

Problem 1. Asymptotic Growth

Question 1.1

Remember the PQ-SORT algorithm (Goodrich&Tamassia p. 96):

PQ-SORT(C)

▷ C is an n element sequence of numbers

1 Let P be an empty new priority queue

2 **while** C is not empty

3 **do** $e \leftarrow C.REMOVEFIRST()$

4 $P.INSERTITEM(e)$

5 **while** P is not empty

6 **do** $e \leftarrow P.REMOVEMIN()$

7 $C.INSERTLAST(e)$

8 **return** C

We consider a version of this algorithm, which uses a balanced binary search tree as a priority queue. We are sorting an input containing n elements.

1.1.1 What is the cost of filling the queue with all the n elements (lines 2–4)?

Answer We do n insertions into a balanced BST, $O(\log n)$ each, achieving the total running time of $O(n \log n)$ for constructing the queue.

1.1.2 What is the cost of a single REMOVEMIN call with this queue? (line 6)

Answer Finding the minimum element is done by traversing the left edges as long as possible. The traversal takes $O(\log n)$ time, removal takes $O(\log n)$ time using the regular procedure, for example with AVL trees. So in total REMOVE-MIN is $O(\log n)$

1.1.3 What is the total asymptotic running time of PQ-SORT with this queue?

Answer The algorithm does the construction of the priority queue, followed by n times REMOVE-MIN, which gives: $O(n \log n) + n \cdot O(\log n) = O(n \log n)$.

1.1.4 Assume availability of a Java class `PriorityQueue` implementing a priority queue for storing integer numbers. It offers following methods:

```
boolean PriorityQueue.isEmpty()
boolean int PriorityQueue.removeMin()
boolean int PriorityQueue.insert(int)
```

The constructor of `PriorityQueue` takes no parameters.

Write a Java method `int[] PQSort(int[] C)`, which implements the algorithm presented above (PQ-SORT), using the available implementation of `PriorityQueue`.

Note: you should not implement the `PriorityQueue` class.

Answer

```
int[] PQSort(int[] C) {
    PriorityQueue P = new PriorityQueue();
    for(x :C) P.insert (x);

    while (!P.isEmpty()) {
        int e = P.removeMin();
        int i = 0;
        C[i++] = e;
    }
    return C;
}
```

Question 1.2 Consider a network which has a topology of a complete binary tree with l leaves (see Figure 1 for an example).

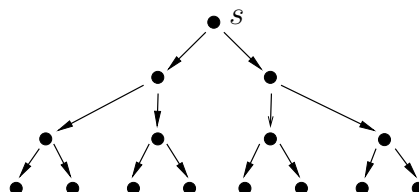


Figure 1: An example of a complete binary network with $l = 8$ leaves.

Assume that the nodes in the network operate synchronously. At each step

a node receives a message from its parent and then sends the same message unaltered to both of its children.

1.2.1 In a network with l leaves, how many synchronous rounds are needed before a message initiated by the root node s reaches the leaf nodes? Give an exact function of l , or the rate of growth. Why?

Answer $O(\log n)$ rounds, since this is the bound on the height of the tree, and the synchronous propagation will descend one level down at each round.

1.2.2 How many messages will be send in total in this network during distribution of a single message starting at the root node? Give a precise function of l or its asymptotic rate of growth. Explain how you arrive at your answer.

Hint: $\sum_{i=0}^k 2^{-i} \leq \sum_{i=0}^{\infty} 2^{-i} = O(1)$

Answer $l + \frac{l}{2} + \frac{l}{4} + \dots + 1 \leq l \sum_{i=0}^{\infty} 2^{-i} = l \cdot O(1) = O(l)$ messages.

Question 1.3 Consider the following program calling a function F .

```

PROCEDURE-1( $n$  :integer)
1  for  $i \leftarrow 1$  to  $n^2$ 
2      do for  $j \leftarrow 1$  to  $n$ 
3          do F( $j$ )
    
```

1.3.1 Assume that $F(j)$ runs in $O(j)$ time. What is the running time of a single call to PROCEDURE-1(n)? Explain how you have obtained your answer.

