COMP 3804 — Solutions Assignment 1

Some useful facts:

1. for any real number $x > 0$, $x = 2^{\log x}$.
2. For any real number $x \neq 1$ and any integer $k \geq 1$,
$$1 + x + x^2 + \cdots + x^{k-1} = \frac{x^k - 1}{x - 1}.$$
3. For any real number $0 < \alpha < 1$,
$$\sum_{i=0}^{\infty} \alpha^i = \frac{1}{1 - \alpha}.$$

Master Theorem:

1. Let $a \geq 1$, $b > 1$, $d \geq 0$, and
$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ a \cdot T(n/b) + O\left(n^d\right) & \text{if } n \geq 2. \end{cases}$$
2. If $d > \log_b a$, then $T(n) = O(n^d)$.
3. If $d = \log_b a$, then $T(n) = O(n^d \log n)$.
4. If $d < \log_b a$, then $T(n) = O(n^{\log_b a})$.

Question 1: Write your name and student number.

Solution: Lionel Messi, 10

Question 2: Consider the following recurrence, where n is a power of 6:

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ n^2 + 11 \cdot T(n/6) & \text{if } n \geq 6. \end{cases}$$

- Solve this recurrence using the *unfolding method*. Give the final answer using Big-O notation.
- Solve this recurrence using the *Master Theorem*.

Solution: We write $n = 6^k$. Unfolding gives

$$\begin{aligned}
T(n) &= n^2 + 11 \cdot T(n/6) \\
&= n^2 + 11\left((n/6)^2 + 11 \cdot T(n/6^2)\right) \\
&= (1 + 11/36)\, n^2 + 11^2 \cdot T(n/6^2) \\
&= (1 + 11/36)\, n^2 + 11^2\left((n/36)^2 + 11 \cdot T(n/6^3)\right) \\
&= \left(1 + 11/36 + (11/36)^2\right) n^2 + 11^3 \cdot T(n/6^3) \\
&= \left(1 + 11/36 + (11/36)^2\right) n^2 + 11^3\left((n/6^3)^2 + 11 \cdot T(n/6^4)\right) \\
&= \left(1 + 11/36 + (11/36)^2 + (11/36)^3\right) n^2 + 11^4 \cdot T(n/6^4) \\
&\vdots \\
&= \left(1 + 11/36 + (11/36)^2 + \cdots + (11/36)^{k-1}\right) n^2 + 11^k \cdot T(n/6^k) \\
&= \sum_{i=0}^{k-1} (11/36)^i n^2 + 11^k \cdot T(1) \\
&= \sum_{i=0}^{k-1} (11/36)^i n^2 + 11^k \\
&\leq \sum_{i=0}^{\infty} (11/36)^i n^2 + 11^k \\
&= \frac{1}{1 - 11/36}\, n^2 + 11^k \\
&= \frac{36}{25}\, n^2 + 11^k.
\end{aligned}$$

Note that, since $n = 6^k$, we have $n^2 = 6^{2k} = 36^k > 11^k$. Therefore,

$$T(n) \leq \frac{36}{25}\, n^2 + n^2 = \frac{61}{25}\, n^2 = O(n^2).$$

Using the Master Theorem: We have $a = 11$, $b = 6$, and $d = 2$. Since

$$\log_b a = \log_6 11 = \frac{\log 11}{\log 6} \approx 1.338 < d,$$

the Master Theorem tells us that $T(n) = O(n^d) = O(n^2)$.

Question 3: Consider the following recurrence:

$$T(n) = n + T(n/5) + T(7n/10).$$

In class, we have seen that $T(n) = O(n)$. In this question, you will prove this using the *recursion tree method*.

Recall from class: The root represents the recursion tree on an input of size n. Consider a node u in the recursion tree that represents a recursive call on an input of size m. Then we write the value m at this node u, we give u a left subtree which is a recursion tree for an input of size $m/5$, and we give u a right subtree which is a recursion tree for an input of size $7m/10$. In this way, $T(n)$ is the sum of the values stored at all nodes in the entire recursion tree.

