COMP 3804 — Winter 2026
Solutions Problem Set 2

Some useful facts:

1. $1 + 2 + 3 + \cdots + n = n(n+1)/2$.
2. for any real number $x > 0$, $x = 2^{\log x}$.
3. For any real number $x \neq 1$ and any integer $k \geq 1$,
$$1 + x + x^2 + \cdots + x^{k-1} = \frac{x^k - 1}{x - 1}.$$
4. 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
$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ a \cdot T(n/b) + \Theta(n^d) & \text{if } n \geq 2. \end{cases}$$
2. If $d > \log_b a$, then $T(n) = \Theta(n^d)$.
3. If $d = \log_b a$, then $T(n) = \Theta(n^d \log n)$.
4. If $d < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

Question 1: You are given an array $A(1 \dots n)$ of n distinct numbers, and an integer k with $1 \leq k \leq n$.

Describe a comparison-based algorithm that returns, in $O(n)$ time, k numbers in A that are closest to the number 2026. (The k output numbers do not have to be in sorted order. The output may not be unique.)

For example, if $k = 3$ and

$$A = (2028, 10, 2, 2022, 1949, -16, 2025, 2030),$$

then both $(2028, 2025, 2022)$ and $(2028, 2025, 2030)$ are valid outputs.

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

Solution: We first compute an array $B(1 \dots n)$, where, for $i = 1, 2, \dots, n$,

$$B(i) = (|A(i) - 2026|, i).$$

In words, the i-th entry in B is the ordered pair, whose first component is the absolute difference between $A(i)$ and 2026, and whose second component is the index i. The purpose of the second component is the following: After running the algorithm, the entries of the array B are in a different order. However, with each entry, we know the index i in the input array A. From this, we can compute the entry in A. For example, if $B(7) = (5, 37)$, then the 5 came from either 2021 or 2031. By looking at position 37 in A, we know which one it is.

In the next step, we run the median-of-median algorithm from class and compute the k-th smallest number of the array B. In this algorithm, we only consider the first components of the entries. Let the k-th smallest number in B be given by (x, i).

We run the re-arranging algorithm on B, and split it into three parts: Those whose first components are less than x, those whose first components are equal to x, and those whose first components are larger than x.

In the re-arranged array B, we take the first k entries. As described above, from these k entries, we can retrieve k numbers in A that are closest to 2026.

The total running time is $O(n)$, because, as we have seen in class, the median-of-median algorithm takes $O(n)$ time. All other steps take $O(n)$ as well.

Question 2: Consider the following recurrence, where $n \geq 1$ is an integer:

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 1 + T(\lfloor \sqrt{n} \rfloor) & \text{if } n \geq 2. \end{cases}$$

Solve this recurrence, i.e., use Big-O notation to express $T(n)$ as a function of n.

Solution: As always, we assume that n is a "nice" number. For this recurrence, "nice" means that n is of the form

$$n = 2^{2^k},$$

i.e., two to the power 2^k. The main observation is that

$$\sqrt{n} = 2^{2^{k-1}}.$$

We use unfolding:

$$\begin{aligned}
T(n) &= T\left(2^{2^k}\right) \\
&= 1 + T\left(2^{2^{k-1}}\right) \\
&= 2 + T\left(2^{2^{k-2}}\right) \\
&= 3 + T\left(2^{2^{k-3}}\right) \\
&= 4 + T\left(2^{2^{k-4}}\right) \\
&\vdots \\
&= k + T\left(2^{2^{k-k}}\right) \\
&= k + T(2) \\
&= k + 1 + T(1) \\
&= k + 2.
\end{aligned}$$

Since $n = 2^{2^k}$, we have

$$\log n = 2^k$$

and

$$\log\log n = k.$$

Therefore,

$$T(n) = k + 2 = \log\log n + 2 = O(\log\log n).$$

By the way: If n is not "nice", then it is still the case that $T(n) = O(\log\log n)$. You may prove this by yourself. Hint: there is a unique integer k such that

$$2^{2^{k-1}} < n \leq 2^{2^k}.$$

Question 3: Justin Bieber is really impressed by the analysis of the expected running time of the randomized selection algorithm:

```
Algorithm RSelect($S, k$):
Input: Sequence $S$ of numbers, integer $k$ with $1 \leq k \leq |S|$
Output: $k$-th smallest number in $S$
if $|S| = 1$
then return the only element in $S$
else $p =$ uniformly random element in $S$;
     by scanning $S$ and making $|S| - 1$ comparisons, divide it into
     $L = \{x \in S : x < p\}$,
     $M = \{x \in S : x = p\}$,
     $R = \{x \in S : x > p\}$;
     if $k \leq |L|$
     then RSelect($L, k$)
     else if $k \geq 1 + |L| + |M|$
          then RSelect($R, k - |L| - |M|$)
          else return $p$
          endif
     endif
endif
```