Answer The inner loop runs in $O(1 + \dots + n) = O(n^2)$ time. The outer loop runs n^2 iterations, so in total: $n^2 \cdot O(n^2) = O(n^4)$.

1.3.2 What should be more economical: calling PROCEDURE-1($3n$) once, or calling PROCEDURE-1(n) three times? Why?

Answer It should be more economical to call PROCEDURE-1(n) three times, since the value $3 \cdot c \cdot n^4$ is smaller than $c(3n)^4$ for any given positive constant c ; more precisely function kcn^4 grows slower in k than ck^4n^4 , for any given $k > 1$.

Problem 2. Ringbuffers

Remember the ring buffer implementation of a queue (Goodrich & Tamassia p. 63):

DEQUEUE()

```
1 if ISEMPY()
2   then throw QueueEmptyException
3    $temp \leftarrow Q[f]$ 
4    $f \leftarrow (f + 1) \bmod N$ 
5   return  $temp$ 
```

ENQUEUE(o)

```
1 if  $size() = N - 1$ 
2   then throw QueueFullException
3    $Q[r] \leftarrow o$ 
4    $r \leftarrow (r + 1) \bmod N$ 
```

Initially $r = f = 0$. In general, the queue is empty whenever $r = f$.

Question 2.1 Remember that for a data structure an invariant is a property that always holds. Mark which of the following sentences are invariant for this data structure, by writing yes or no into the box.

$f < r$

$f \leq N - 1$

$r \leq N - 1$

Let $r' \leftarrow r$, before calling ENQUEUE(o). Then after the call we have that $r = r' + 1$, or an exception has been thrown.

Let $f' \leftarrow f$ before calling DEQUEUE(o). Then after the call we have that $f = (f' + 1) \bmod N$, or an exception has been thrown.

Answer no, yes, yes, no, yes

Question 2.2 One problem with the above implementation is that it can only afford N elements simultaneously in the queue. Use exponential doubling to design a new ENQUEUE algorithm that would address this limitation. Describe your algorithm using pseudocode, or in English, or in Danish.

Answer

ENQUEUE(o)

```

1  if size() = N - 1
2      then Allocate a new queue  $Q'$  with buffer size 2N
3           Remove all elements from  $Q$  using DEQUEUE
4           and insert them into  $Q'$ 
5            $Q \leftarrow Q'$ 
6           ENQUEUE( $o$ )
7      return
8   $Q[r] \leftarrow o$ 
9   $r \leftarrow (r + 1) \bmod N$ 

```

Question 2.3 What is the amortized running time of a *single* ENQUEUE call in a series of n consecutive ENQUEUE operations into an initially empty queue with your modification?

Answer Most of ENQUEUEs happen in constant time. Those that reallocate and double the queue take linear time, but this happens at geometrically decreasing frequency. The analysis is standard as for dynamic hashing or dynamically growing arrays. Effectively the amortized time of ENQUEUE is constant.

Problem 3. Sorting and Dictionaries

Question 3.1

3.1.1 What good general purpose alternative for MERGE-SORT do you know?

Answer QUICK-SORT (HEAP-SORT is also a good answer).

3.1.2 You need to find just one element in a large array. Which is more eco-

nomical: to make a sequential search in the array, or to sort the array first and then use a binary search? Explain why.

Answer If only one search is needed, then it is more economical to do linear search since it takes $O(n)$ time, as opposed to $O(n \log n)$ time required by sorting.

3.1.3 You need to sort a file that does not fit into memory. Suggest a way, using algorithms you know, to sort this file using only $O(n \log n)$ read/write operations.

Remark: A single read (write) operation can only read (write) a constant amount of elements from a file. In this exercise assume that one read (write) is equivalent to accessing one input element, like in a linked list.

Answer I would use a MERGE-SORT like algorithm which would operate on files instead of on sequences.

Question 3.2 Consider two sets of integers stored in two arrays A and B . Neither A nor B can contain duplicates, but they can share some values. We want to compute the difference of these two sets: $A - B$. Answer the following two questions describing algorithms in Danish or in English.

