Problem 1. (25 points)

Consider the Activity Selection problem we discussed in class. Suppose that instead of always selecting the first activity to finish, we instead select the last activity to start (as long as it is compatible with all the previously selected activities). In other words, instead of considering the activities by "earliest finish", we consider them by "latest start". Explain why this approach is a greedy algorithm, analyze its time complexity, and prove that it yields an optimal solution (greedy choice and optimal substructure).

Answer: In this problem we are sorting the activity by start time, and we schedule first the activity that starts last. In this way we leave as much time as possible for other activities to be scheduled. In this sense, this algorithm is greedy. The complexity of this algorithm for an input of n tasks is still $O(n \log n)$ since we only modify the type of ordering compared to the algorithm we covered in class. The proof of optimality is almost identical to the one covered in the slides.

First, we need to prove that there is at least one optimal solution that contains the greedy choice (last start). **Proof:** Suppose A is an optimal solution for a set S of n tasks ordered by start time. Let's also order the activities in A by start time. Let k be the last activity in A. If $k = n$, the schedule A begins with a greedy choice (latest start time). If $k \neq n$, we show that there is another optimal solution B that begins with the greedy choice (activity n). Let $B = (A - \{k\}) \cup \{n\}$. We need to show first that B is feasible: activities in B are non-conflicting because activities in A are non-conflicting, k is the last activity to start in A and $s_k \leq s_n$. Since we have $|B| = |A|$, B is optimal.

For the optimal substructure we want to prove that A is optimal to S if and only if $A' = A - \{n\}$ is optimal to $S' = \{i \in S : f_i \leq s_n\}$.

One direction: let us prove that if A is optimal to S, then $A' = A - \{n\}$ is optimal to $S' = \{i \in S : f_i \leq s_n\}$. **Proof:** By contradiction. If we could find a solution B' to S' with more activities than A', adding activity n to B' would yield a solution B to S with more activities than A contradicting the optimality of A.

Opposite direction: let us prove that if that if A' is optimal to $S' = \{i \in S : f_i \leq s_n\}$ then $A = A' \cup \{n\}$ is optimal to S. **Proof:** By contradiction. If A is not optimal, we could a solution $B = B' \cup \{n\}$ with more activities than A. But B' is a solution to S' because the tasks in B' are non-overlapping. But $|B'| > |A'|$ which contradicts the optimality of A'.

Problem 2. (25 points)

A server has n customer waiting to be served. The service time required by each customer is known in advance: it is t_i minutes for customer i. So if, for example, the customers are served in order of increasing i, then the i-th customer has to wait $\sum_{j=1}^{i} t_j$ minutes. We want to minimize the total waiting time:

$$T = \sum_{i=1}^{n} (\text{time spent waiting by customer } i)$$

Give a greedy (efficient) algorithm for computing the optimal order in which to process the customers.

Answer: We simply proceed by a greedy strategy, by sorting the customers in the increasing order of service times and servicing them in this order. The running time is $O(n \log n)$.

We prove the correctness directly (instead of using the greedy choice/optimal substructure proof). For any ordering of the customers, let $\sigma(j)$ denotes the j-th customer in the ordering. Then:

$$T = \sum_{i=1}^{n} \sum_{j=1}^{i} t_{\sigma(i)} = \sum_{i=1}^{n} (n - i + 1) t_{\sigma(i)}$$

For any ordering, if $t_{\sigma(i)} > t_{\sigma(j)}$ for $i < j$, then swapping the positions of the two customers gives a better ordering (i.e., the value of T is decreased). Since we can generate all possible orderings by swaps, an ordering which has the property that $t_{\sigma(1)} \leq \ldots \leq t_{\sigma(n)}$ must be the global optimum.

Instead if one decided to prove the greedy choice property and the optimal substructure, here is my solution. For the greedy choice, we need to show that there is at least one optimal solution that contains the greedy choice, i.e., serving first the customer that has the smallest service time. **Proof:** Let A be an optimal ordering where $\sigma(j)$ denote the j-th customer in the ordering. If A begins with the greedy choice, i.e., if $\sigma(1) = j$ for the shortest service time $t_j = \min_i\{t_i\}$, we are done. If not, then the first customer in A must have a service time t_k equal to t_j (for some $k \neq j$), otherwise the total waiting time

$$T = \sum_{i=1}^{n} \sum_{j=1}^{i-1} t_{\sigma(i)} = \sum_{i=1}^{n} (n - i) t_{\sigma(i)}$$

would increase, violating the optimality of A. If we swap out customer k and swap in customer j in A, we can create a new ordering B which begins with the greedy choice, as we wanted.

