Student number:	2467273
Course title:	COMPSCI2007 Algorithms and Data Structures 2
Questions answered:	ALL

1.

a) The algorithm reverses the order of an array A from indices p to r, assuming p is smaller than r. It checks this assumption, saves value at index p in a temporary variable, then swaps elements at indices p and r. Finally, it calls itself recursively by increasing p by 1 and decreasing r by 1.

b) [1,4,3,9,7,5,1]

c) F(A,0,6) → F(A,1,5) → F(A,2,4) → F(A,3,3). There is no return value as it modifies the given array. The modified array, cascading from the bottom, is the one given in part b).

d) F is tail recursive as the very last operation performed in the method is a call to itself – F(A,p+1,r-1).

e) F is an in-place algorithm as it requires additional space only to store the temporary variable x, which is a constant requirement. Otherwise, it modifies the given array.

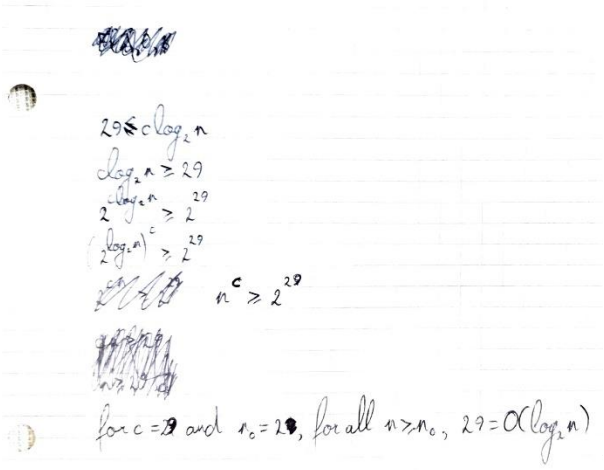
f)

```
F(A, p, r)
    while (p < r):
        x := A[p]
        A[p] := A[r]
        A[r] := x
        p = p+1
        r = r-1
```

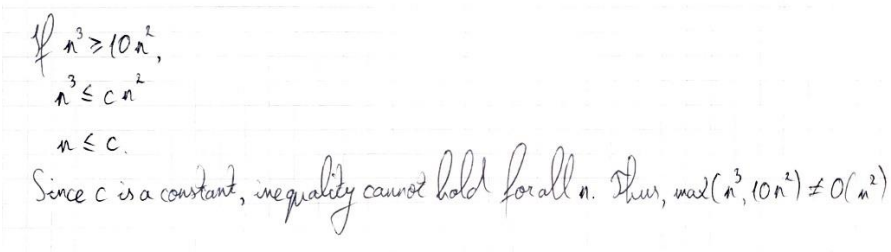
If p = 0 and r = n-1 (spanning all of the array), the worst-case complexity is O(n/2) because the indices meet in the middle of the array and each of the two index variables passes each index once. Simplified, the complexity is O(n).

2.

a)

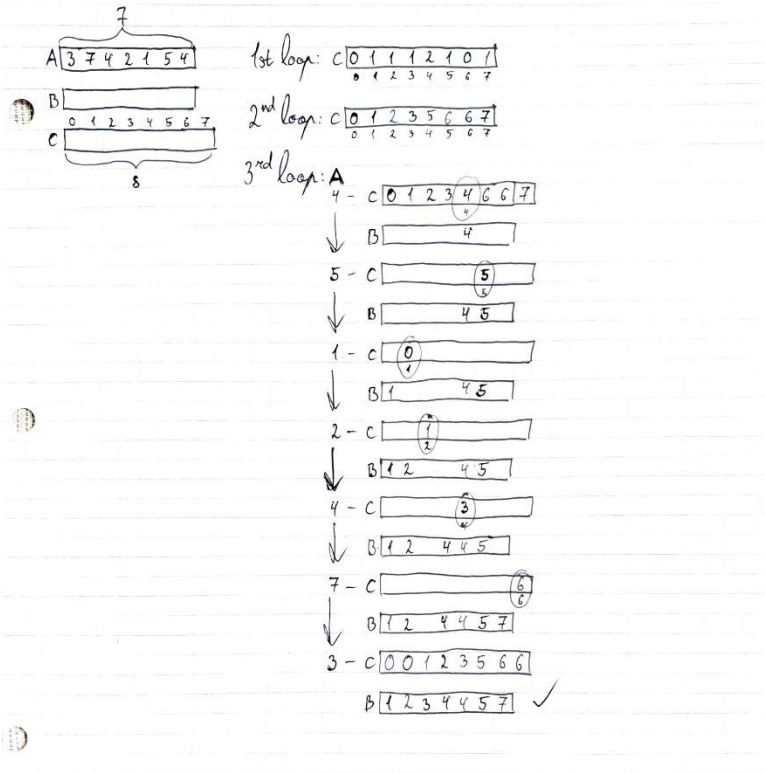


b)