Reminder: $A - B = \{x \mid x \in A \text{ and } x \notin B\}$

3.2.1 How can $A - B$ be computed guaranteeing $O(n \log n)$ worst case time?

Answer Sort both arrays ($O(n \log n)$) then merge them in such way that for every pair of values (a, b) in heads of the sequences: 1) if b is smaller then it is removed (but not placed in the output) 2) if $a = b$ then both are removed and not placed in the output. Total time: $O(n \log n) + O(n) = O(n \log n)$.

3.2.2 How can $A - B$ be computed guaranteeing expected linear running time? Hint: use a suitable dictionary data structure.

Answer Hash all values from A into a hash table. For every value b of B see if it is already in the hash table. If so then remove it from the hash table. Traverse the hashtable to print out $A - B$ (or use the hash table directly to represent it). We perform expected constant time operations for $|A| + |B|$ elements, obtaining expected linear execution time.

Name:

CPR:

page 7 of 9

Question 3.3 Consider a modification of the INSERTION-SORT algorithm, which performs a binary search (instead of a sequential one) to find a position where the next element should be placed.

3.3.1 What is the worst case asymptotic running time of this algorithm?

Answer It is the same as that of insertion sort, $O(n^2)$. Binary search is faster than sequential. The latter takes linear time, but still this linear time is needed to shift the elements in the array.

3.3.2 Is it asymptotically better or worse than the original INSERTION-SORT?

Answer The same. Argument above.

Problem 4. Graphs and Trees

Question 4.1 Consider the following Huffman tree:

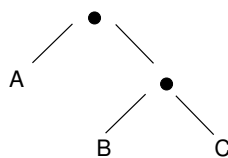


Figure 2: A Huffman tree.

We know that the following string has been encoded using the above tree.

1111 1111 1010 0

4.1.1 What was the input to the coding algorithm? Decode the string and write just the result in the box below.

Answer CCCCBBA

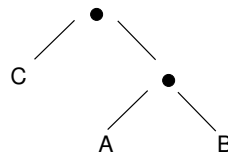
4.1.2 Is the tree of Figure 2 optimal for that input sequence? Why?

Name:

CPR:

page 8 of 9

Answer No. The following tree



gives the encoding 0000 1111 10, which is shorter by 3 bits.

Question 4.2

A directed acyclic graph (DAG) is called a lattice if it contains a vertex from which you can reach every other vertex **and** it contains a vertex that can be reached from every other vertex.

Design an algorithm to determine if a given DAG is a lattice. Describe the algorithm in Danish, or in English, or in pseudocode.

Hint: How many vertices with no incoming edges are in a lattice? How many with no outgoing edges?

Answer If a DAG is a lattice it must have a single source (a single node s of in-degree 0). Find s and do a DFS starting in s and see if everything is visited. Similarly find the single node p with out-degree zero and run a DFS starting in p interpreting the edges in the reverse direction. Check if every node was visited. If s or p does not exist, or are not unique then the graph, or any of the searches has not visited the whole DAG, then this is not a lattice.

Question 4.3 A sensitivity matrix for a pair of vertices u, v in a weighted acyclic graph $G(V, E)$ is a two dimensional array S of $|V|$ by $|V|$ boolean values, such that $S[x, y]$ is 1 if the weight of the edge from x to y can be increased without increasing the weight of the shortest path from u to v . Otherwise $S[x, y] = 0$.

Note that for $S[x, y] = 1$ it is fine if the shortest path itself changes after increasing the weight of (x, y) . It is only important that the *length* of the shortest path does not change. Also the length must remain unchanged if we increase *one* edge weight, not several of those for which $S[x, y] = 1$.

Design an algorithm computing a sensitivity matrix for two given vertices u , and v in G . Describe your algorithm in pseudocode, or English, or Danish.

Name: CPR: page 9 of 9

Answer Initialize S with zeros. Find a shortest path p from u to v , perhaps using Dijkstra's algorithm. For every edge (u', v') not in p set $S[u', v'] \leftarrow 1$. For every edge (u', v') in p remove it from the graph and run Dijkstra again. If the length of the shortest path does not change then, set $S[u', v'] \leftarrow 1$. Put the edge back into the graph before proceeding with the next one on the path.