For the optimal substructure we need to show that A is an optimal ordering for the set of all the customers $S = \{1, 2, \ldots, n\}$ and $\sigma(1) = j$ is (one of) the customers with the smallest service time, if and only if $A' = A - \{j\}$ is an optimal ordering for the set of customers $S' = \{1, 2, \ldots, j-1\} \cup \{j+1, j+2, \ldots n\}$.

One direction: let us prove that if A is an optimal ordering for the set of all the customers $S = \{1, 2, \ldots, n\}$ and $\sigma(1) = j$ is (one of) the customers with the smallest service time, then $A' = A - \{j\}$ is an optimal ordering for the set of customers $S' = \{1, 2, \ldots, j-1\} \cup \{j+1, j+2, \ldots n\}$. **Proof:** For sake of contradiction, let's assume that A' is not optimal for S' (while at the same time A is optimal for S). That means that we can find another solution B for S' which the total waiting time is shorter. However recall that

$$T = \sum_{i=1}^{n} (n - i) t_{\sigma(i)} = (n - 1) t_{\sigma(1)} + \sum_{i=2}^{n} (n - i) t_{\sigma(i)}$$

If B has a shorter total waiting time for S' that means that $\sum_{i=2}^{n} (n - i) t_{\sigma(i)}$ is smaller. In this case we could add to B the greedy choice j and obtain a better solution than A, which creates a contradiction because A is optimal.

Opposite direction: let us prove that if A' is an optimal ordering for the set of customers $S' = \{1, 2, \ldots, j-1\} \cup \{j+1, j+2, \ldots n\}$ where $\sigma(1) = j$ is (one of) the customers with the smallest service time, then $A = A' \cup \{j\}$ is an optimal ordering for the set of all the customers $S = \{1, 2, \ldots, n\}$. **Proof:** For sake of contradiction, let's assume that A is not optimal for S (while at the same time A' is optimal for S'). That means that we can find another solution B for S which the total waiting time is shorter. However recall that

$$T = \sum_{i=1}^{n} (n - i) t_{\sigma(i)} = (n - 1) t_{\sigma(1)} + \sum_{i=2}^{n} (n - i) t_{\sigma(i)}$$

If B has a shorter total waiting time for S that means that $\sum_{i=1}^{n}(n-i)t_{\sigma(i)}$ is smaller. In this case $\sum_{i=2}^{n}(n-i)t_{\sigma(i)}$ will be smaller for S', contradicting the fact that A' is optimal for S'.

Problem 3. (25 points)

Assume that you are given two arrays $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_n\}$ of n real numbers.

1. Describe an efficient greedy algorithm to determine an ordering of the elements of A and B such that $W = \sum_{i=1}^{n} |a_i - b_i|$ is minimized
2. Analyze the time complexity of your algorithm
3. State and prove the greedy-choice property of your algorithm
4. State and prove the optimal substructure property of your algorithm

Hint: consider using the following fact. Given real numbers $x_1 \leq x_2$ and $y_1 \leq y_2$, then

$$|x_1 - y_1| + |x_2 - y_2| \leq |x_1 - y_2| + |x_2 - y_1|\,.$$

Answer: The greedy algorithm works as follows.

Sort both arrays A and B. Suppose the new order is $A = \{a_{\sigma(1)}, \ldots, a_{\sigma(n)}\}$ and $B = \{b_{\sigma(1)}, \ldots, b_{\sigma(n)}\}$, where $a_{\sigma(1)} \leq a_{\sigma(2)} \leq \ldots \leq a_{\sigma(n)}$, and $b_{\sigma(1)} \leq b_{\sigma(2)} \leq \ldots \leq b_{\sigma(n)}$. Pick the pairs $(a_{\sigma(1)}, b_{\sigma(1)})$, $(a_{\sigma(2)}, b_{\sigma(2)}), \ldots, (a_{\sigma(n)}, b_{\sigma(n)})$. The algorithms runs in $O(n \log n)$ time.

First, we have to show that a greedy choice at the first step results in an optimal solution. In other words, we prove that in the optimal solution O, $(a_{\sigma(1)}, b_{\sigma(1)})$ are paired. Let O' be another optimal solution where $a_{\sigma(1)}$ is paired with $b_{\sigma(i)}$ and $b_{\sigma(1)}$ is paired with $a_{\sigma(j)}$, where i and j cannot be 1. Since $i > 1$ and $j > 1$, and the arrays are sorted, then $a_{\sigma(1)} \leq a_{\sigma(j)}$ and $b_{\sigma(1)} \leq b_{\sigma(i)}$. But we know that given real numbers $a_{\sigma(1)} \leq a_{\sigma(j)}$ and $b_{\sigma(1)} \leq b_{\sigma(i)}$

$$|a_{\sigma(1)} - b_{\sigma(1)}| + |a_{\sigma(j)} - b_{\sigma(i)}| \leq |a_{\sigma(1)} - b_{\sigma(i)}| + |a_{\sigma(j)} - b_{\sigma(1)}|$$

and therefore the solution in which $(a_{\sigma(1)}, b_{\sigma(1)})$ are paired is also optimal.

