *such that* $f_1(n) \geq c_1 g_1(n)$ *for all* $n \geq n_1$.
$f_2(n) = \Omega(g_2(n)) \implies$ *There exists positive constants* n_2 *and* c_2 *such that* $f_2(n) \geq c_2 g_2(n)$ *for all* $n \geq n_2$.
So we can find positive constants $c = c_1 c_2$ *and* $n_0 = max\{n_1, n_2\}$ *such that* $f_1(n) \times f_2(n) \geq c \times g_1(n) \times g_2(n)$ *for all* $n \geq n_0$.
Then $f_1(n) \times f_2(n) = \Omega(g_1(n) \times g_2(n))$.

(c) (5pt) $2^n + n^2 = O(2^n)$

Answer: *True. Need to show that* $n^2 = O(2^n)$.
$n^2 = O(2^n) \implies$ *There exists constants* c, n_0 *s.t.* $n^2 \leq c2^n$ *for all* $n \geq n_0$.
$n^2 \leq c2^n \implies c \geq \frac{n^2}{2^n}$. *Pick* $c = 1$ *and* $n_0 = 4$.

(d) (5pt) A stack follows a FIFO (first-in-first-out) rule.

Answer: *False. A stack follows a FILO (first-in-last-out) rule.*

(e) (5pt) When we use a max heap to implement a priority queue, the time complexity of both insert and delete operations are $O(n)$.

Answer: *True. The operations* `insert()` *and* `delete()` *spend* $O(h)$, *where* $h =$*height of max heap. Because the max heap is a complete tree we have* $h \leq \lg n \implies$ *The time complexity of* `insert()` *and* `delete()` $= O(\lg n) = O(n)$.

(f) (5pt) $T(n) = T(n-1) + n$, $T(1) = 1$. Then $T(n) = O(n^3)$.

Answer: *True.*

$$\begin{array}{ll} T(n) & = T(n-1) + n \\ T(n-1) & = T(n-2) + n - 1 \\ \ldots & \\ T(2) & = T(1) + 2 \\ + & \\ \hline T(n) & = 1 + 2 + ... + n = n(n+1)/2 = O(n^2) = O(n^3) \end{array}$$

(g) (5pt) In a circular doubly linked list with 10 nodes, we will need to change 4 links if we want to delete a node other than the head node.

Answer: *False. We only need to change the* `next` *attribute of the node that precedes the node we want to delete, and the* `pre` *attribute of the node that follows the node we want to delete.*

Problem 2 (6pt)

Consider insertion sort and merge sort. For each algorithm, what will be the worst case asymptotic upper bound on the running time if you know additionally the following about the input (no explanation is needed, just give the running time):

(a) the input is already sorted?
Answer: *Insertion sort* $O(n)$*, merge sort* $O(n \lg n)$.

(b) the input is reversely sorted?
Answer: *Insertion sort* $O(n^2)$*, merge sort* $O(n \lg n)$.

(c) the input is a list containing n copies of the same number?
Answer: *Insertion sort* $O(n)$*, merge sort* $O(n \lg n)$.

Problem 3 (8pt)

Suppose that we have numbers between 1 and 100 in a binary search tree and want to search for the number 45. Which (possibly multiple) of the following sequences could be the sequence of nodes examined?

A. 5, 2, 1, 10, 39, 34, 77, 63.

B. 1, 2, 3, 4, 5, 6, 7, 8.

C. 9, 8, 63, 0, 4, 3, 2, 1.

D. 8, 7, 6, 5, 4, 3, 2, 1.

E. 50, 25, 26, 27, 40, 44, 42.

F. 50, 25, 26, 27, 40, 44.

Explain, in one sentence each, why the remaining sequences cannot be the sequence of nodes examined.

Answer: *B, F.*
In (A), the left child (2) is checked after root (5). Should have checked right child, which is ≥ 5.
In (C), the left child (8) is checked after root (9). Should have checked right child, which is ≥ 9.
In (D), the left child (7) is checked after root (8). Should have checked right child, which is ≥ 8.
In (E), the left child (42) is checked after (44). Should have checked right child, which is ≥ 44.