Let n be the size of the sequence in the first call to RSelect. In class, the entire computation was divided into *phases*: For any integer $i \geq 0$, a call to algorithm RSelect is in phase i, if

$$(3/4)^{i+1} \cdot n < \text{ the length of the sequence in this call } \leq (3/4)^i \cdot n.$$

Justin wonders why the number 3/4 is used. He thinks that it is much more natural to say that a call to algorithm RSelect is in *Bieber-phase* i, if

$$(1/2)^{i+1} \cdot n < \text{ the length of the sequence in this call } \leq (1/2)^i \cdot n.$$

- Use Bieber-phases to prove that the expected running time of algorithm RSelect on an input sequence of length n is $O(n)$.

Solution: We follow the approach that we have seen in class. Let n be a large integer, let S be a sequence of length n, and let T be the running time of algorithm RSelect(S, k). Note that T is a random variable. We are going to prove that $\mathbb{E}(T) = O(n)$.

At the start of the algorithm, we are in Bieber-phase $i = 0$. During recursive calls, we either stay in the same Bieber-phase or we move to a Bieber-phase with a larger index. Which case happens depends on the pivot.

Consider a call in Bieber-phase i, and let m be the length of the sequence in this call. We know that

$$(1/2)^{i+1} \cdot n < m \leq (1/2)^i \cdot n.$$

For the purpose of the analysis, consider these m numbers in sorted order. Each of the smallest $m/4$ numbers and each of the largest $m/4$ numbers is called *bad*. All other numbers are called *good*.

Assume that the algorithm chooses a good pivot. If there is a next call to the algorithm, then it is on a sequence of length at most

$$m - m/4 = (3/4)m \leq (3/4)(1/2)^i \cdot n.$$

This next call may still be in Bieber-phase i.

Assume that the algorithm chooses another good pivot. If there is a next call to the algorithm, then it is on a sequence of length at most

$$(3/4)^2 m = (9/16)m \leq (9/16)(1/2)^i \cdot n.$$

This next call may still be in Bieber-phase i.

Assume that the algorithm chooses a third good pivot. If there is a next call to the algorithm, then it is on a sequence of length at most

$$(3/4)(9/16)m \leq m/2 \leq (1/2)^{i+1} \cdot n.$$

This next call is in Bieber-phase more than i.

Define the random variable X_i to be the number of calls in Bieber-phase i. Then $\mathbb{E}(X_i)$ is bounded from above by the expected number of times to flip three heads using a coin that comes up heads with probability 1/2. For the first heads, we flip the coin, expected, twice. For the second heads, we flip the coin, expected, twice. For the third heads, we flip the coin, expected, twice. Thus,

$$\mathbb{E}(X_i) \leq 6.$$

The time for re-arranging the sequence during one call in Bieber-phase i is at most

$$c(1/2)^i \cdot n,$$

where c is a constant. We assume that $c = 1$.

The total running time of algorithm RSelect(S, k) is at most

$$T \leq \sum_{i=0}^{\infty} X_i \cdot (1/2)^i \cdot n.$$

Using Linearity of Expectation, we get

$$\begin{aligned}
\mathbb{E}(T) &\leq \sum_{i=0}^{\infty} \mathbb{E}(X_i) \cdot (1/2)^i \cdot n \\
&\leq \sum_{i=0}^{\infty} 6 \cdot (1/2)^i \cdot n \\
&= 6n \sum_{i=0}^{\infty} (1/2)^i \\
&= 12n.
\end{aligned}$$

Question 4: Let $n \geq 2$ be an integer, and let $A[1 \ldots n]$ be an array storing n pairwise distinct numbers.

It is easy to compute the two smallest numbers in the array A: Using $n-1$ comparisons, we find the smallest number in A. Then, using $n-2$ comparisons, we find the smallest number among the remaining $n-1$ numbers. The total number of comparisons made is $2n-3$. By a similar argument, we can find the smallest and largest numbers in A using $2n-3$ comparisons. In this question, you will show that the number of comparisons can be improved.