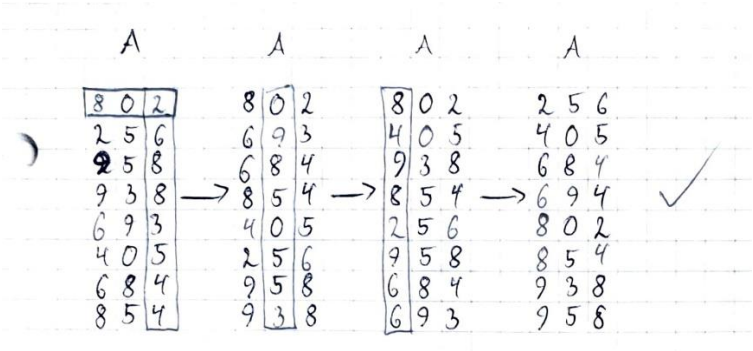


3.

a) Counting sort is given two arrays, A and B, both of size n, and an integer k, where A has elements from 0 to k. First, an empty array C is created of size k. The first of 3 loops populates C by counting occurrences of elements in A in the corresponding index of C (C's index == A's value). The second loop modifies C into a running sum. The third loop goes backwards through A, finds the corresponding index in C for an element in A, and the index's value is decreased by 1 to find the index of the element in B, the final sorted output array.



b) Radix sort is given an array A and an integer d (digits of elements in A). It sorts elements with counting sort digit-by-digit from least to most significant digit in a stable manner (does not swap elements if equal).



4.

a)

PUSH(S, 5): S.top = 0, S[0] = 5

PUSH(S, 2): S.top = 1, S[1] = 2

PUSH(S,4): S.top = 2, S[2] = 4

POP(S): S.top = 1

PUSH(S,7): S.top = 2, S[2] = 7

POP(S): S.top = 1

Resulting array: [5,2,7,0,0] (S[2] is a legacy element)

S.top = 1

b)

PUSH(S, 5): S.top points to node with 5

PUSH(S, 2): S.top points to node with 2, which points to 5

PUSH(S, 4): S.top points to node with 4, which points to 2, which points to 5

POP(S): S.top points to node with 2, which points to 5

PUSH(S, 7): S.top points to node with 7, which points to 2, which points to 5

POP(S): S.top points to node with 2, which points to 5

Resulting linked list: S.top points to 2, which points to 5.

5.

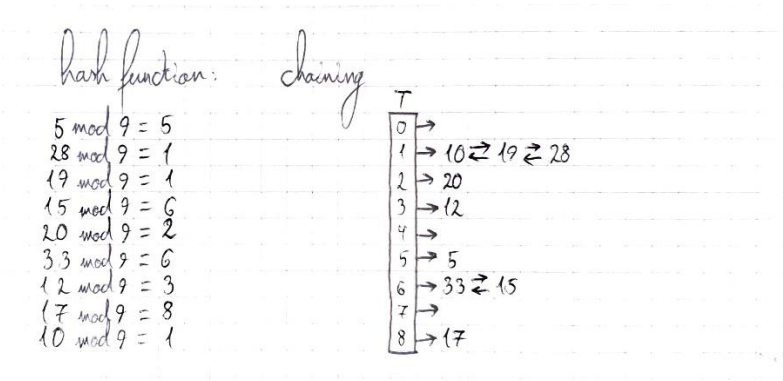
a) A hash table is a data structure that consists of an array T of fixed size (the table itself) and a hash function which maps keys to indices in the array (thus, the target set is the same size as the array). All operations (INSERT, DELETE, SEARCH) take O(1) time on average, but act differently depending on what technique is used to resolve hash collisions. In INSERT, the key is assigned to the index of T that

is returned from the hash function with the key as an input. DELETE and SEARCH are slower when their given key had been part of a hash collision.

b) A hash collision is the phenomenon when two separate keys are assigned to the same index in the hash table because the hash function returned the same value. An example for a hash table could be the database for a board game like Scrabble to associate letters with their value (points) because sorting does not matter and lookup is extremely fast.

c) Hash collision resolution by chaining implements a linked list in every table index that experiences a hash collision. New elements in the same index are added to the linked list and deleted elements removed. In this case, deletion can be in constant time (O(1)) if the input is the entire node.

d)



e) Hash collision resolution by open addressing, every time a hash collision occurs, searches next available indices in the table and inserts the key into an available one if it finds one. This searching (probing) can be linear, quadratic, or done by double hashing, which implements two different hashing functions to find an available slot. For the entire table to be searched, linear probing or double hashing with the second hash function being relatively prime to the size of the table have to be used.

f)