1. Using the set definition of the big O, prove that $O(x^2) \subseteq O(2^x)$.

   Solution.

   Let $f(x) \in O(x^2)$.
   Our goal is to show that $f(x) \in O(2^x)$.
   This means that there exists $c, x_0 > 0$ such that

   $$0 \leq |f(x)| \leq cx^2 \text{ for all } x \geq x_0$$

   We know that exponential functions such as 2^x grow faster than polynomial functions such as x^2.

   Given a constant $c > 0$. This implies that there exists x_1 (let $x_1 = 5$) such that

   $$cx^2 \leq c(2^x) \text{ for all } x \geq 5$$

   Given $c > 0$, let $x_2 = \max\{x_0, 5\}$. We have

   $$0 \leq |f(x)| \leq cx^2 \leq c(2^x) \text{ for all } x \geq x_2,$$

   By set definition of big O, $f(x) \in O(2^x)$, which means that $O(x^2) \subseteq O(2^x)$.

2. Prove: $\sum_{k=1}^{n} \frac{1}{k} = \theta(ln(n))$. You may use the limit definition of big Theta.

   Solution.

   Let $H(i) = \sum_{k=1}^{i} \frac{1}{k} = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{i}$
   We can approximate $H(i)$ using Riemann sums.

   $$H(i) \approx \int \frac{1}{i} di = ln(i)$$

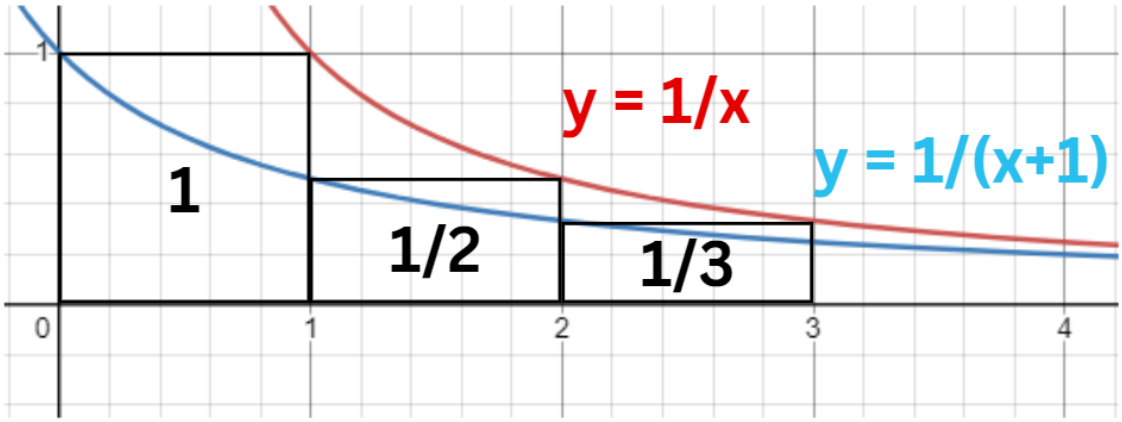

Figure 1: Desmos Graph of the sum

The next goal is to find a lower bound and upper bound for $H(n)$. Notice from figure 1, that the area under $y = 1/x$ is greater than $H(n)$ while the area under $y = 1/(x + 1)$ is less than $H(n)$.

Ignoring the constant of integration, we have

$$\int \frac{1}{x}dx = ln(x) \text{ and } \int \frac{1}{x+1}dx = ln(x+1).$$

Thus, we have obtained bounds for $H(n)$. $ln(n+1) \leq H(n) \leq ln(n)$ holds over all positive n.

Clearly, $ln(n) \in \theta(ln(n))$. We have to show that $ln(n+1) \in \theta(ln(n))$.

We use the limit definition of Big Theta and L'Hopital's Rule.

$$\lim_{n\to\infty} \frac{ln(n+1)}{ln(n)} = \lim_{n\to\infty} \frac{\frac{1}{n+1}}{\frac{1}{n}} = \lim_{n\to\infty} \frac{n}{n+1} = 1$$

Since $0 < 1 < \infty$, $ln(n+1) \in \theta(ln(n))$.

Since we know that $ln(n+1) \in \theta(ln(n))$, $ln(n) \in \theta(ln(n))$ and $ln(n+1) \leq H(n) \leq ln(n)$, by the Squeeze Theorem, $H(n) \in \theta(ln(n))$.