(4.1) Consider the following algorithm TwoSmallest(A, n), which computes the two smallest numbers in the array A:

```
Algorithm TwoSmallest($A, n$):
if $A[1] < A[2]$        (*)
then $smallest = A[1]$; $secondsmallest = A[2]$
else $smallest = A[2]$; $secondsmallest = A[1]$
endif;
for $i = 3$ to $n$
do if $A[i] < smallest$        (**)
    then $secondsmallest = smallest$; $smallest = A[i]$
    else if $A[i] < secondsmallest$        (***)
         then $secondsmallest = A[i]$
         endif
    endif
endfor;
return $smallest$ and $secondsmallest$
```

In each of the lines (*), (**), and (***), the algorithm *compares* two input numbers.

Assume that A stores a uniformly random permutation of the set $\{1, 2, \ldots, n\}$. Let X be the total number of *comparisons* made when running algorithm TwoSmallest(A, n). Observe that X is a *random variable.* Prove that the expected value of X satisfies

$$\mathbb{E}(X) = 2n - \Theta(\log n).$$

Hint: For each $i = 3, 4, \ldots, n$, use an indicator random variable X_i that indicates whether or not line (***) is executed in iteration i.

Solution: For each $i = 3, 4, \ldots, n$, define the indicator random variable

$$X_i = \begin{cases} 1 & \text{if line (***) is executed in iteration } i, \\ 0 & \text{otherwise.} \end{cases}$$

Note that line (*) is executed exactly once, whereas line (**) is executed exactly $n-2$ times. It follows that

$$X = n - 1 + \sum_{i=3}^{n} X_i.$$

We first determine the expected value of X_i. Since X_i is an indicator random variable, we have

$$\mathbb{E}(X_i) = \Pr(X_i = 1) = \Pr(\text{line (***) is executed in iteration } i).$$

We observe that line (***) is executed in iteration i if and only if $A[i] > \textit{smallest}$, i.e., the smallest number in the subarray $A[1 \dots i]$ is *not* at position i. Since the array is a random permutation, this happens with probability $1 - 1/i$. Thus,

$$\mathbb{E}(X_i) = 1 - \frac{1}{i}.$$

Using Linearity of Expectation and some algebra, we get:

$$\begin{aligned}
\mathbb{E}(X) &= \mathbb{E}\left(n - 1 + \sum_{i=3}^{n} X_i\right) \\
&= n - 1 + \sum_{i=3}^{n} \mathbb{E}(X_i) \\
&= n - 1 + \sum_{i=3}^{n} \left(1 - \frac{1}{i}\right) \\
&= n - 1 + n - 2 - \sum_{i=3}^{n} \frac{1}{i} \\
&= 2n - 3 - \sum_{i=3}^{n} \frac{1}{i} \\
&= 2n - 3 + \left(1 + \frac{1}{2}\right) - \sum_{i=1}^{n} \frac{1}{i} \\
&= 2n - \frac{3}{2} - H_n.
\end{aligned}$$

Since $\frac{3}{2} + H_n = \Theta(\log n)$, we conclude that

$$\mathbb{E}(X) = 2n - \Theta(\log n).$$

Remark: At the end of these solutions, you will see a variant of algorithm TwoSmallest that makes, expected, only $n + \Theta(\log n)$ comparisons.

(4.2) Consider the following algorithm SmallestLargest(A, n), which computes the smallest and largest numbers in the array A:

```
Algorithm SmallestLargest$(A, n)$:
if $A[1] < A[2]$      (*)
then $smallest = A[1]$; $largest = A[2]$
else $smallest = A[2]$; $largest = A[1]$
endif;
for $i = 3$ to $n$
do if $A[i] < smallest$      (**)
   then $smallest = A[i]$
   else if $A[i] > largest$      (***)
        then $largest = A[i]$
        endif
   endif
endfor;
return $smallest$ and $largest$
```

Observe that this algorithm is very similar to algorithm TwoSmallest(A, n).

Assume that A stores a uniformly random permutation of the set $\{1, 2, \dots, n\}$. Let Y be the total number of *comparisons* made when running algorithm SmallestLargest(A, n). Observe that Y is a *random variable*. Argue, in a few sentences, that the same analysis as for **(4.1)** implies that

$$\mathbb{E}(Y) = 2n - \Theta(\log n).$$

Solution:

- Both algorithms execute line (*) exactly once.
- Both algorithms execute line (**) exactly $n - 2$ times.
- In both algorithms, *smallest* is the smallest number seen so far.
- In both algorithms, line (***) is executed if and only if $A[i] > smallest$.
- Therefore, $\mathbb{E}(X) = \mathbb{E}(Y)$.