For the optimal substructure, we need to show that Q is an an optimal solution to the subproblem that requires us to pair optimally $A - \{a_{\sigma(1)}\}$,$B - \{b_{\sigma(1)}\}$ if and only if $Q \cup \{(a_{\sigma(1)}, b_{\sigma(1)})\}$ is optimal for the original problem that require us to pair A, B.

One direction: let us prove that if an optimal solution O to the original problem that require us to pair A, B begins with the pair $(a_{\sigma(1)}, b_{\sigma(1)})$, then the solution $Q = O - \{(a_{\sigma(1)}, b_{\sigma(1)})\} = \{(a_{\sigma(2)}, b_{\sigma(2)}), \ldots, (a_{\sigma(n)}, b_{\sigma(n)})\}$ is an optimal solution to the subproblem that requires us to pair optimally $A - \{a_{\sigma(1)}\}$,$B - \{b_{\sigma(1)}\}$. **Proof:** Note that the objective function on P can be rewritten as follow

$$\sum_{i=2}^{n} \left|a_{\sigma(i)} - b_{\sigma(i)}\right| + \left|a_{\sigma}(1) - b_{\sigma(1)}\right| = \sum_{i=1}^{n} \left|a_{\sigma(i)} - b_{\sigma(i)}\right| \leq \sum_{i=1}^{n} |a_i - b_i| \tag{1}$$

Once equation (1) is established, the proof is by contradiction. Assume that Q is not optimal for the subproblem. That means that another solution Q' exists such that $\sum_{Q'} |a_i - b_i| < \sum_{Q} |a_i - b_i|$.

Then, because of (1) we could create a new solution $O' = Q' + \{(a_{\sigma(1)}, b_{\sigma(1)})\}$ for which the objective function would be lower than the objective function on the pairs in O. This creates a contradiction, therefore Q must be optimal for P.

Opposite direction: let us prove that if Q is an an optimal solution to the subproblem that requires us to pair optimally $A - \{a_{\sigma(1)}\}$,$B - \{b_{\sigma(1)}\}$ then $Q \cup \{(a_{\sigma(1)}, b_{\sigma(1)})\}$ is optimal for the original problem that require us to pair A, B. **Proof:** Once equation (1) is established, the proof is by contradiction. If $O = Q \cup \{(a_{\sigma(1)}, b_{\sigma(1)})\}$ is not optimal for A, B, then there is another solution $O' = Q' \cup \{(a_{\sigma(1)}, b_{\sigma(1)})\}$ for A, B such that $\sum_{O'} |a_i - b_i| < \sum_O |a_i - b_i|$. Then, we could use Q' as a solution for $(A - \{a_{\sigma(1)}\}$,$B - \{b_{\sigma(1)}\})$ and $\sum_{Q'} |a_i - b_i| < \sum_Q |a_i - b_i|$, which is a contradiction (because Q is optimal).

Problem 4. (25 points)

Suppose a data file contains a sequence of 8-bit characters, such that all the 256 characters are about as common. More precisely, the maximum character frequency is less than twice the minimum character frequency. Explain why Huffman coding in this case is no more effective than using an ordinary 8-bit fixed-length code.

Answer: Let's prove that this set of frequencies produces a balanced binary code tree.

Let the frequencies (in increasing order) be $f_1, f_2, \ldots, f_{256}$. We know that $f_{256} < 2f_1$. Thus, when the first node is combined by the Huffman algorithm, it has frequency $g_1 = f_1 + f_2 > 2f_1 > f_{256}$. Thus the new list in order is $f_3, f_4, \ldots, f_{256}, g1$. The second node will have frequency $g_2 = f_3 + f_4 > 2f_3 > g_1$. In this way, the list will become a list of nodes with two leaves. It can also be shown that the initial frequency property is preserved: $G_{128} = f_{255} + f_{256} < 2f_{256} < 4f_1 < 2f_1 + 2f_2 = 2g_1$

Now we assume that list has frequencies (in increasing order) $f_1, f_2, \ldots, f_{2n}$ and all nodes corresponding to the frequencies of the list are balanced trees. The exact same argument as above will show that, after the next pass of the algorithm, a new list $g_1, \ldots, g_n$ will be produced, and will consist of balanced binary trees.

The two statements combined show that the final result is a balanced binary tree. Thus, each code word will have length 8. Huffman does no better than an 8-bit fixed length code (in fact it IS an 8-bit fixed length code).