3. There are several operations which can be performed on sets in Python. If A and B are sets, then A.difference(B) is a new set with all elements from A which are not also in B. This can also be accomplished with the syntax $A - B$. Submit a Python program which performs this operation without using the difference function or the $-$ operator. Output the result as a single line of space-separated integers sorted in ascending order. Your algorithm must run in $O((n+m)log(n+m))$.

   Solution.

   The code for the difference algorithm is in the jupyter notebook.

   Time Complexity:
   Let n be the length of the first set A and m be the length of the second set B.

   - Getting the input is $\theta(n) + \theta(m)$.
   - Running the for loop through the elements of A is $\theta(n)$. In each iteration of the loop, we check if the element of A is not in B. The "not in" operation for sets runs in $\theta(1)$. The add operation also runs in constant time $\theta(1)$. Thus, the for loop runs in $\theta(n)$.
   - Converting the set into a list is $\theta(n)$.
   - Sorting the list is $\theta(nln(n))$.
   - Lastly, printing the elements of the output list is at most $\theta(n)$.

   The overall time complexity of the difference algorithm is $\theta(nln(n))$. Our algorithm runs under $O((n+m)log(n+m))$ time.

4. Consider the activity selection problem where there are n activities. Recall that the greedy strategy we employed in class is "Repeatedly choose available non-conflicting activities with the earliest finishing time." For each of the following alternative greedy strategies:

   - Provide a Python implementation of the resulting greedy algorithm. (Each of your implementations must run in $O(n^2log(n).)$ Indicate and briefly explain the asymptotic time complexity of each of your implementations.
   - Either prove that the resulting algorithm always constructs an optimal schedule or describe a small input example for which the algorithm does not produce an optimal schedule. Assume that all algorithms break ties arbitrarily (that is, in a manner that is completely out of your control).

(a) Repeatedly choose available non-conflicting activities with the latest finishing time.

(b) Repeatedly choose available non-conflicting activities with the earliest starting times.

(c) Repeatedly choose available non-conflicting activities with the latest starting time.

(d) Repeatedly choose available non-conflicting activities with the shortest duration.

Solution.

(a) Greedy Strategy: choose activity with the latest finishing time

This greedy strategy is not optimal.

Let X be the set of activities chosen using our greedy strategy.

Let O be the optimal set of activities.

Counter Example:

Given 3 activities in format [starting time, finish time]: [1,8], [3,4], [5,6].

Our greedy algorithm will end up with $X = \{[1, 8]\}$ since it would choose the first activity since it has the latest finishing time. The two remaining activities conflict with the first activity.

However, the optimal solution would have two activities: $O = \{[3, 4], [5, 6]\}$.

Since $|X| < |O|$, this greedy strategy is not optimal.

Time Complexity:

Let n be the number of activities.

- Getting the input is $\theta(n)$.
- Creating the intervals list is $\theta(n)$.
- Sorting n intervals is $\theta(n\log(n))$
- The for loop runs through all the possible activities, so it runs in $\theta(n)$. In each iteration of the for loop, the algorithm checks if there is a conflict with any of the activities in the solution list. This inner for loop runs at most in $\theta(n)$. The entire for loop runs in $\theta(n^2)$.
- Lastly, printing the activities in the solution is $\theta(n)$.

Thus, the overall time complexity of this greedy algorithm is $\theta(n^2)$. The algorithm runs in less than $O(n^2 log(n))$.

(b) Greedy Strategy: choose activity with the earliest starting time

This greedy strategy is not optimal.

Let X be the set of activities chosen using our greedy strategy.

Let O be the optimal set of activities.

Counter Example:

Given 3 activities in format [starting time, finish time]: [1,10], [2,5], [6,9].

Our greedy algorithm will end up with $X = \{[1, 10]\}$ since it will choose the first activity since it begins at the earliest time. This activity would end at time 10 so the algorithm would be unable to do the second or third activities.

However, the optimal solution would do the second and third activities: $O = \{[2, 5], [6, 9]\}$.

Since $|X| < |O|$, this greedy strategy is not optimal.

Time Complexity:

Let n be the number of activities.

- Getting the input is $\theta(n)$.
- Creating the intervals list is $\theta(n)$.

- Sorting n intervals is $\theta(n\log(n))$
- The for loop runs through all the possible activities, so it runs in $\theta(n)$. In each iteration of the for loop, the algorithm checks if there is a conflict with any of the activities in the solution list. This inner for loop runs at most in $\theta(n)$. The entire for loop runs in $\theta(n^2)$.
- Lastly, printing the activities in the solution is $\theta(n)$.