(4.3) Assume that $n \geq 2$ is a power of 2. Furthermore, assume that the array $A[1 \dots n]$ stores an arbitrary sequence of n pairwise distinct numbers. Describe, in English, an algorithm that computes the two smallest numbers in the array A, and that makes $n + \log n - 2$ comparisons. Justify your answer.
Hint: Consider a tennis tournament with n players. The players are numbered from 1 to n. For each player i, the number $A[i]$ is the ATP-ranking of this player.

The n players play as they do in any Grand Slam tournament: They play against each other in pairs; after any game, the winner goes to the next round and the loser goes home. This is a special tournament: If player i plays against player j, then the player with the smaller A-value (i.e., higher ATP ranking) is guaranteed to win.

In this way, the smallest number in A corresponds to the tennis player who wins the tournament. The second smallest number in A corresponds to the second best player. This second best player must lose some game. Who can beat the second best player?

Solution: We write $n = 2^k$.

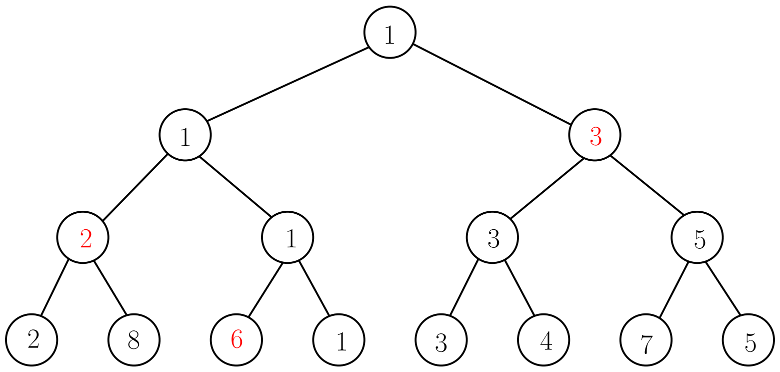
- The algorithm starts by constructing a binary tree with levels $0, 1, 2, \ldots, k$, where the root has level 0. The tree has $2^k = n$ leaves, which are all at level k.
- Write the values $A[1], A[2], \ldots, A[n]$ at the leaves (at level k), from left to right.
- For $i = k-1, k-2, \ldots, 0$:
  - Walk along the 2^i nodes at level i, from left to right.
  - For each node u at level i, let a and b be the two numbers stored at its two children.
  - If $a < b$, then we write the number a in the node u. Otherwise, we write the number b in the node u.

Observe that the root will store the smallest number in the entire array $A[1 \ldots n]$.

So far, the number of comparisons made is equal to

$$\sum_{i=0}^{k-1} 2^i = 1 + 2 + 2^2 + 2^3 + \cdots + 2^{k-1} = 2^k - 1 = n - 1.$$

Here is an example for the array $[2, 8, 6, 1, 3, 4, 7, 5]$:



Given this binary tree, how do we find the second smallest number? Let x be the smallest number and let y be the second smallest number in the array $A[1 \ldots n]$. We know that x is stored at the root. The number y must have lost exactly one game. The only way for y to lose is by playing against x. It follows that, at some point, y must have been compared with x.

How to find y:

- Initialize an empty list L.
- Start at the leaf storing x, and walk to the root of the tree. Note that all nodes on this path store x.
- Every time we walk from the current node, say u, to its parent, say v, we add to L the number stored at the other child of v.
- Note that, at the end, L has size $k = \log n$. Also, L stores the number y we are trying to find. In fact, y is the smallest number in L. (In the example, L consists of exactly the red numbers.) So far, we did not make any comparison.
- Walk along L and find the smallest element using $|L| - 1 = \log n - 1$ comparisons.

The total number of comparisons is equal to

$$(n-1) + (\log n - 1) = n = \log n - 2.$$

(4.4) Assume that $n \geq 2$ is an even integer. Furthermore, assume that the array $A[1 \ldots n]$ stores an arbitrary sequence of n pairwise distinct numbers. Describe, in English, an algorithm that computes the smallest and largest numbers in the array A, and that makes $\frac{3}{2}n - 2$ comparisons. Justify your answer.
Hint: If $A[1] < A[2]$, is it possible that $A[1]$ is the largest number? If $A[1] > A[2]$, is it possible that $A[1]$ is the smallest number? If $A[3] < A[4]$, is it possible that $A[3]$ is the largest number? If $A[3] > A[4]$, is it possible that $A[3]$ is the smallest number?