Problem 4 (5pt)

Suppose that we first insert an element x into a binary search tree that does not already contain x. Suppose that we then immediately delete x from the tree. Will the new tree be identical to the original one? If yes give the reason in a few sentences. If no, give a counterexample. Draw pictures if necessary. Can you give the name of a data structure where this is not the case?

Answer: *Yes. When you insert a node into a binary search tree, that node becomes a leaf. No other operations are applied to the tree. So nothing changes when we delete the node. Example: Red-black trees*

Problem 5 (8pt)

Fill in the table below with the worst-case asymptotic running time of each operation when using the data structure listed. Assume that L is a list, x is a pointer to an element, and k is the key of an element.

Answer:

	INSERT(L,x)	SEARCH(L,k)	DELETE(L,x)	DELETE(L,k)
Singly-linked unordered list	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Doubly-linked unordered list	$O(1)$	$O(n)$	$O(1)$	$O(n)$

Problem 6 (10pt)

Complete this Python code so that it properly implements insertion sort. Make sure the result is in-place (do not use another array).

Answer:

```
def insertionSort(items):
    for j in range(1, len(items)):
        element = items[j]
        i = j
        while 0 < i and element < items[i-1]:
            items[i] = items[i - 1]
            i -= 1
        items[i] = element
```

Problem 7 (13pt)

Suppose you have the following hash table, implemented using linear probing. The hash function we are using is the identity function, $h(x) = x \mod 9$.

0	1	2	3	4	5	6	7	8
9	18		12	3	14	4	21	

(a) (8pt) In which order could the elements have been added to the hash table? There are several correct answers, and you should give all of them. Assume that no elements have been deleted from the table.

A. 9, 14, 4, 18, 12, 3, 21

B. 12, 3, 14, 18, 4, 9, 21

C. 12, 14, 3, 9, 4, 18, 21

D. 9, 12, 14, 3, 4, 21, 18

E. 12, 9, 18, 3, 14, 21, 4

Explain, in one sentence each, why the remaining sequences cannot be the sequence of nodes examined.

Answer: *C and D. In A, 4 would be inserted at index 4 instead of 6. In B, 18 would be inserted at index 0 instead of 1. In E, 21 would be inserted at index 6 instead of 7.*

(b) (2pt) Delete the element with key 3 from the hash table, and write down how it looks afterwards.
Answer:

0	1	2	3	4	5	6	7	8
9	18		12	DELETED	14	4	21	

(c) (3pt) If we want a hash table that stores a set of strings, one possible hash function is the string length, $h(x) = x.length \mod m$, where m is the size of the hash table. Is this a good hash function? Explain your answer.

Answer: *No. Strings with the same length will have the same hash value. If we insert lots of strings with the same length, search will take $O(n)$ time instead of $O(1)$.*

Problem 8 (15pt)

The Fibonacci numbers are given by the recurrence:

$F_0 = 0$

$F_1 = 1$

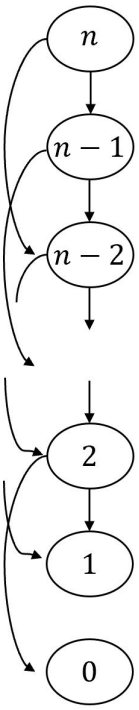
$F_i = F_{i-1} + F_{i-2}$

yielding the sequence $0, 1, 1, 2, 3, 5, 8, \ldots$

(a) (10pt) Write an $O(n)$-time dynamic programming algorithm (in pseudocode) to compute the n'th Fibonacci number.

Answer:

```
FIBONACCI(n)
    let fib[0..n] be a new array
    fib[0]=0
    fib[1]=1
    for i=2 to n
        fib[i]=fib[i-1]+fib[i-2]
    return fib[n]
```



(b) (5pt) Draw the subproblem graph. How many vertices and edges are in the graph?

Answer:

$n + 1$ vertices.
2 edges leaving vertices from $2, 3, \ldots, n$. No edges leaving 0 and 1. $\implies 2n - 2$ edges.