Thus, the overall time complexity of this greedy algorithm is $\theta(n^2)$. The algorithm runs in less than $O(n^2 log(n))$.

(c) Greedy Strategy: choose activity with the latest starting time

Goal: Show that this greedy algorithm is optimal.

Let X be the schedule of activities selected using the greedy strategy, and let O be the optimal schedule of activities.

We have to show that $|X| = |O|$.

Let $x_1, x_2, ..., x_k$ be the activities added to X in order by decreasing starting time.

Meanwhile, let $o_1, o_2, ..., o_m$ be the activities in O ordered by decreasing starting time.

Using the greedy stays ahead argument, we need to show that each of the classes in X begins later or at least at the same time as the corresponding class in O.

We prove the following lemma:

Lemma 1. *Let S_{x_i} be the starting time of the activity x_i.*
For all $r \leq k, S_{x_r} \geq S_{o_r}$.

Proof.

We use proof by induction.

For the base case, assume that $r = 1$. According to our greedy strategy, X would choose the activity with the latest possible starting time. Thus, $S_{x_1} \geq S_{o_1}$.

For the induction hypothesis, assume that for $r > 1$, $S_{x_{r-1}} \geq S_{o_{r-1}}$. Since the $(r-1)$th class in X starts later or at least at the same time as the $(r-1)$th class in O, any class that can be added to O can also be added to X. At worst, you pick the same class to add to X. Thus, $S_{x_r} \geq S_{o_r}$, and we have proved the lemma.

Now, we will show that X is an optimal solution.

Proof.

We use proof by contradiction.

Suppose X is not optimal. Thus, $k < m$. This means that there exists a class o_{k+1} in O that is not in X. This class ends before class o_k starts at S_{o_k}. By the lemma, we know that $S_{x_k} \geq S_{o_k}$ so the class o_{k+1} ends before class x_k. Thus, class o_{k+1} should have been added to X according to our greedy algorithm. Hence, k cannot be less than m, so by contradiction $k = m$.

X is an optimal solution.

Time Complexity:
Let n be the number of activities.

- Getting the input is $\theta(n)$.
- Creating the intervals list is $\theta(n)$.
- Sorting n intervals is $\theta(n\log(n))$
- The for loop runs through all the possible activities, so it runs in $\theta(n)$. In each iteration of the for loop, the algorithm checks if there is a conflict with any of the activities in the solution list. This inner for loop runs at most in $\theta(n)$. The entire for loop runs in $\theta(n^2)$.

- Lastly, printing the activities in the solution is $\theta(n)$.

Thus, the overall time complexity of this greedy algorithm is $\theta(n^2)$. The algorithm runs in less than $O(n^2 log(n))$.

(d) Greedy Strategy: choose activity with the shortest duration

This greedy strategy is not optimal.

Let X be the set of activities chosen using our greedy strategy.

Let O be the optimal set of activities.

Counter Example:
Given 3 activities in format [starting time, finish time]: [1,5], [4,7], [6,10].

Our greedy algorithm will end up with $X = \{[4, 7]\}$ since it will choose the second activity since it has the shortest duration, lasting only 3 time units. However, the first and third activities on the list conflict with the second activity.

However, the optimal solution would be able to do two activities, $O = \{[1, 5], [6, 10]\}$.

Since $|X| < |O|$, this greedy strategy is not optimal.

Time Complexity:
Let n be the number of activities.

- Getting the input is $\theta(n)$.
- Creating the intervals list is $\theta(n)$.
- Sorting n intervals is $\theta(n\log(n))$
- The for loop runs through all the possible activities, so it runs in $\theta(n)$. In each iteration of the for loop, the algorithm checks if there is a conflict with any of the activities in the solution list. This inner for loop runs at most in $\theta(n)$. The entire for loop runs in $\theta(n^2)$.
- Lastly, printing the activities in the solution is $\theta(n)$.

Thus, the overall time complexity of this greedy algorithm is $\theta(n^2)$. The algorithm runs in less than $O(n^2 log(n))$.

5. A ship arrives at a dock, with n containers with integer weights $\omega_1, \omega_2, ..., \omega_n$. Standing on the dock is an (unlimited) set of trucks, each of which can hold an integer K units of weight. You can stack multiple containers in each truck, subject to the weight restriction of K; the goal is to minimize the number of trucks that are needed in order to carry all the containers. There is currently no known efficient and exact algorithm for this problem.

