# Algorithms and Data Structures

Mock Exam

July 12, 2022

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

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

   (b) An algorithm executes an $O(\log n)$-time computation for each entry of an $n$-element sequence. Give a Big-Oh upper bound and a Big-Omega lower bound on its running time.

   (c) Give the best possible Big-Oh upper bound for the running time of the following Python function which takes as input two Python lists `A` and `B` with respective lengths $n$ and $m$.

```
def mystery_function (A,B):
    i = 0
    j = 0
    while i<len(A) and j<len(B):
        if A[i]==B[j]:
            return True
        elif A[i]<B[j]:
            i += 1
        else:
            j += 1
    return False
```

   (d) What is `mystery_function` good for? State, if necessary, conditions on the input arrays `A` and `B` that make `mystery_function` perform useful work.

(5+5+5+5)

2. Is each of the following statements true or false? Explain your answer in 1–2 sentences.

   (a) One can implement a *stack* based on a *linked list* such that each push or pop operation completes in $O(1)$-time.

   (b) One can implement a *stack* based on a *dynamic array* such that each push or pop operation completes in $O(1)$-time.

   (c) It is possible to append a *linked list* to another in $O(1)$-time.

   (d) *Heap-sort* is always faster then *insertion-sort*.

(e) *Heap-sort* is faster than *insertion-sort* when the input is a list containing $n$ copies of the same number.

(2+2+2+2+2)

3. (a) The nodes of a complete binary tree have keys that represent their position in a breadth-first traversal of the tree. Argue that this tree is a heap.

   (b) Give a pseudo-code (or Python) representation of the breadth-first traversal of a tree with an auxiliary queue.

   (c) Argue that the run-time of this algorithm is $O(n)$, where $n$ is the number of nodes in the tree.

   (d) Alternatively, you can process the nodes of this tree breadth-first by repeatedly calling the `remove_min` method of the heap. Does this algorithm also run in $O(n)$ time? Explain!

(5+5+5+5)

4. Attached is an excerpt of a code listing for a buggy implementation of a priority queue with a binary heap.

   (a) Draw an example of a heap with exactly 5 nodes so that a call to `remove_min` produces an invalid heap.

   (b) Identify the part of the code that is buggy. Explain!
   *Hint:* The functions `_parent` to `_has_right` which implement the index arithmetic are correct, you do not need to look there.

   (c) Fix the bug.

   (d) Rewrite the function `_upheap` to use a loop instead of recursion.

(5+5+5+5)

```
1  class HeapPriorityQueue(PriorityQueueBase):
2
3    def _parent(self, j):
4      return (j-1) // 2
5
6    def _left(self, j):
7      return 2*j + 1
8
9    def _right(self, j):
10     return 2*j + 2
11
12   def _has_left(self, j):
13     return self._left(j) < len(self._data)
14
15   def _has_right(self, j):
16     return self._right(j) < len(self._data)
17
18   def _swap(self, i, j):
19     """Swap the elements at indices i and j of array."""
20     self._data[i], self._data[j] = self._data[j], self._data[i]
```

```
21
22    def _upheap(self, j):
23      parent = self._parent(j)
24      if j > 0 and self._data[j] < self._data[parent]:
25        self._swap(j, parent)
26        self._upheap(parent)
27
28    def _downheap(self, j):
29      if self._has_left(j):
30        left = self._left(j)
31        small_child = left
32        if self._has_right(j):
33          right = self._right(j)
34          if self._data[right] < self._data[left]:
35            small_child = right
36        self._swap(j, small_child)
37        self._downheap(small_child)
38
39    def __init__(self):
40      """Create a new empty Priority Queue."""
41      self._data = []
42
43    def __len__(self):
44      """Return the number of items in the priority queue."""
45      return len(self._data)
46
47    def add(self, key, value):
48      """Add a key-value pair to the priority queue."""
49      self._data.append(self._Item(key, value))
50      self._upheap(len(self._data) - 1)
51
52    def min(self):
53      """Return but do not remove (k,v) tuple with minimum key.
54
55      Raise Empty exception if empty.
56      """
57      if self.is_empty():
58        raise Empty('Priority queue is empty.')
59      item = self._data[0]
60      return (item._key, item._value)
61
62    def remove_min(self):
63      """Remove and return (k,v) tuple with minimum key.
64
65      Raise Empty exception if empty.
66      """
67      if self.is_empty():
68        raise Empty('Priority queue is empty.')
69      self._swap(0, len(self._data) - 1)
70      item = self._data.pop()
71      self._downheap(0)
72      return (item._key, item._value)
```

5. Write an algorithm `min_list`, in pseudo-code or in Python, that returns a list with the values of all nodes of a heap whose key is identical to the minimal key. (10)