Solution: The algorithm does the following:

Step 1: Intialize $S = \emptyset$ and $L = \emptyset$.

Step 2: For $i = 1, 2, \ldots, n/2$:

- If $A[2i-1] < A[2i]$: Add $A[2i-1]$ to S and add $A[2i]$ to L.
- Else: Add $A[2i]$ to S and add $A[2i-1]$ to L.

Comment: In this step, we make exactly $n/2$ comparisons. At the end of this step, S has size $n/2$ and L has size $n/2$. Also, at the end of this step, the smallest number in $A[1 \ldots n]$ is the smallest element in S, whereas the largest number in $A[1 \ldots n]$ is the largest element in L.

Step 3: Walk along the elements of S and find the smallest element using $|S| - 1 = n/2 - 1$ comparisons.

Step 4: Walk along the elements of L and find the largest element using $|L| - 1 = n/2 - 1$ comparisons.

The total number of comparisons made in Steps 1, 2, and 3 is

$$n/2 + (n/2 - 1) + (n/2 - 1) = 3n/2 - 2.$$

Remark Question 4:

We have seen in Question **(4.1)** that algorithm TwoSmallest(A, n) computes the two smallest numbers in the array $A[1 \dots n]$. If the elements of this array are in random order, then the expected number of comparisons is equal to

$$2n - \Theta(\log n).$$

Below, we will see that a variant of algorithm TwoSmallest solves the same problem making, expected, only

$$n + \Theta(\log n)$$

comparisons.

Consider the following algorithm BetterTwoSmallest(A, n), which computes the two smallest numbers in the array $A[1 \dots n]$:

```
Algorithm BetterTwoSmallest(A, n):
if A[1] < A[2]        (*)
then smallest = A[1]; secondsmallest = A[2]
else smallest = A[2]; secondsmallest = A[1]
endif;
for i = 3 to n
do if A[i] < secondsmallest        (**)
    then if A[i] < smallest        (***)
            then secondsmallest = smallest; smallest = A[i]
            else secondsmallest = A[i]
            endif
    endif
endfor;
return smallest and secondsmallest
```

Assume that A stores a uniformly random permutation of the set $\{1, 2, \dots, n\}$. Let X be the total number of *comparisons* made when running algorithm BetterTwoSmallest(A, n). We are going to determine the expected value of this random variable.

For each $i = 3, 4, \dots, n$, define the indicator random variable

$$X_i = \begin{cases} 1 & \text{if line (***) is executed in iteration } i, \\ 0 & \text{otherwise.} \end{cases}$$

Note that line (*) is executed exactly once, whereas line (**) is executed exactly $n - 2$ times. It follows that

$$X = n - 1 + \sum_{i=3}^{n} X_i.$$

We first determine the expected value of X_i. Since X_i is an indicator random variable, we have

$$\mathbb{E}(X_i) = \Pr(X_i = 1) = \Pr(\text{line (***) is executed in iteration } i).$$

We observe that line (***) is executed in iteration i if and only if $A[i] < \mathit{secondsmallest}$, i.e., $A[i]$ is less than the second smallest number in the subarray $A[1 \ldots i-1]$.

Let x be the smallest number in the subarray $A[1 \ldots i]$, and let y be the second smallest number in $A[1 \ldots i]$. Then, line (***) is executed in iteration i if and only if $A[i] = x$ or $A[i] = y$.

Since the array is a random permutation, the probability that $A[i] = x$ is equal to $1/i$. Similarly, the probability that $A[i] = y$ is equal to $1/i$. Since the events "$A[i] = x$" and "$A[i] = y$" are disjoint, it follows that line (***) is executed in iteration i with probability $2/i$. Thus,

$$\mathbb{E}(X_i) = \frac{2}{i}.$$

Using Linearity of Expectation and some algebra, we get:

$$\begin{aligned}
\mathbb{E}(X) &= \mathbb{E}\left(n - 1 + \sum_{i=3}^{n} X_i\right) \\
&= n - 1 + \sum_{i=3}^{n} \mathbb{E}(X_i) \\
&= n - 1 + \sum_{i=3}^{n} \frac{2}{i} \\
&= n - 1 + \sum_{i=1}^{n} \frac{2}{i} - (2 + 1) \\
&= n - 4 + 2 \cdot H_n \\
&= n + \Theta(\log n).
\end{aligned}$$