   A greedy algorithm you might use for this is the following. Start with an empty truck, and begin piling containers 1, 2, 3, ... into it until you get to a container that would overflow the weight limit. Now declare this truck "loaded" and send it off; then continue the process with a fresh truck. This algorithm, by considering trucks one at a time, may not achieve the most efficient way to pack the full set of containers into an available collection of trucks.

   (a) Provide a Python implementation of the given greedy algorithm which outputs the number of trucks used. No need to indicate or prove time complexity.

   (b) Give an example of a set of weights, and a value of K, where this algorithm does not use the minimum possible number of trucks.

   (c) Prove that the number of trucks used by this algorithm is within a factor of 2 of the minimum possible number, for any set of weights and any K.

Solution.

(a) The greedy algorithm is in the jupyter notebook.

(b) Let $n = 4$ and $K = 5$.

The four containers weigh 3, 3, 2, 2, respectively.

According to the greedy algorithm, the first truck will load the first container. It will then be sent off since it cannot load the second container. The second truck would then load containers 2 and 3, and the third truck would load container 4. Thus, our greedy algorithm would use three trucks.

However, the optimal solution will only use two trucks to load all four containers. Each truck will have a container weighing 3 and another container weighing 2.

Thus, the greedy algorithm is not optimal.

(c) Let N^* be the optimal number of trucks.

Let N be the number of trucks used by our greedy algorithm.

Let K be the maximum weight that a truck can carry.

Suppose that there are n containers in total with weights $\omega_1, \omega_2, ..., \omega_n$.

Our goal is to find an upper bound and a lower bound for N that are within a factor of 2 of N^*.

Since N^* is the optimal number of trucks,

$$N^* \leq N.$$

Next, we have to find an upper bound for N.

Clearly, if $N = 1$, then

$$N = N^*.$$

Moving forward, we consider when $N > 1$.

Lemma 2.

$$N^* \geq \frac{1}{K} \sum_{i=1}^{n} \omega_i$$

The summation, $\frac{1}{K} \sum_{i=1}^{n} \omega_i$ represents the absolute minimum number of trucks to carry n containers, where each truck carries its maximum weight K.

Consider an arbitrary truck l, where C_l are the containers in truck l with total combined weight W_l.

For any $l > 1$,

$$W_l + W_{l-1} > K. \tag{1}$$

According to our greedy algorithm, the weight in truck l (W_l) would exceed the weight limit K if it were piled on top of the containers in truck $l - 1$.

Lemma 3.

$$\sum_{l=1}^{N} W_l = \sum_{i=1}^{n} \omega_i.$$

The total weight of all the containers is just equal to the combined weight of all the containers in each truck.

Our next objective is to find a lower bound for $\sum_{l=1}^{N} W_l$ in terms of N.

Consider two cases depending on the parity of N.

Case 1. Suppose N is even so $N = 2m$ for some integer m. Equivalently, $m = \frac{N}{2}$.

By inequality 1, it follows that,

$$\sum_{l=1}^{N} W_l = \sum_{l=1}^{m} (W_{2l} + W_{2l-1}) > Km. \tag{2}$$

From inequality 2, we have

$$\sum_{l=1}^{N} W_l > K(\frac{N}{2}) = \frac{1}{2}K(N). \tag{3}$$

From inequality 3 and lemmas 2 and 3, we have

$$\frac{1}{2}K(N) < \sum_{l=1}^{N} W_l = \sum_{i=1}^{n} \omega_i \leq KN^*.$$

Thus, $\frac{1}{2}(N) < N^*$ so $N < 2N^*$.

Case 2. Suppose N is odd so $N = 2m+1$ for some integer m. Equivalently, $m = \frac{N-1}{2}$. By inequality 1, it follows that,

$$\sum_{l=1}^{N} W_l = \sum_{l=1}^{m} (W_{2l} + W_{2l-1}) + W_{2l+1} > Km. \tag{4}$$

From inequality 4, we have

$$\sum_{l=1}^{N} W_l > K(\frac{N-1}{2}) = \frac{1}{2}K(N-1). \tag{5}$$

From inequality 5 and lemmas 2 and 3, we have

$$\frac{1}{2}K(N-1) < \sum_{l=1}^{N} W_l = \sum_{i=1}^{n} \omega_i \leq KN^*.$$

Thus, $\frac{1}{2}(N-1) < N^*$ so $N - 1 < 2N^*$, which is equivalent to $N \leq 2N^*$.
We have found an upper bound and lower bound for N within a factor of 2 of N^*.