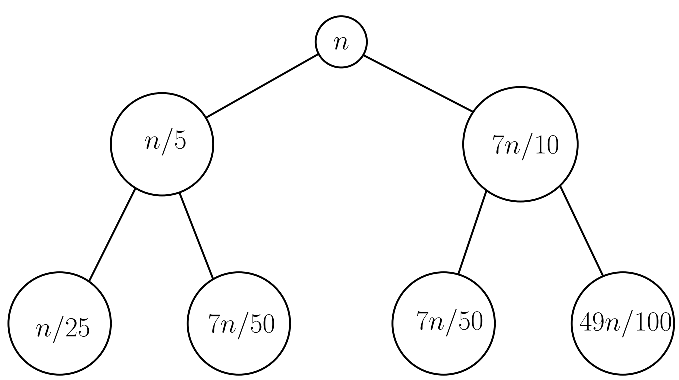
Below, we assume that the *levels* in the recursion tree are numbered $0, 1, 2, \ldots,$, where the root is at level 0. For each $i \geq 0$, let S_i be the sum of the values of all nodes at level i.

- Determine S_0.
- Determine S_1.
- Determine S_2.
- Use induction to prove the following claim: For every $i \geq 0$,

  $$S_i \leq (9/10)^i \cdot n.$$

  Hint: Consider level i, let $k = 2^i$, and let the values stored at the nodes at level i be $m_1, m_2, \ldots, m_k$. What are the values stored at the nodes at level $i + 1$?
- Complete the proof by showing that $T(n) = O(n)$.

Solution: In the following figure, you see levels 0, 1, and 2, in the recursion tree:



From this figure, we see that $S_0 = n$,

$$S_1 = n/5 + 7n/10 = (9/10) \cdot n,$$

and

$$S_2 = n/25 + 7n/50 + 7n/50 + 49n/100 = (9/10)^2 \cdot n.$$

There seems to be a pattern!

Now we prove by induction on i that $S_i \leq (9/10)^i \cdot n$.

Base case: $i = 0$. We have seen above that $S_0 = n$. Since $(9/10)^i \cdot n = n$, the claim is true.

Induction step: Let $i \geq 0$, and assume that $S_i \leq (9/10)^i \cdot n$. We follow the hint: Let $k = 2^i$, and let the values stored at the nodes at level i be $m_1, m_2, \ldots, m_k$. Note that

$$m_1 + m_2 + \cdots + m_k = S_i \leq (9/10)^i \cdot n.$$

1. The values stored at the two children of m_1 are $m_1/5$ and $7m_1/10$. Their sum is $(9/10) \cdot m_1$.
2. The values stored at the two children of m_2 are $m_2/5$ and $7m_2/10$. Their sum is $(9/10) \cdot m_2$.
3. Etc. Etc.
4. The values stored at the two children of m_k are $m_k/5$ and $7m_k/10$. Their sum is $(9/10) \cdot m_k$.

It follows that the sum of the values stored at all nodes at level $i+1$ is equal to

$$S_{i+1} = (9/10) \cdot (m_1 + m_2 + \cdots + m_k) = (9/10) \cdot S_i.$$

We conclude that

$$S_{i+1} = (9/10) \cdot S_i \leq (9/10) \cdot (9/10)^i \cdot n = (9/10)^{i+1} \cdot n.$$

For the last part of the question, we get

$$T(n) \leq \sum_{i=0}^{\infty} (9/10)^i \cdot n = \frac{n}{1 - 9/10} = 10n = O(n).$$

Question 4: Zoltan is not only your friendly TA, he is also the owner of the popular budget airline ZoltanJet that offers flights in Canada. As you all know, there are n airports in Canada. We denote these airports, in order from west to east, by $A_1, A_2, \ldots, A_n$.

William, who is the CEO of ZoltanJet, has designed a *flight plan* which is a list of ordered pairs (A_i, A_j) of airports such that there is a direct flight from A_i to A_j. This flight plan has the following two properties:

- (P.1) Every flight is going eastwards[1]. In other words, if (A_i, A_j) is in the flight plan, then $i < j$.
- (P.2) For any two indices i and j with $1 \leq i < j \leq n$, it is possible to fly from A_i to A_j in at most two *hops*. In other words, either (A_i, A_j) is in the flight plan, or there is an index k such that both (A_i, A_k) and (A_k, A_j) are in the flight plan. Note that, because of (P.1), $i < k < j$.

[1] But how do I get home? A customer service representative will tell you "that is your problem".

Observe that ZoltanJet can guarantee (P.1) and (P.2) by offering direct flights between all $\binom{n}{2} = \Theta(n^2)$ pairs (A_i, A_j) of airports, where $1 \le i < j \le n$.

- Prove that ZoltanJet can guarantee (P.1) and (P.2) using a flight plan having only $O(n \log n)$ pairs of airports. You may assume that n is a power of two.

  Hint: Since this is the divide-and-conquer assignment, you probably have to use ...

Solution: We define $P(n)$ to be the number of pairs of airports in the flight plan if the number of airports is n.

The base case is when $n = 1$. In this case, there is only one airport and, thus, there are no flights in the flight plan, i.e., $P(1) = 0$.

Assume that $n \ge 2$ is a power of two. Let $k = n/2$ so that A_k is the airport in the middle.

1. For each i with $1 \le i \le k-1$, we add (P_i, P_k) to the flight plan.
2. For each j with $k+1 \le j \le n$, we add (P_k, P_j) to the flight plan.
3. Note: By doing this, we can fly from any airport A_i, with $1 \le i \le k$, to any airport A_j, with $k+1 \le j \le n$, in at most two hops.
4. We now apply the construction recursively to the airports A_i with $1 \le i \le k$.
5. We also apply the construction recursively to the airports A_j with $k+1 \le j \le n$.

From this, we obtain the recurrence

$$P(n) = (n-1) + 2 \cdot P(n/2) \le n + 2 \cdot P(n/2).$$

This is the merge-sort recurrence (with a different base case). We have seen in class that this recurrence solves to $P(n) = O(n \log n)$.

Question 5: Professor Justin Bieber needs a fast algorithm that searches for an arbitrary element x in a sorted array $A[1 \ldots n]$ of n numbers. He remembers that there is something called "binary search", which maintains an interval $[\ell, r]$ of indices such that, if x is present in the array, then it is contained in the subarray $A[\ell \ldots r]$. In one iteration, the algorithm takes the middle index, say p, in the interval $[\ell, r]$. Then the algorithm either finds x at the position p, or it recurses in the interval $[\ell, p-1]$, or it recurses in the interval $[p+1, r]$. Unfortunately, Professor Bieber does not remember the expression[2] for p in terms of ℓ and r.

Professor Bieber does remember that, instead of choosing p in the middle of the interval $[\ell, r]$, it is often enough to choose p uniformly at random in this interval. Based on this, he obtains the following algorithm: The input consists of the sorted array $A[1 \ldots n]$, its size n, and a number x. If x is in the array, then the algorithm returns the index p such that $A[p] = x$. Otherwise, the algorithm returns "not present". We assume that all numbers in A are distinct.

[2] is it $\lfloor (r-\ell)/2 \rfloor$, or $\lceil (r-\ell)/2 \rceil$, or $\lfloor (r-\ell+1)/2 \rfloor$, or $\lceil (r-\ell+1)/2 \rceil$?

```
Algorithm BieberSearch(A, n, x):
ℓ = 1; r = n;
while ℓ ≤ r
do p = uniformly random element in {ℓ, ℓ + 1, ..., r};
    if A[p] < x
    then ℓ = p + 1
    else if A[p] > x
         then r = p − 1
         else return p
         endif
    endif
endwhile;
return "not present"
```

Let T be the running time of this algorithm on an input array of length n. Note that T is a random variable. Prove that the expected value of T is $O(\log n)$.
Hint: Most solutions that you find on the internet are wrong.

Solution: In one iteration of the while-loop, the algorithm searches for x in the subarray $A[\ell \ldots r]$; this subarray has length $r - \ell + 1$. In each iteration, if the algorithm does not terminate, either ℓ increases or r decreases; thus, the next iteration searches a smaller subarray.

Let $i \geq 0$ be an integer. We say that the while-loop is in *phase* i if, at the beginning of this iteration,

$$(3/4)^{i+1} \cdot n < r - \ell + 1 \leq (3/4)^i \cdot n.$$

At the start of the first iteration, $r - \ell + 1 = n$ and, thus, the while-loop is in phase 0.

We first determine the largest possible phase number: If an iteration takes place in phase i, then $\ell \leq r$ (this is the condition in the while-loop) and, thus, $1 \leq r - \ell + 1$. It follows that

$$1 \leq (3/4)^i \cdot n,$$

which is equivalent to

$$(4/3)^i \leq n,$$

which is equivalent to

$$i \cdot \log(4/3) \leq \log n,$$

which is equivalent to

$$i \leq \frac{\log n}{\log(4/3)}.$$

Consider one phase i. Let $m = r - \ell + 1$. Divide $\{\ell, \ell + 1, \ldots, r\}$ into three pieces: The first $m/4$ elements, the middle $m/2$ elements, the last $m/4$ elements. If p belongs to the middle piece and if there is a next iteration, with values ℓ' and r', then

$$\ell' - r' + 1 \leq m - m/4 = (3/4) \cdot m \leq (3/4)^{i+1} \cdot n.$$

Thus, the next iteration is in a phase with number at least $i+1$.

Let X_i be the random variable whose value is the number of iterations in phase i. Since p is in the middle piece with probability $1/2$, we have $\mathbb{E}(X_i) \leq 2$. (We have seen this in lecture 5.)

Let c be a constant such that one iteration takes at most c time. Let $L = \frac{\log n}{\log(4/3)}$. Then the running time T satisfies

$$T \leq \sum_{i=0}^{L} c \cdot X_i .$$

Thus,

$$\begin{aligned} \mathbb{E}(T) &\leq \mathbb{E}\left(\sum_{i=0}^{L} c \cdot X_i \right) \\ &= \sum_{i=0}^{L} c \cdot \mathbb{E}(X_i) \\ &\leq \sum_{i=0}^{L} 2c \\ &= 2c(L+1) \\ &= O(\log n). \end{aligned}$$

Question 6: You are given a sequence S consisting of n numbers; not all of these numbers need to be distinct.

Describe an algorithm, in plain English, that decides, in $O(n)$ time, whether or not this sequence S contains a number that occurs more than $n/4$ times.

You may use any result that was proven in class. Justify the correctness of your algorithm and explain why the running time is $O(n)$.

Hint: The algorithm must be comparison-based; you are not allowed to use hashing, bucket-sort, or radix-sort.

Solution: We assume for simplicity that n is divisible by four.

The main observation is the following: If there is a number a that occurs more than $n/4$ times, then a is the $(n/4)$-th smallest number in S, or a is the $(n/2)$-th smallest number in S, or a is the $(3n/4)$-th smallest number in S.

Let us first prove that this observation is correct. Let x be the $(n/4)$-th smallest number in S, let y be the $(n/2)$-th smallest number in S, and let z be the $(3n/4)$-th smallest number in S. We assume, by contradiction, that $a \neq x$, $a \neq y$, and $a \neq z$. There are four possibilities:

1. $a < x$. This is a contradiction, because the number of elements in S that are less than x is less than $n/4$.
2. $x < a < y$. This is a contradiction, because the number of elements in S that are between x and y is less than $n/4$.

3. $y < a < z$. This is a contradiction, because the number of elements in S that are between y and z is less than $n/4$.

4. $z < a$. This is a contradiction, because the number of elements in S that are larger than z is less than $n/4$.

Thus, our main observation is correct.

Based on this, we get the following algorithm:

1. Compute the $(n/4)$-th smallest number, say x, in S. Walk along S and count the number of times that x occurs. If x occurs more than $n/4$ times, then we return x.

2. Compute the $(n/2)$-th smallest number, say y, in S. Walk along S and count the number of times that y occurs. If y occurs more than $n/4$ times, then we return y.

3. Compute the $(3n/4)$-th smallest number, say z, in S. Walk along S and count the number of times that z occurs. If z occurs more than $n/4$ times, then we return z.

4. If the algorithm did not return anything yet, then we know that there is no element in the input that occurs more than $n/4$ times.

Using results proven in class. The entire algorithm runs in $O(n)$ time.